Michael Tischer

LA BIBLE PC

EDITIONS MICRO APPLICATION

Auteur Michael Tischer

Traducteurs Hassina Abbasbhay, Christophe Stehly & Fawzi Benachera

ISBN : 2-86899-661-2

Collection dirigée par Philippe Olivier
Edition réalisée par Frédérique Beaudonnet

Préface

Quand j'ai décidé de réaliser la troisième édition de cet ouvrage monumental, j'étais conscient de l'ampleur du travail qui m'attendait. Aujourd'hui, l'informatique, et plus particulièrement la micro-informatique, évolue vite, très vite. Vous avez certainement été frappé par la rapidité avec laquelle les éditeurs de logiciels annoncent les versions successives de leurs produits.

L'informatique a connu depuis deux ans des bouleversements importants dans l'aspect matériel des ordinateurs et par corollaire dans la programmation système. Il y a très peu de temps, le terme PC ne représentait qu'un simple système équipé du processeur 8088, un bus grossier, un BIOS en ROM, 256 Ko de RAM et une carte texte monochrome. Aujourd'hui, on ne parle plus que de processeurs 32 bits intelligents, de bus MCA ou EISA, de cartes graphiques SVGA, de VESA, etc. La liste serait longue pour décrire ici toutes ces formidables évolutions.

Cette nouvelle approche d'ordinateurs graphiques et puissants, accessibles à un nombre sans cesse croissant d'utilisateurs, a modifié la donne dans les installations lourdes. On ne peut aujourd'hui imaginer un système informatique sans PC, que celui-ci constitue le serveur, un poste de travail ou simplement un terminal. Et le mot qui nous vient à l'esprit est bien entendu "réseau". Ce terme si vaste, si effrayant pour certains, fait partie aujourd'hui de notre vocabulaire ; il faut savoir l'intégrer et le maîtriser.

C'est dans ce contexte que j'ai écrit la troisième édition de la Bible. Pour donner à tous les programmeurs les moyens de comprendre et d'exploiter les évolutions matérielles, système et logicielles des PC. Pour leur permettre d'aller plus haut, plus loin, ne pas se cantonner à des compatibilités vieillissantes, ne pas confondre ni réduire une carte SVGA à une simple carte VGA. Pour leur donner une longueur d'avance sur toutes les modifications introduites par le DOS 5.0 et Windows, évoluer dès maintenant de la technologie VCPI vers DPMI. Pour analyser le fonctionnement des DOS Extenders et surtout mettre en oeuvre les mécanismes du mode protégé.

Ce que j'ai voulu réaliser avant tout, c'est un livre, un vrai livre qui apporte la lumière sur toute la programmation de tous les composants système. J'ai écrit le maximum d'exemples pour que le lecteur puisse mettre en pratique toutes les techniques de programmation. Et enfin, les annexes donnent la liste commentée des interruptions et fonctions nécessaires.

Il ne me reste plus qu'à vous souhaiter une bonne lecture.

Michael Tischer

Sommaire

Annexes

1. Le PC et la programmation système

La première partie de cet ouvrage est consacrée aux fondements de la programmation système. Il y sera question de sa nature, de ses buts, de ses méthodes et de ses outils. Les différentes parties du PC et les rôles respectifs du matériel, du BIOS et de DOS seront soigneusement étudiés. Mais pour commencer, il faut d'abord se demander ce qu'est la programmation système.

1.1. Qu'est-ce que la programmation système ?

Lorsque l'on débute sur PC et que l'on pose cette question, on est amené à entendre toutes sortes de réponses parfois fort curieuses. Les uns tiennent la programmation système pour une philosophie, une technique de programmation qui montre comment passer d'un projet à sa réalisation concrète. Ce point de vue est erroné. D'autres soutiennent que la programmation système existe parce que des programmes développés sur un certain système ne marchent pas sur les autres. Ce n'est pas tout à fait faux, mais nous ne trouvons pas encore au coeur de la vérité.

A mon avis, il est plus facile de définir la programmation système par rapport à la programmation des applications. Programmer des applications revient à manipuler des informations, à les ranger dans des listes, des arbres et à les traiter de diverses façons. Les algorithmes qui interviennent sont indépendants du système d'exploitation, ils peuvent être repris pour tous les ordinateurs existants (de type machine de Turing).

Par contre la façon dont les données parviennent à l'"intérieur" du programme et dont ils ressortent après traitement dépend du système. C'est là l'affaire de la programmation système, dont on peut ainsi définir le domaine : l'accès aux fichiers, le clavier, l'écran et tous les dispositifs qui introduisent ou font sortir des informations de l'ordinateur.

Comme chaque programme communique nécessairement avec le monde extérieur, la programmation système en recouvre nécessairement un aspect, mais l'inverse est également vrai : un programme système comporte forcément des traitements qui le classent en partie dans la catégorie des applications. Il paraît donc évident que la programmation d'un micro-ordinateur ne peut pas être uniquement de la programmation d'application, elle comporte inévitablement une part de programmation système.

Il n'est pas juste de réduire la programmation système à l'interrogation du clavier et l'affichage sur écran. La programamtion système se préoccupe de l'accès à toutes les ressources du PC : en plus du clavier, de l'écran et de tous les périphériques il faut aussi prendre en compte la mémoire vive et le processeur. Ce n'est pas seulement la programmation du matériel qui est visée mais aussi la collaboration avec DOS et le BIOS en ROM. Nous aborderons ce dernier sujet dans le prochain chapitre.

1.2. Le modèle à trois couches

L'une des tâches fondamentales de la programmation système consiste à accéder au matériel constituant le PC, comme nous venons de le voir. Cet accès ne se fait pas nécessairement de manière directe : ainsi les programmes système n'écrivent pas forcément dans les circuits d'une carte vidéo ("contrôleur vidéo"). La manière usuelle de procéder consiste à utiliser un intermédiaire qui propose des services spécialisés. Cet intermédiaire peut être le BIOS ou DOS, qui sont justement des interfaces logicielles créées pour gérer le matériel.

Avantages des interfaces de DOS et du BIOS

Le grand avantage pour le programme est qu'il n'est pas obligé de "se salir les mains" car il n'entre pas en contact avec l'électronique. Il se contente d'appeler une routine du BIOS qui prend en charge le travail demandé, par exemple en testant si une touche du clavier a été enfoncée et le cas échéant en indiquant de laquelle il s'agit.

Le programmeur évite beaucoup de peine car programmer un appel de fonction est en général infiniment plus simple que de prévoir un accès direct au matériel. Mais ce n'est pas le seul avantage.

Car les interfaces du BIOS et de DOS prennent également soin d'isoler les programmes des caractéristiques physiques du matréiel. C'est ainsi que les cartes graphiques monochromes (MDA, Hercules) se commandent d'une toute autre façon que les cartes graphiques couleur (CGA, EGA, VGA, super VGA). Si un programme devait supporter toutes ces cartes, il lui faudrait prévoir des routines pour tous les standards vidéo, ce qui représenterait un surcroît de travail considérable. Mais les fonctions du BIOS qui se chargent de l'affichage sont paramétrées pour exploiter la carte installée, de sorte qu'elles peuvent être appelées par un programme sans que ce dernier ait à se faire du souci à propos du type de carte en service.

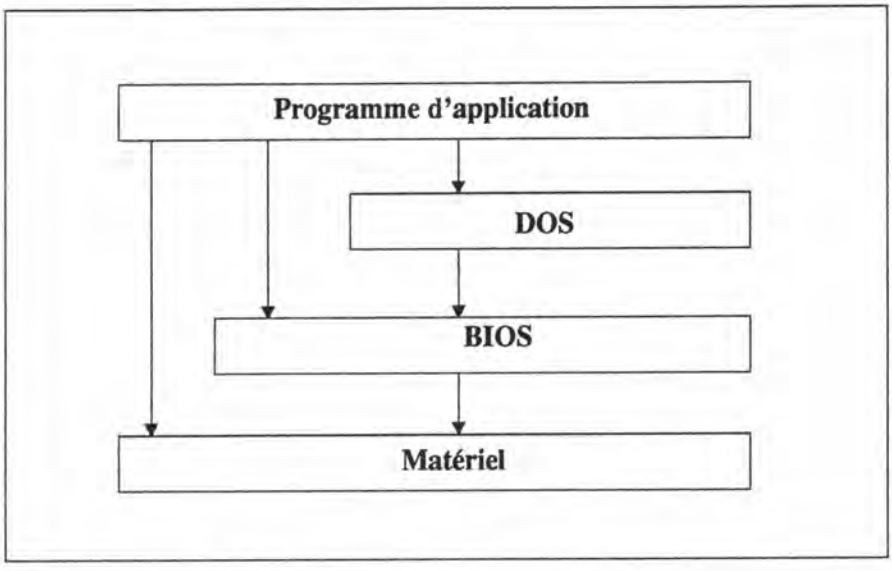
Le BIOS

Comme le montre l'illustration suivante, le BIOS peut être considéré comme une couche située juste au-dessus du matériel. Il permet d'accéder aux composants suivants :

- ✓ carte vidéo
- ✓ mémoire vive (étendue)
- ✓ disquettes
- ✓ disques durs
- ✓ interfaces série
- ✓ interfaces parallèles

- ✓ clavier
- ✓ horloge CMOS

Un programme pourrait court-circuiter le BIOS en accédant directement au matériel, mais dans la plupart des cas il gagne à se servir des fonctions du BIOS dont les services sont standardisés et identiques dans tous les PC. Par ailleurs le BIOS se trouve en ROM et cette ROM qui fait partie des circuits de la carte mère du PC est immédiatement accessible dès le démarrage du PC. Nous étudierons le BIOS et ses fonctions au chapitre 3.



Le modèle à trois couches

L'interface DOS

Tout comme le BIOS, le DOS propose également des fonctions d'accès au matériel. Elles présentent cependant un autre caractère que les fonctions du BIOS car elles considèrent le matériel non pas sous son angle physique mais comme un dispositif logique. Dans le cas des disquettes et des disques durs, le phénomène est particulièrement évident : alors que le BIOS raisonne en pistes et en secteurs, le DOS se place au niveau des fichiers et des sous-répertoires.

On peut ainsi s'adresser au DOS en lui demandant d'ouvrir tel fichier et d'en extraire les 1000 premiers caractères sans se préocupper de l'emplacement du fichier sur le disque. Il n'est même pas utile de connaître le type de la mémoire de masse (disquette, disque, serveur de réseau), il suffit de désigner le périphérique par une lettre (comme A: ou C:) pour que le DOS sache à quoi il aura affaire.

Il est vrai que l'accès ne sera pas uniquement régi par le DOS : le BIOS sera impliqué dans le mécanisme. Il se pourrait dans certaines situations que le DOS accède directement au matériel mais peu importe pour le programmeur.

Le chapitre 3 vous montrera comment appeler les fonctions du BIOS et du DOS. Mais quelle est la meilleure voie pour accéder au matériel : faut-il user de la programmation directe, faire appel aux fonctions du BIOS ou à celles du DOS ?

Le choix du service

Il faut d'abord bien se rendre compte que l'on n'a pas toujours le choix entre programmation directe du matériel, et fonctions du BIOS ou du DOS. Il existe toute une série de tâches qui ne sont couvertes ni par le BIOS ni par le DOS. Si vous voulez tracer des cercles ou des droites dans le mode graphique de votre carte vidéo, vous pouvez toujours chercher la fonction du BIOS ou du DOS correspondante : elle n'existe pas! Il faut alors piloter soi-même le matériel ou recourir à des bibliothèques du commerce qui sont spécialisées dans ce domaine.

Lorsque des fonctions du BIOS et du DOS se disputent la faveur du programmeur, la décision se fera en fonction de l'application concernée. Pour travailler avec des fichiers, l'usage du DOS est inévitable mais pour formater une disquette ou pour écrire un caractère sur l'écran c'est le passage par le BIOS qui est obligatoire. Si vous désirez que les sorties de votre programme puissent être détournées de l'écran sur un fichier (comme dans la commande DOS dir > liste.txt), vous devrez recourir aux fonctions de DOS qui sont les seules à gérer ce mécanisme. Mais les fonctions du BIOS sont plus sérieuses dans le contrôle de l'affichage, le DOS n'étant même pas capable de positionner librement le curseur. Vous voyez que le choix s'opère au cas par cas.

Dans certains cas la vitesse d'exécution constitue un inconvénient majeur des fonctions du BIOS ou de DOS. Car plus il y a de couches à traverser pour convertir l'intention d'un programme (par exemple l'affichage d'un caractère) en accès au matériel, plus l'exécution se trouvera ralentie. Lors de la lecture d'un fichier, la vitesse de transfert des données issues du disque dur est réduite de 80 % lorsqu'on passe de la commande directe du matériel au programme à travers les fonctions du BIOS et du DOS.

Le ralentissement est dû à la gestion des différentes couches. Avant de transmettre le contrôle à une autre couche, il faut convertir de nombreux paramètres, charger des informations dans des tables internes, constituer des mémoires tampons, etc... Le temps perdu de cette façon est appelé "overhead", plus il est important, plus il dégrade l'humeur du programmeur.

Lorsque la vitesse d'exécution est un facteur crucial et que la programmation directe du matériel n'est pas trop difficile, on s'y résoud finalement en laissant de côté les fonctions du BIOS ou du DOS. L'affichage en mode texte constitue un bon exemple. Pratiquement tous les programmes commerciaux s'engagent dans cette voie car dans ce domaine les

fonctions du BIOS et du DOS manquent de souplesse et de vélocité. Comme nous le verrons au chapitre 4, il est relativement facile de commander une carte vidéo en mode texte. En mode graphique par contre c'est une autre paire de manches.

Nous allons examiner comment on appelle les services de DOS et du BIOS et comment on commande directement le matériel. Mais auparavant il nous faut jeter un coup d'oeil attentif sur les composants du PC.

1.3. Compréhension de base du matériel

Lorsqu'on se lance dans la programmation système, on est confronté à l'architecture de base du PC. Une compréhension minimale du matériel est obligatoire pour assimiler de nombreuses questions abordées dans cet ouvrage.

1.3.1. La naissance du micro-ordinateur

Le micro-ordinateur est né à une époque où beaucoup de choses qui paraissent naturelles aujourd'hui n'étaient même pas imaginables. L'idée d'offrir un miniordinateur disponible sur un simple bureau n'était pas nouvelle mais elle ne pouvait être réalisée que par des firmes d'une envergure comparable à IBM. IBM, justement, venait d'achever le projet d'un système appelé System 23 / Data Master mais ce Data Master était équipé d'un processeur 8 bits 8085 d'Intel, un composant déjà dépassé à l'heure où apparaissaient les unités centrales à 16 bits. Il s'agissait alors de préparer rapidement une machine nouvelle et révolutionnaire.

Choix du processeur

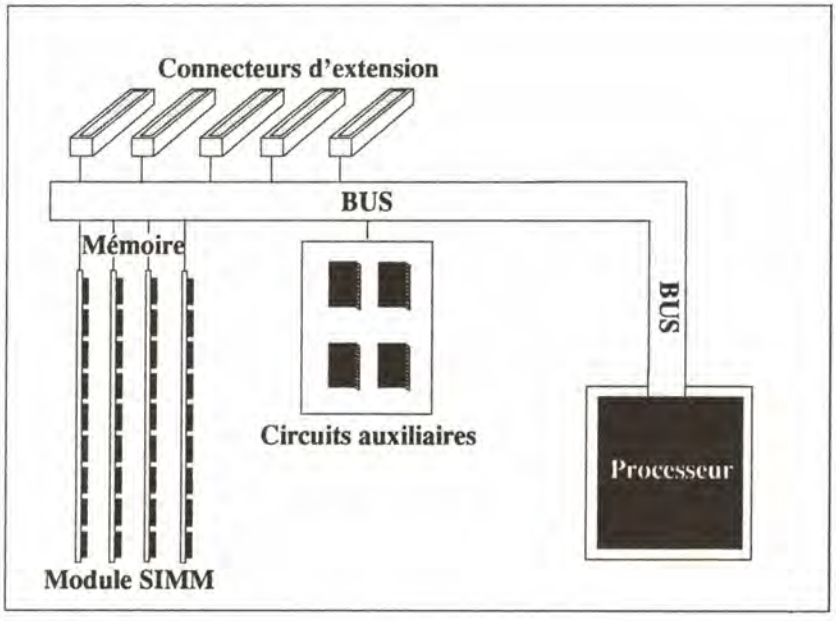
Les premiers représentants de la nouvelle classe des 16 bits étaient les processeurs 8086 et 8088 d'Intel. L'un et l'autre disposaient de registres de 16 bits et pouvaient adresser, non plus 64 Ko, mais 1 Mo de mémoire. Cette capacité paraissait alors démesurée, un peu comme nous percevons une RAM de 1 Go aujourd'hui.

Elle ne constituait pas la seule raison qui conduisit au choix de ces processeurs. D'autres circonstances jouèrent un rôle important. L'équipe de développement travaillait dans le cadre d'un planning très serré et les processeurs en question disposaient déjà d'une gamme de circuits auxiliaires compatibles. Ces circuits étaient indispensables car l'intégration des fonctions n'était pas aussi développée que maintenant. Par ailleurs, une jeune société particulièrement dynamique, Microsoft, avait mis au point un système d'exploitation et un interpréteur BASIC pour les deux processeurs.

Les responsables optèrent pour le 8088 qui par rapport à son concurrent, le 8086, avait l'avantage de travailler en 16 bits au niveau interne, tout en communiquant avec

l'extérieur par un bus de données à 8 bits. Pour ce qui était du bus de données à 8 bits, on disposait déjà de celui du Data Master.

Ce bus relie la carte mère du PC, qui contient le processeur et ses circuits auxiliaires, à la mémoire et aux cartes d'extension engagées dans les connecteurs. C'est ici le moment de le regarder de plus près.



Architecture d'un PC

1.3.2. Le bus

Bien que le bus joue un rôle tout à fait élémentaire dans le fonctionnement d'un ordinateur, son développement constitue un des chapitres les plus sombres de l'histoire de la micro-informatique. Tandis qu'IBM s'efforçait de réaliser un système ouvert et d'en publier tous les aspects techniques, la documentation des signaux du bus fut négligée, probablement parce qu'on pensait qu'elle n'intéressait personne.

Tel ne fut pas le cas car l'un des facteurs qui assura le succès du PC fut son ouverture et la possibilité d'y intégrer sans problème des cartes d'extension du matériel. De nombreux utilisateurs exploitèrent cette possibilité en achetant non seulement des cartes d'extension fabriquées par IBM mais aussi des cartes en provenance d'autres fournisseurs. Car le PC rend accessible à l'extérieur tous ses bus de données et d'adresses en y raccordant la mémoire vive, les cartes d'extensions et certains circuits auxiliaires.

Fonctionnement du bus PC

Le bus n'est au fond rien d'autre qu'un câble de 62 lignes qui permet de faire passer des données du processeur à la mémoire et vice-versa. Il constitue une sorte d'autoroute qui traverse le PC, réglementée par le processeur.

On y distingue plus précisément deux voies : le bus de données et le bus d'adresses. A chaque accès mémoire, le processeur dépose sur le bus d'adresses l'adresse de la mémoire visée. Chaque ligne a une signification binaire, elle peut être à 0 ou à 1 et l'ensemble des lignes forme un nombre qui désigne une mémoire (=une adresse). Plus le nombre de lignes est important, plus le nombre d'adresses accessibles augmente. A l'origine, le bus d'adresses comportait 20 lignes car avec ce nombre il est possible d'adresser 1 Mo de mémoire, ce qui correspondait à la capacité du processeur.

Les données proprement dites ne sont pas émises sur le bus des adresses mais sur le bus des données qui au départ ne comportait que 8 lignes, ce qui autorise la transmission d'un octet à la fois. Pour transférer en mémoire le contenu d'un registre de 16 bits ou un nombre exprimé avec 16 bits, il fallait opérer en deux étapes de 1 octet.

Si en théorie, les choses paraissent presque simples, elles le sont beaucoup moins dans la pratique. Les lignes d'adresses et de données sont complétées par des lignes transportant des signaux de synchronisation entre le processeur et la mémoire. En fin de compte, toutes les cartes sont à l'écoute du bus et lors de l'émission d'un signal approprié du processeur il doit s'en trouver une qui se déclare concernée par l'adresse en cause. Les autres cartes se coupent alors de la suite de la communication : elles se rebrancheront lors du cycle de transfert suivant, en espérant qu'elles seront cette fois concernées.

Ce mécanisme explique pourquoi l'installation de cartes d'extension provoque parfois des problèmes de conflit : il arrive que deux cartes revendiquent le même domaine d'adresses. Le problème se résoud en maniant les interrupteurs DIP situés sur les cartes et qui permettent justement de fixer leur adressage. L'une des cartes doit céder la place à l'autre en acceptant d'être reconfigurée.

Heureusement le programmeur système n'entre pas en contact avec les signaux du bus, et même d'une façon générale on peut dire que le fonctionnement du bus ne joue aucun rôle dans la programmation système, exception faite de quelques considérations marginales.

La connaissance de la synchronisation des signaux du bus est par contre très importante pour les fabricants de cartes d'extension qui doivent respecter les protocoles en vigueur sous peine de voir défaillir les ordinateurs. Pourtant IBM n'a jamais publié ce protocole . Les fabricants furent obligés de tester en profondeur les cartes existantes pour pouvoir les reproduire. Dans ces conditions il n'est pas étonnant que de temps à autre certaines d'entre elles dépassant les tolérances admissibles aient rencontré des problèmes.

Le bus At

Ce n'est qu'en 1991 qu'un standard international en la matière fut publié par l'institut américain IEEE (Institute of Electrical and Electronic Engineers) mais il ne concerne que le bus AT. Car le bus du PC a connu une évolution. Si tout à l'heure nous l'avons comparé à une autoroute, il ne faut pas oublier cependant que ses voies étaient limitées à 8 bits. Lorsque l'AT fut introduit, il fut doté d'un nouveau bus de 16 bits de large, tout en demeurant compatible avec son prédécesseur. Les anciennes cartes de 8 bits pouvaient coexister avec les nouvelles cartes de 16 bits à l'intérieur d'un AT. Bien entendu les cartes de 16 bits sont plus performantes, elles traitent deux fois plus de données par unité de temps que les 8 bits.

Le bus d'adresses fut également amélioré : son nombre de lignes fut porté à 24, de sorte que l'AT est capable d'adresser 16 Mo de mémoire. Par ailleurs la fréquence des signaux de contrôle fut augmentée, ce qui accélérait l'ensemble des échanges sur le bus. Alors que le PC de base fonctionnait à 4,77 MHz, l'AT affichait à ses débuts une cadence de 8 MHz qui augmenta progressivement pour atteindre aujourd'hui 50 MHz sur les modèles les plus performants. En conséquence, le bus finit par représenter un goulot d'étranglement, les données restant toujours à la traîne entre la mémoire et le processeur. Les nouveaux disques durs ont maintenant des vitesses de transfert supérieures à celle du bus.

Les cartes d'extension durent être munies de freins de secours : les "wait state signals" qui permettent aux composants trop lents de fournir à leur rythme des données au processeur.

Le bus AT a aujourd'hui trouvé ses successeurs : le Micro Channel et le bus Eisa qui sont plus puissants mais ne se sont pas encore imposés sur le marché. Jusqu'à leur apparition, le bus AT n'avait même pas de nom précis, c'est pourquoi face à la concurrence on le baptisa à la hâte bus ISA (Industrie Standard Architecture).

Les problèmes posés par les cartes de 16 bits sur le bus de l'AT

La plupart des AT 386 et 486 étant fournis avec un bus ISA, le monde du PC se débat encore dans quelques problèmes suscités par le bus originel de l'AT. La coexistence de cartes d'extension de 8 bits et de 16 bits se fait difficile lorsque les adresses attribuées à ces cartes se trouvent dans une même zone de 128 Ko. Le dilemme commence lorsqu'une carte de 16 bits signale sur une ligne de contrôle , au début d'un transfert de données, qu'elle est en mesure de prélever un mot de 16 bits sur le bus, n'étant pas obligée comme une carte de 8 bits de faire un découpage en deux octets.

Le signal en question doit être émis à un moment où la carte ne sait pas encore qu'elle est adressée. Car sur les 24 lignes du bus d'adresses, seules les lignes A17 à A23 sont initialisées à ce moment précis : la carte ne connaît donc que les bits 17 à 23 de l'adresse.

Ces bits recouvrent une zone de 128 Ko, indépendamment du contenu des bits 0 à 16 qui vont suivre. La carte sait simplement que l'adresse se trouve entre 0 et 127 Ko, ou 128 et 256 Ko , etc...

Si à ce moment la carte de 16 bits envoie le signal pour demander une transmission sur 16 bits, elle se fait l'interprète de toutes les cartes situées dans la même zone de mémoire. Celles-ci en ressentent le contrecoup à l'instant d'après car les bits 0 à 16 vont arriver sur le bus d'adresses, ce qui va déterminer laquelle des cartes était véritablement concernée. S'il s'agit effectivement d'une carte de 16 bits, tout est parfait. Mais si c'était une carte de 8 bits qui était visée, la carte de 16 bits se déconnecte de la communication et laisse à son destin la carte de 8 bits. Mais cette dernière, ne fonctionnant qu'avec 8 bits, ne pourra pas se débrouiller correctement avec les informations reçues. Une erreur va donc se produire dans son fonctionnement.

1.3.3. Les circuits auxiliaires

Le processeur est entouré de petits circuits complémentaires dont la fonctionnalité demeure, même si de nos jours ils sont fabriqués par d'autres fournisseurs et s'ils sont davantage intégrés jusqu'à parfois ne faire qu'un avec lui. Ces cicuits sont parfois appelés des contrôleurs car ils "contrôlent" (commandent) une partie du matériel par délégation du processeur qui se décharge ainsi d'une partie de son travail. En contrepartie, le programme en cours s'exécute plus rapidement du point de vue de l'utilisateur.

Les paragraphes suivants décrivent les différents contrôleurs en mentionnant le type de circuit mis en service à l'origine par IBM. Lorsqu'ils sont programmables et que leur programmation est utile sous l'angle de la programmation système, leur étude est reprise plus loin dans ce livre, en liaison avec le matériel associé.

Le contrôleur DMA (8237)

DMA veut dire Direct Memory Access, un acronyme qui décrit une technique de transfert des données. Dans cette technique, les données issues d'un périphérique (par exemple d'un disque dur ou d'un lecteur de disquette) sont canalisées directement dans la mémoire. Dans les conditions de fonctionnement classique, le processeur réclame les données au périphérique octet par octet, ou mot par mot, avant de les envoyer dans la RAM. Lié au rythme du bus, l'accès DMA fait gagner du temps lorsque le processeur est lent. Mais les processeurs les plus modernes travaillent aujourd'hui cinq fois plus vite que le bus : l'avantage de l'accès DMA disparaît en conséquence, sans oublier que le circuit en question est plutôt obsolète. Il serait intéressant qu'il puisse déplacer de grandes quantités de données de la mémoire centrale dans la mémoire d'écran, mais là encore il est trop lent.

Malgré tout le contrôleur DMA est toujours présent dans les PC. Il est vrai qu'il ne remplit plus du tout sa fonction d'origine qui consistait à accélérer les transferts entre disque(tte)

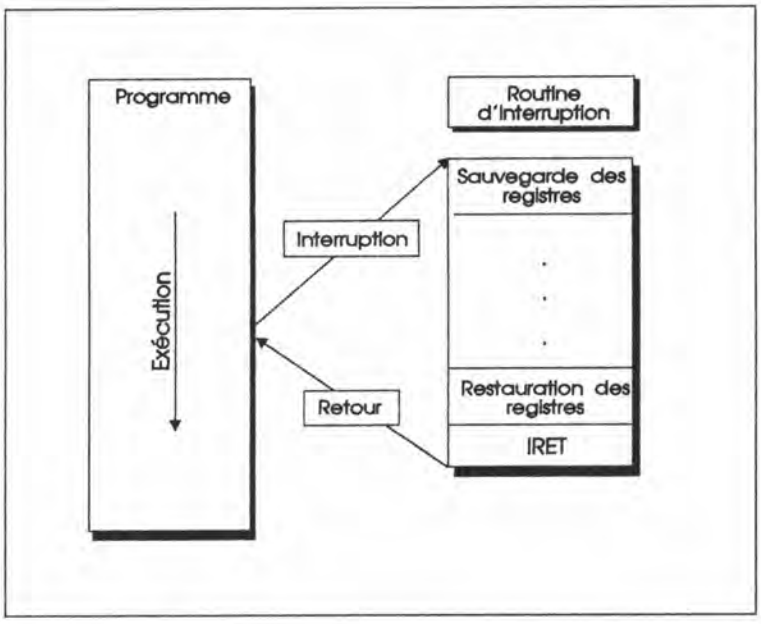
et mémoire centrale. L'AT en possède même deux. Son nouveau rôle est de rafraîchir la mémoire vive (processus dit de Ram Refresh).

Les micro-ordinateurs actuels sont en effet équipés de mémoire dynamique (DRAM, dynamic RAM) par opposition à la mémoire statique (SRAM, static RAM). La mémoire dynamique perd ses informations au bout de quelques fractions de seconde, il faut la soumettre à une recharge électrique incessante. Ce rafraîchissement est pris en charge par le contrôleur de DMA qui décharge ainsi le processeur d'une besogne harassante.

Le contrôleur d'interruption (8259)

Le contrôleur d'interruption joue un rôle central dans la commande des périphériques comme le clavier, le disque dur ou l'interface série. En théorie, un clavier devrait être sans cesse interrogé par le processeur pour que toute frappe puisse être immédiatement prise en compte. Mais cette scrutation permanente ("polling") consommerait des ressources précieuses, car la plupart du temps elle ne servirait qu'à constater qu'il ne s'est rien passé, l'utilisateur ne pouvant taper des informations au rythme de travail de l'ordinateur. Si on ralentit la fréquence d'interrogation du clavier, on diminue la vitesse de réaction des programmes, ce qui n'est pas non plus intéressant.

Une autre voie a donc été choisie. Ce n'est plus le processeur qui interroge les périphériques mais les périphériques qui se signalent à lui lorsqu'une information doit être traitée. Le mécanisme porte le nom d'interruption matérielle : le processeur interrompt en effet le programme en cours pour déclencher une routine appelée gestionnaire d'interruption, généralement mise à disposition par le BIOS et qui est capable de traiter la demande émise. Quand cette routine est achevée, le processeur poursuit l'exécution du programme interrompu, comme si rien ne s'était passé. On voit donc que le processeur n'est mis à contribution qu'en cas de besoin.



Interruption d'un programme

Entre le déclenchement de l'interruption et sa prise en compte par le processeur et le gestionnaire spécialisé, le cheminement est assez long. Les demandes d'interruption des cartes d'extension et des circuits auxiliaires ne sont pas envoyées directement au processeur mais parviennent d'abord au contrôleur d'interruption. Il ne faut pas oublier que le PC dispose de plusieurs lignes d'interruption reliées chacune à un périphérique : les interruptions peuvent donc survenir simultanément de plusieurs provenances.

Mais le processeur ne peut en traiter qu'une à la fois. Il faut donc définir et gérer des priorités. C'est là précisément la tâche du contrôleur d'interruption.

Dans un PC/XT il est à même de traiter jusqu'à 8 sources d'interruption c'est-à-dire 8 requêtes en même temps. Pour un AT cette capacité n'est pas suffisante : on y a donc installé deux contrôleurs qui peuvent gérer 15 interruptions simultanées. Les interruptions matérielles sont étudiées au chapitre 2 exclusivement consacré à ce sujet.

Interface de périphérie (8255)

Comme son nom l'indique, l'interface de périphérie établit la liaison entre l'unité centrale et les périphériques tels que le clavier ou le haut-parleur. Elle se contente en fait de jouer un rôle d'intermédiaire exploité par le processeur lorsqu'il désire accéder à un périphérique pour lui transmettre des signaux. Vous ferez plus amplement connaissance avec ce circuit au chapitre 13 lorsqu'il sera question de produire des sons.

L'horloge (8248)

Si on compare le microprocesseur à un cerveau, l'horloge peut être considérée comme le coeur du système informatique. Ce coeur bat à une vitesse de plusieurs millions de pulsations par seconde (14,3228 Megahertz très précisément) et il anime ainsi le microprocesseur et les autres composants du système. Comme aucun de ces circuits ne supporte une fréquence d'horloge aussi élevée, celle-ci est divisée au contact de chacun d'eux, en fonction de son rythme propre.

Le temporisateur (8253)

Le circuit appelé temporisateur ou Timer peut être exploité comme compteur ou comme chronomètre. Il possède en effet la faculté d'émettre des impulsions à intervalles fixes sur l'un de ses canaux de sortie. La fréquence à laquelle ces impulsions se répètent n'est pas définie une fois pour toutes, elle est paramétrable par logiciel. On la fixe par rapport à la cadence de l'horloge à laquelle le temporisateur est relié par une ligne spéciale.

Le temporisateur dispose de différents canaux de sortie qui peuvent présenter des fréquences variées. Chacun de ces canaux de sortie est relié à un périphérique différent. C'est ainsi par exemple qu'une liaison avec le contrôleur DMA impose le rythme de réfraîchissement de la mémoire dynamique. D'autres canaux conduisent au haut-parleur et au contrôleur d'interruption. Ce dernier déclenche lors de chaque impulsion reçue sur "son" canal une interruption matérielle qui reprise par le BIOS sert à faire "avancer" l'heure sur le PC.

Le contrôleur d'écran

Contrairement aux circuits présentés jusqu'ici, le contrôleur d'écran ne fait pas partie de la carte mère du PC mais de la carte vidéo (ou adaptateur graphique) qui est enfichée dans l'un des connecteurs.

A l'origine ce contrôleur était du type Motorola 6845 et constituait le coeur des cartes CGA et MDA qui furent les premières sur le marché de l'IBM PC. Ces cartes disparurent peu à peu, relayées par les cartes EGA puis VGA qui possèdent des contrôleurs plus évolués et plus puissants. Les nouvelles cartes ne sont plus compatibles avec le contrôleur Motorola, mais le processeur n'en fait pas beaucoup de cas car il n'entre pas directement en contact avec le contrôleur vidéo. Ce fardeau est supporté par le BIOS qui est spécialement prévu pour supporter toute une gamme de contrôleurs. Ce point sera étudié au chapitre 4 consacré exclusivement à la programmation des cartes vidéos d'écran.

Le contrôleur de disquette (765)

Ce circuit immatriculé NEC PD765 ne se trouve pas non plus sur la carte mère du PC mais sur une carte d'extension qui commande le(s) lecteur(s) de disquette. Autre similitude avec le contrôleur d'écran : le processeur ne s'adresse pas directement à lui mais délègue cette tâche au BIOS.

Les premiers modèles de PC n'étaient pas munis de lecteurs de disquettes mais disposaient d'une prise pour raccorder un magnétophone à cassette, support de données prévu par IBM. Une fois que les lecteurs de disquettes furent disponibles, les utilisateurs abandonnèrent très vite leurs magnétophones pour des raisons évidentes de confort, de sécurité et surtout de rapidité.

Le chapitre 6 est spécialement dédié aux disquettes, aux disques durs et à leurs contrôleurs.

Les coprocesseurs arihmétiques (8087/80287/80387)

Last but not least, un PC comporte à côté du processeur un socle vide destiné à recevoir un coprocesseur arithmétique.

Comme ni le 8088 ni ses successeurs (à l'exception du 80486) ne sont capables de traiter ce qu'on appelle les nombres à virgule flottante ou nombres réels, Intel a développé dès l'origine des circuits spéciaux pour collaborer avec les processeurs dans le domaine du calcul numérique. Ces coprocesseurs ont été adaptés de génération en génération aux différents successeurs du 8088, de sorte qu'il existe un coprocesseur adapté à chaque microprocesseur. Ils accélèrent considérablement la vitesse d'exécution de certains logiciels (notamment des logiciels de CAO) lorsque les calculs numériques sont nombreux et complexes.

Pourtant les coprocesseurs n'ont jamais connu de grand succès dans le monde du PC, probablement parce que le prix demandé par Intel les rangeait dans la catégorie des accessoires de luxe.

La programmation des coprocesseurs ne fait pas partie des thèmes abordés dans cet ouvrage car elle est pas du ressort de la programmation système. Il s'agit simplement d'une programmation ordinaire en langage machine. Malgré tout le chapitre 4 vous emmènera faire un petit tour dans le monde des coprocesseurs, lorsqu'il s'agira de déterminer si un tel circuit est installé dans un PC.

1.3.4. Organisation de la mémoire

Au moment de sa naissance, le PC était équipé de 16 Ko que l'on pouvait étendre à 64 Ko sur la carte mère. Par ailleurs IBM fournissait des cartes d'extension mémoire de 64 Ko que l'on pouvait placer dans les cinq connecteurs d'extension disponibles. Le nombre maximal de cartes susceptibles d'être installées était de 3, ce qui donnait une capacité totale de 256 Ko, un nombre que l'on considérait comme gigantesque à l'époque.

Les développeurs du PC savaient pourtant qu'une évolution serait inévitable. Ils ont donc organisé la mémoire de façon à à en permettre l'extension jusqu'à une limite de 640 Ko. Malgré ses précautions, l'avenir les a en quelque bien vite rattrapés, comme le sait aujourd'hui tout utilisateur de DOS. L'espace d'adressage complet d'un 8088 allant jusqu'à 1 Mo, la place restante fut attribuée à la mémoire d'écran des cartes vidéo, à la ROM prévue pour le BIOS ainsi qu'à d'autres ROM complémentaires. Notez que le processeur se soucie peu de savoir si un emplacement mémoire est occupé par des RAM ou des ROM pour peu évidemment que l'on n'essaie pas d'écrire des informations dans les ROM. Le processeur ne se préoccupe pas non plus de savoir si la mémoire à laquelle il accède est physiquement présente. Si la capacité est de 1 Mo, cela ne veut pas dire qu'il y ait vraiment un circuit de ROM ou de RAM à chaque adresse.

Comme le montre le tableau suivant, la structure de la mémoire repose sur un découpage en segments de 64 Ko. Le 8088 et ses successeurs gèrent en effet la mémoire par blocs de cette taille, comme le montrera encore le chapitre 22. L'espace mémoire complet est constitué par seize de ces blocs.

Organisation de la mémoire du PC

Bloc	Adresse	Contenu
15	F000:0000 - F000:FFFF	BIOS
14	E000:0000 - E000:FFFF	libre pour insertion de ROM
13	D000:0000 - D000:FFFF	libre pour insertion de ROM
12	C000:0000 - C000:FFFF	ROM BIOS supplémentaire
11	B000:0000 - B000:FFFF	RAM d'écran
10	A000:0000 - A000:FFFF	RAM d'écran supplément (EGA/VGA)
9	9000:0000 - 9000:FFFF	RAM de 576 Ko à 640 Ko
8	8000:0000 - 8000:FFFF	RAM de 512 Ko à 576 Ko
7	7000:0000 - 7000:FFFF	RAM de 448 Ko à 512 Ko
6	6000:0000 - 6000:FFFF	RAM de 384 Ko à 448 Ko
5	5000:0000 - 5000:FFFF	RAM de 320 Ko à 384 Ko
4	4000:0000 - 4000:FFFF	RAM de 256 Ko à 320 Ko

Organisation de la mémoire du PC		
Bloc	**Adresse**	**Contenu**
3	3000:0000 - 3000:FFFF	RAM de 192 Ko à 256 Ko
2	2000:0000 - 2000:FFFF	RAM de 128 Ko à 192 Ko
1	1000:0000 - 1000:FFFF	RAM de 64 Ko à 128 Ko
0	0000:0000 - 0000:FFFF	RAM de 0 Ko à 64 Ko

Les 10 premiers segments de la mémoire sont réservés à la RAM de la mémoire centrale dont la taille est donc limitée à 640 Ko. Le segment 0 joue un rôle particulier car il contient des données et des routines cruciales du système d'exploitation.

A la suite de la mémoire RAM commune se trouve un segment de mémoire A qui appartient à une carte graphique EGA ou VGA et qui mémorise le contenu de l'écran dans les différents modes graphiques.

Le segment de mémoire B est réservé aux cartes vidéo monochromes et couleur. Elles se partagent ce segment pour y stocker l'affichage sur l'écran. La carte monochrome occupe les 32 Ko inférieurs et la carte couleur les 32 Ko supérieurs du segment. Chaque carte ne dispose cependant que de la mémoire réellement nécessaire pour l'écran concerné. Sur une carte monochrome, cette mémoire se limite à 4 Ko mais sur une carte couleur elle peut aller jusqu'à 16 Ko à cause des possibilités ouvertes par l'affichage graphique.

Les segments de mémoire suivants ne sont plus occupés par de la RAM mais par la ROM, à commencer par le segment C. Sur certaines machines ce segment accueille des routines du BIOS qui ne faisaient pas partie du noyau d'origine du BIOS. Sur le XT, il s'agira par exemple des routines de commande du disque dur.

Les segments D et E sont réservés à ce qu'on appelle les cartouches de ROM, comme on en trouve dans les ordinateurs domestiques pour charger des jeux. Cette possibilité n'est cependant pratiquement jamais exploitée sur les PC de sorte que la zone corespondante reste presque toujours inutilisée.

Le segment F contient finalement les véritables routines du BIOS, le programme de chargement initial du système ainsi que le BASIC en ROM qui ne se trouvent plus que sur les ordinateurs anciens.

Respect de l'organisation

Le matériel du PC n'est pas lié à une organisation spécifique de la mémoire. Bien qu'IBM ne l'ait pas voulu, il a fixé les règles que tous les autres constructeurs ont respectées par la suite. C'est surtout le logiciel qui est concerné par la structure décrite : BIOS et DOS

se sont spécialisés dans la gestion des diverses zones de mémoire à emplacements fixes (exemple : la mémoire d'écran). Chaque programme apparu sur le marché a en quelque sort scellé l'organisation déjà en vigueur, car les logiciels d'application eux aussi s'appuient sur une structure donnée, et il n'est pas bon de les tromper à cet égard.

1.3.5. La succession du PC original

L'IBM-PC d'origine n'a pas mis fin aux développements en micro-informatique mais il a figé un certain nombre de principes qui n'ont pas été remis en cause jusqu'à ce jour : par exemple les fonctionnalités du BIOS, l'organisation de la mémoire et la collaboration entre le processeur et les circuits auxiliaires.

Malgré tout, l'introduction de l'XT et de l'AT a provoqué quelques adaptations. L'arrivée de l'XT en 1983 correspond à l'apparition d'un disque dur dont la capacité de 10 Mo paraît aujourd'hui un peu faiblarde. L'ensemble du système n'en fut pas affecté pour autant : il y eut un peu de ROM supplémentaire installée dans le segment C et le BIOS se vit ajouter quelques fonctions nouvelles pour gérer le disque.

L'AT

En 1984, à peine plus d'un an après l'XT, apparut l'AT dont le nom "Advanced Technology" évoquait une série de nouveautés. La plus marquante concernait le processeur. Le 8088 fut abandonné au profit du 80286, le dernie-né d'Intel. Le bus de données fut étendu à 16 bits de sorte que les accès à des nombres de 16 bits n'avaient plus à être décomposés en deux temps lorsque la mémoire et la carte d'extension acceptaient de jouer le jeu. Par ailleurs le bus d'adresses passa de 20 à 24 bits car le 80286 était à même de gérer des adresses sur 24 bits, autrement dit un espace mémoire de 16 Mo.

Pour ce qui est de la mémoire de masse, l'AT dédoubla la capacité du disque dur en la portant à 20 Mo. Le lecteur de disquette supportait désormais des disquettes haute densité de 5' 1/4 dont la capacité atteint 1.2 Mo et qui sont encore en usage aujourd'hui. On vit aussi apparaître une horloge en temps réel alimentée par batterie, qui conservait enfin l'heure d'une session à l'autre. En outre, détail peu connu, le nombre de contrôleurs DMA et d'interruption fut augmenté : il y en eut deux de chaque type dans chaque machine.

Pour soutenir tout ce nouveau matériel, des fonctions supplémentaires furent ajoutées au BIOS, par exemple pour accéder à l'horloge alimentée par batterie.

En fait, l'AT méritait vraiment son qualificatif d'"advanced technology". Mais il restait en-deçà de ses possibilités et manifestait une nette tendance à freiner les solutions d'avenir pour préserver l'acquis. Ce principe de "compatibilité descendante" est clairement illustré par la limitation du mode protégé, un mode d'exploitation qui détachait le

80286 de ses prédécesseurs mais qui ne fut réellement exploité que depuis l'introduction du 80386 et de Windows 3.

A l'époque, en effet, ni le BIOS, ni DOS, ni les logiciels d'application dans lesquels les sociétés avaient investi des millions et des millions de dollars, n'étaient en mesure de tirer profit du mode protégé. On continua de travailler dans le mode dit réel, qui consiste à faire fonctionner le 80286 comme un super 8088 sans exploiter vraiment ses possibilités. On en est encore là aujourd'hui, ce n'est que le changement de génération induit par Windows NT et OS/2 qui est susceptible de tordre enfin le cou au mode réel.

PS/2

Une fois l'At lancé, IBM chercha une fois de plus à imposer un nouveau standard en développant les systèmes PS/2 qui se caractérisent esssentiellement par un bus très amélioré (appelé MCA = Micro Channel). Mais IBM commit l'erreur de garder secrètes les spécifications du nouveau bus, ne les révélant aux fabricants de cartes d'extension que contre des licences coûteuses. En conséquence de cette attitude, l'offre de cartes demeura discrète, et l'intérêt pour les PS/2 resta réduit car il fallait renoncer à ses cartes d'extension préférées lorsqu'on remplaçait un AT par un PS/2. Les cartes ISA ne peuvent pas être installées dans des systèmes à bus MCA car les lignes de ce bus sont complètement différentes.

Après l'AT : plus de standard

Le PS/2 annonce en fait le début de la fin de la suprématie d'IBM sur le marché du PC. De plus en plus de fabricants se pressèrent pour offrir des micro-ordinateurs d'abord moins chers et ensuite plus performants. Ce sont des sociétés comme Compaq qui produisirent le premier AT à base de 80386, développèrent des portables, poursuivant ainsi inlassablement la course au progrès.

Cette évolution présentait cependant un revers de la médaille. Car après IBM il n'y eut plus d'institution capable de donner clairement la direction à suivre. Dans un marché fragmenté en une multitude de fabricants, aucune société n'était en mesure de définir des standards matériels et logiciels et de les imposer. Ce n'est que plus tard que se formèrent divers comités pour tenter de mettre fin aux développements sauvages, comme celui de la carte Super-VGA, qu'aucun logiciel n'aurait pu vraiment exploiter à fond.

Après l'AT aucune nouvelle génération de PC bâtie sur le bus ISA ne fut définie. C'est pourquoi les systèmes équipés des processeurs 80386 ou i486 sont encore qualifiés d'AT, parce que sur l'essentiel ils reposent encore sur la technologie introduite par IBM avec l'AT.

1.4. Le processeur

Pour faire de la programmation système il ne faut pas forcément être un virtuose de l'assembleur. La programmation système se pratique aussi avec des langages évolués comme Basic, Pascal ou C. Il est cependant indispensable de bien connaître le fonctionnement du processeur, certains mécanismes jouent un rôle important dans la programmation système et doivent être maîtrisés jusqu'au niveau des langages évolués : par exemple la signification des registres, l'adressage de la mémoire, la notion d'interruption, les règles d'accès au matériel.

Depuis l'introduction du 8088 les principes n'ont pas changé, bien que cinq générations de processeurs se soient déjà succédé et que la puissance des derniers venus était impensable il y a dix ans. Les seules modifications concernent l'accroissement de la vitesse, elles n'affectent en rien le fonctionnement de base, du moins en mode réel sous DOS. Il est intéressant malgré tout de se remémorer l'évolution des processeurs d'Intel.

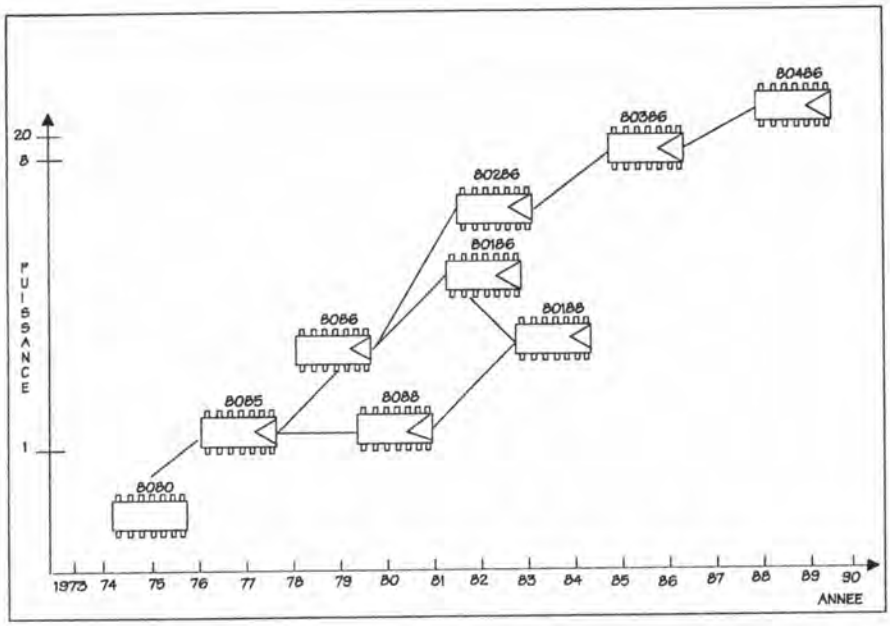
1.4.1. Le cerveau du PC

Le processeur a pour tâche de traiter des séquences d'instructions machine : c'est là le seul langage qu'il comprend vraiment. Ces instructions machine sont très élémentaires. Elles ne peuvent pas être comparées à des instructions en Basic, Pascal, C qui sont équivalentes à une multitude d'entre elles. Ainsi pour afficher un résultat à l'écran, il suffit d'une seule instruction appelée PRINT en BASIC, alors qu'il faut plus d'une centaine d'instructions machine.

A priori, le langage machine varie d'un ordinateur à l'autre, il est lié au type du micropocesseur présent. Les membres de la famille 80xxx d'Intel exploitent cependant le même langage, ce qui assure leur compatibilité.

Le premier-né de la famille, le 8086, a été mis au point en 1978. Ses successeurs ont subi diverses modifications et ont parfois été améliorés de façon radicale sans pour autant perdre le bénéfice de la compatibilité avec les prédécesseurs. Le 8088 est un cas à part qui constitue une régression. Disposant du jeu d'instructions du 8086 et de la même structure interne, il n'était capable de communiquer avec la mémoire qu'à travers un bus de 8 bits, comme nous l'avons vu précédemment.

Les autres membres de la famille sont des 8086 améliorés. Ainsi le 80186 présente des instructions supplémentaires, tandis que le 80286 s'est vu ajouter de nouveaux registres avec un espace d'adressage accru. Mais la grande nouveauté introduite par le 80286 est le mode protégé, décrit au chapitre XX, qui n'a pas su être exploité par DOS.



Evolution de la famille des processeurs 80xx d'Intel.

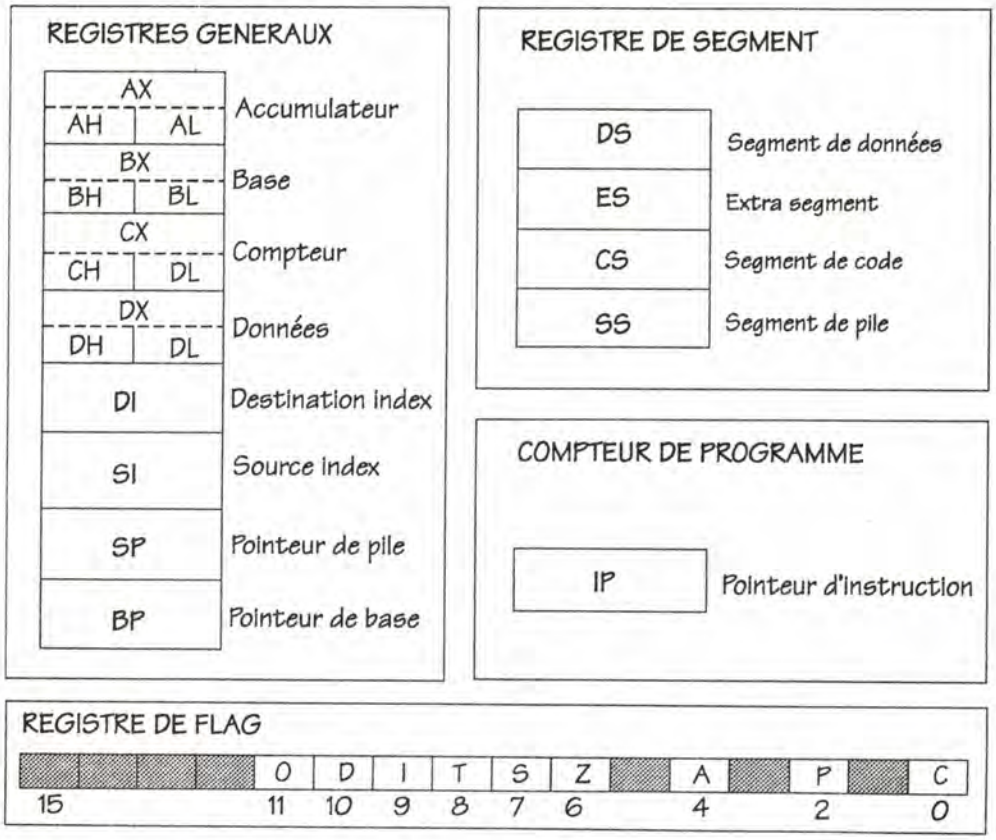
Au 80286 succéda le 80386 qui est sans doute le standard actuel pour les débutants sur PC. Son mode protégé a été perfectionné, c'est le premier processeur d'Intel qui dispose de registres de 32 bits mais ces registres ne sont pas vraiment exploitables sous DOS. Le 80386 existe dans deux versions appelées DX et SX qui se différencient par la cadence de fonctionnement et la largeur du bus de données. Le SX fonctionne avec un bus de données de 16 bits, tandis que le DX est capable, grâce à son bus de 32 bits, de transférer en une seule opération des mots de cette dimension.

L'état actuel de l'art est représenté par le 80486, souvent désigné comme "i486" par Intel. Par rapport au 80386, il intègre un coprocesseur arithmétique 80387 et une mémoire cache pour les instructions, le traitement des instructions se trouvant accéléré. Il assure une compatibilité descendante avec le 8086.

1.4.2. Les registres du processeur

Au centre de chaque processeur se trouvent ses registres, sortes de "variables matérielles". Le processeur y charge des données en provenance de la mémoire, les traite à l'aide des instructions machine puis les renvoie dans la mémoire. Ce mécanisme est plus rapide que la manipulation directe de la mémoire car les registres sont implantés à l'intérieur même du processeur, leur contenu est donc accessible sans passage par le bus. Dans la programmation système, les registres sont importants car ils voient passer le flux des informations entre les programmes et les fonctions du BIOS ou de DOS.

Du point de vue de la programmation système, l'usage des registres n'a guère varié depuis le 8086. Le BIOS et DOS ont en effet été développés pour ce processeur et ils n'exploitent que les registres de 16 bits. Les registres de 32 bits d'un 80386 et de l'i486 ne sont d'aucune utilité à la programmation système sous DOS. Les explications suivantes se limitent donc aux registres du 8088 qui sont également disponibles dans tous les processeurs ultérieurs de la famille.



Les registres du 8088

Tous les registres ont une taille de 16 bits (2 octets). Ils peuvent donc mémoriser une valeur entre 0 et 65535 (1111111111111111b ou FFFFh)

La figure précédente montre que les registres se divisent en quatre groupes : les registres généraux, les registres de segment, le compteur ordinal et le registre des indicateurs. Cette subdivision est due à une spécialisation des registres dont chaque groupe joue un rôle spécifique dans l'exécution des programmes et les accès à la mémoire.

Les registres généraux

Pour appeler les fonctions du système d'exploitation et du BIOS, les registres généraux auront pour nous une importance toute particulière : ils servent en effet à transmettre les paramètres d'appel. Ils peuvent par ailleurs être utilisés pour des opérations arithmétiques (addition, soustraction, etc...) qui constituent la base de l'activité du processeur dans les programmes. Les registres AX, BX, CX et DX se singularisent du fait qu'ils peuvent être chacun divisés en deux registres de 8 bits (un octet). Chaque registre général se compose donc en quelque sorte de trois registres : un grand registre de 16 bits et deux petits registres de 8 bits.

Les "petits" registres sont désignés respectivement par H (high = octet de poids fort) et L (low = octet de poids faible). Le registre AX, de 16 bits, se compose donc des registres de 8 bits appelés AH et AL. Le registre L occupe toujours les 8 bits inférieurs de poids faible (bits 0 à 7) du registre X et le registre H les 8 bits supérieurs de poids fort (bits 8 à 15) du registre X. Le registre AH se compose donc des bits 8 à 15 et le registre AL des bits 0 à 7 du registre AX. Il ne s'agit cependant pas là de trois registres indépendants. En effet, si le bit 3 du registre AH est modifié, la même modification affecte automatiquement le bit 11 du registre AX. On ne peut donc changer le contenu du registre AH sans modifier également celui du registre AX. Celui du registre AL reste par contre inchangé dans notre exemple puisque ce registre se compose des bits 0 à 7 du registre AX et qu'il ne contient donc pas le bit 11 du registre AX. Cette dépendance entre les registres AX, AH et AL s'applique aussi à tous les autres registres généraux et elle peut être exprimée en termes mathématiques

Il est en effet possible de calculer le contenu du registre X à partir des contenus des registres H et L correspondants et vice-versa. Pour obtenir le contenu du registre X, il suffit de multiplier le contenu du registre H par 256 et d'ajouter à ce produit le contenu du registre L.

Par exemple : si le registre CH contient 10, et CL le nombre 118, le contenu de CX sera égal à CH*256+CL, c'est-à-dire 10*256+118 = 2678.

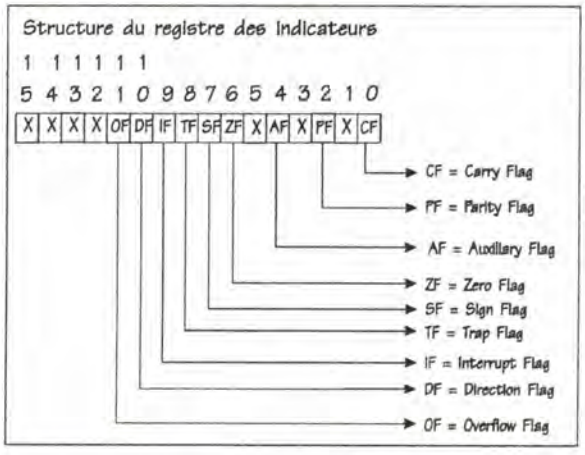
Grâce aux "petits" registres, il est possible de lire ou d'écrire des données réduites à 8 bits dans la RAM. Les transferts entre la mémoire et l'unité centrale ne peuvent en effet s'effectuer que sur des registres entiers, ils ne peuvent jamais porter sur des bits isolés.

Normalement, avec des registres de 16 bits, il est obligatoire de transférer systématiquement 16 bits à la fois. La présence des petits registres apporte davantage de souplesse.

En plus des registres généraux, la programmation système s'applique aussi à exploiter activement les registres de segment et le registre des indicateurs. La signification des registres de segment va être décrite dans la section suivante, mais nous n'attendrons pas plus longtemps pour étudier le registre des indicateurs.

Le registre des indicateurs

Le registre des indicateurs a d'abord une fonction de communication entre des instructions machine successives : il sert en effet à mémoriser l'état du processeur à la suite de diverses opérations artithmétiques et logiques. Ainsi, lors d'une addition de deux registres de 16 bits, un programme pourra tester l'indicateur de retenue (Carry Flag) pour voir si le résultat ne dépasse pas 65535, c'est-à-dire la capacité d'un registre de 16 bits. Les indicateur de signe, de zéro, de débordement ont des rôles analogues. Après une instruction de comparaison de deux registres, leur examen révèle si le contenu du premier registre était plus grand, inférieur ou égal au contenu du second registre.



Dans la programmation système avec les langages évolués, on n'exploite que le registre de retenue et quelquefois le registre de zéro. La plupart des fonctions de DOS et du BIOS utilisent ces indicateurs pour signaler une erreur d'exécution : par exemple en cas de mémoire insuffisante, ou lorsqu'on donne un nom de fichier inconnu. Nous verrons au chapitre 2 comment on procède pour lire la valeur d'un indicateur avec un langage évolué.

1.4.3. Formation des adresses mémoire

Le programmeur système est obligé de savoir comment le processeur traite les adresses parce que de nombreuses fonctions de DOS et du BIOS demandent qu'on leur communique des adresses de buffers. Le mécanisme d'adressage des 8088 et de ses successeurs est relativement compliqué. Il ne s'explique que par certaines considérations historiques sur le développement du 8086.

Il s'agissait à l'époque de construire un microprocesseur qui démontre sa suprématie sur ses concurrents de 8 bits (6502, Z80, etc...) en dépassant la capacité d'adressage des 64 Ko, considérée comme une barrière difficilement franchissable. Les développeurs

d'Intel se fixèrent un but très ambitieux. Il ne contentèrent pas de doubler la capacité d'adressage (c'est-à-dire le nombre de cellules de mémoire auxquelles un processeur peut accéder) mais ils l'étendirent carrément à 1 Mo, ce qui revenait à la multiplier par 16.

Mais il faut savoir le nombre de cellules de mémoire adressables dépend directement de la largeur d'un registre spécial, le registre d'adresse, qui lors de chaque accès mémoire accueille l'adresse concernée.

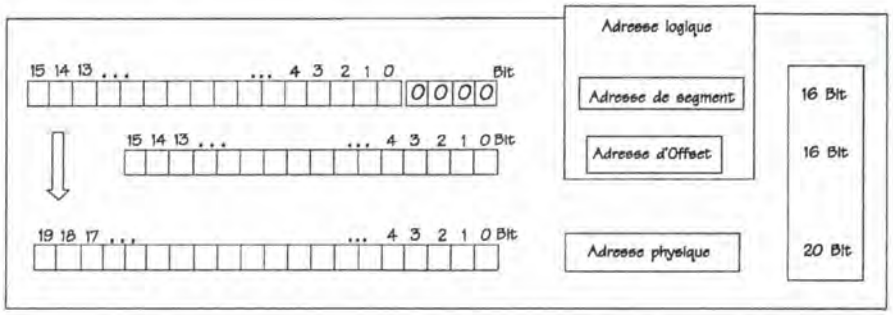
Chaque cellule de mémoire étant associée à un numéro qui lui est propre, et qui constitue son adresse, la capacité du registre d'adresse définit la capacité mémoire adressable.

Sur les prédécesseurs du 8088, le registre d'adresse avait 16 bits. Il ne pouvait donc mémoriser qu'un nombre compris entre 0 et 65535 (voyez plus haut), ce qui équivaut à une capacité d'adressage de 65536 cellules de mémoire (ne pas oublier de prendre en compte la cellule 0 !). La mémoire est donc limitée à 64 Ko. Il est aisé d'en déduire que le registre d'adresse doit nécessairement avoir une largeur de plus de 16 bits si on veut accéder à plus d'un million de cellules de mémoire. Il faut effectivement 20 bits pour étendre à 1 Mo la capacité d'adressage d'un microprocesseur. Finalement la solution ne semble pas très compliquée, il devrait suffire en principe d'intégrer un registre d'adresse de 20 bits sur notre processeur.

Mais à l'époque la technologie existante ne permettait pas encore de construire des processeurs d'une telle complexité. Il a donc fallu trouver une astuce pour contourner cette difficulté et pour parvenir malgré tout à former une adresse d'une largeur de 20 bits.

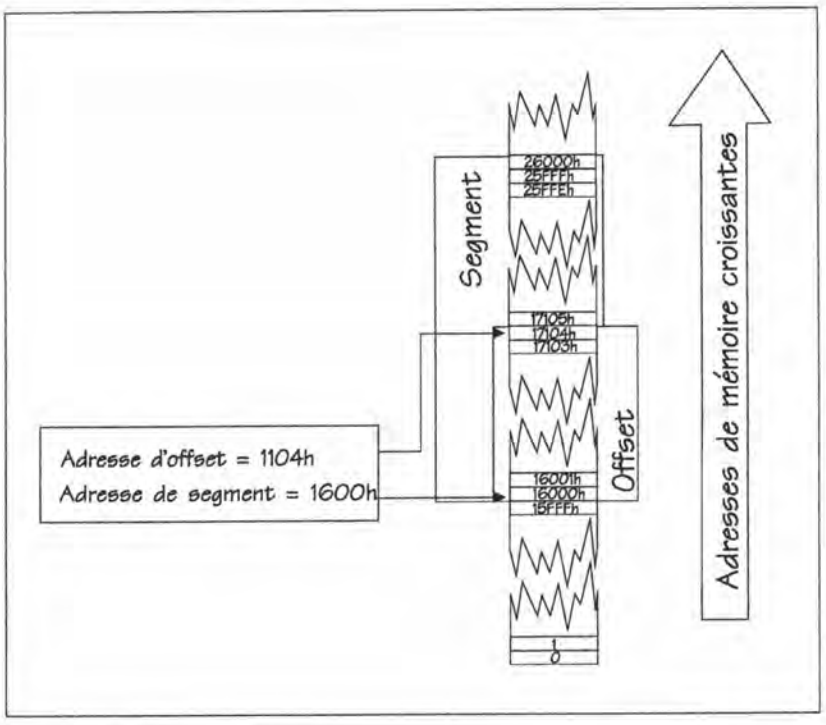
Cette astuce repose sur le principe suivant. Le 8088 ne dispose plus d'un registre d'adresse spécialisé. A chaque accès mémoire il compose l'adresse concernée en se servant de deux registres ou nombres de 16 bits tout à fait ordinaires, qu'il combine pour former un nombre de 20 bits.

Le premier nombre est toujours constitué par le contenu d'un registre de segment. Le second nombre est tiré d'un autre registre ou d'une cellule de mémoire. Les deux nombres en question ne sont pas purement et simplement additionnés (une telle addition ne donnerait d'ailleurs pas une adresse de 20 bits mais tout au plus une adresse de 17 bits). En fait, le premier nombre qui représente le contenu du registre de segment est tout d'abord décalé de 4 bits (c'est-à-dire d'un chiffre hexadécimal) vers la gauche, ce qui équivaut à une multiplication par 16 ou bien à l'adjonction de 4 bits. Il a maintenant une largeur de 20 bits. Le second nombre lui est alors additionné.



Formation d'une adresse à partir d'un segment et d'un offset

Les deux nombres traités sont des adresses partielles souvent désignées sous les termes d'offset et de segment. L'adresse de segment est l'adresse formée à partir du registre de segment et qui indique le début d'une zone de la mémoire (c'est ce qu'on appelle un segment) alors que l'adresse d'offset est l'adresse qui est additionnée à l'adresse de segment pour former l'adresse complète. L'offset fournit en quelque sorte le numéro exact de la cellule de mémoire à l'intérieur du segment dont le début est défini par le registre de segment. Supposons que l'offset et le segment soient tous deux égaux à 0. La mémoire adressée est la mémoire d'adresse 0. Si on augmente le segment de 1, la nouvelle mémoire en ligne n'est pas la mémoire 1 mais la mémoire 16, car dans la formation de l'adresse le segment est multiplié par 16. Si on continue d'incrémenter ainsi le segment, on parvient successivement aux adresses 32,48,60 etx.... Nous supposons évidemment que l'offset est maintenu à 0. Au bout de ce processus, on atteint l'adresse maximale de 1 Mo, lorsque le segment vaut 65535 (FFFFh). Si on maintient le segment constant tout en augmentant l'offset, on observe qu'en fin de compte le segment représente l'adresse de base à partir de laquelle on peut atteindre 65536 cellules de mémoire différentes. Chacun de ces segments s'étend sur 64 Ko. L'offset représente la distance d'une mémoire comptée à partir du début du segment.



Segment et offset

Comme les différents segments ne sont séparés que par 16 octets tout en occupant 64 Ko, il est clair qu'ils se recouvrent partiellement. Une adresse comme 130, par exemple, peut se définir d'une multitude de façon avec des segments et des offsets différents. On peut prendre le segment 0 et un offset de 130, mais aussi le segment 1 et un offset de 114, ou le segment 2 et un offset de 98, et ainsi de suite.

Mais peu importe ce chevauchement, vous êtes en droit de choisir la définition qui vous plaît. Ce qui compte, c'est que la multiplication par 16 du segment et l'addition de l'offset donne bien l'adresse recherchée.

Le principe de la définition des adresses par segment et offset a engendré une notation particulière pour indiquer l'adresse d'une cellule de mémoire :

On indique tout d'abord le segment, puis séparé par un double point, l'offset de la cellule de mémoire. Les deux composants sont exprimés sous forme de nombres hexadécimaux de 4 chiffres. Comme on se sert toujours de l'hexadécimal dans ces conditions, on peut laisser tomber le h utilisé généralement pour désigner les nombres hexadécimaux.

Ainsi l'adresse composée du segment 2000h et de l'offset AF3h, s'écrira 2000:0AF3.

Le rôle des registres de segment pendant l'exécution des programmes

Comme nous l'avons vu, le 8088 dispose de quatre registres de segment qui, en liaison avec d'autres registres, jouent un rôle décisif lors de l'exécution d'un programme en langage machine. L'existence même de 4 registres de segment, et non d'un seul, tient à la structure des programmes. Un programme se compose en effet, en premier lieu, d'une collection d'instructions, qu'on appelle le code. Mais il se compose aussi, au-delà du code, d'un certain nombre de données et de variables auxquelles il devra accéder et qu'il devra manier. Une programmation bien ordonnée devra donc veiller à séparer autant que possible, à l'intérieur de la mémoire, le code des données. Cette règle est respectée en attribuant à chacune des parties un segment particulier. En contrepartie, il faudra prévoir un registre de segment pour le code et un autre pour les données. Examinons donc de plus près la fonction des différents registres de segment :

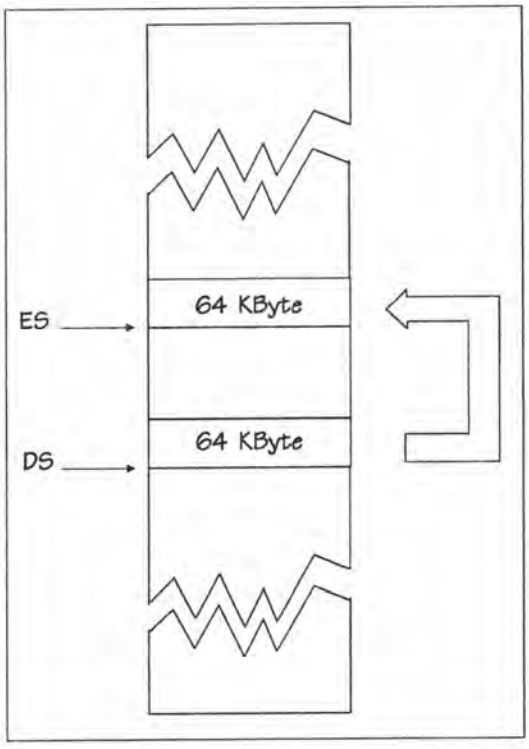
CS Le registre CS (Code Segment) détermine, en association avec le registre IP (comme offset) la cellule de mémoire où réside la prochaine instruction machine à exécuter. IP est l'abréviation de Instruction Pointer, c'est-à-dire pointeur d'instruction ou compteur ordinal (on utilise parfois aussi la désignation PC pour Program Counter). Dès que le processeur a exécuté une instruction, le registre IP est automatiquement incrémenté de façon à référencer la prochaine instruction machine : c'est ainsi que le programme peut être exécuté instruction par instruction.

DS Contrairement au registre CS, le registre DS (Data Segment) est utilisé pour former une adresse de données , c'est-à-dire chaque fois qu'il s'agit de lire ou d'écrire des données en mémoire. L'offset additionné au contenu du registre DS peut être située dans un autre registre ou bien constituer une partie de l'instruction utilisée.

SS La zone de mémoire qui commence à l'emplacement référencé par le registre SS (Stack Segment) est appelée pile. Elle est notamment utilisée pour recueillir l'adresse de retour lorsqu'on appelle un sous-programme. C'est ainsi, par exemple, que lors de l'exécution de l'instruction CALL, qui appelle un sous-programme, le processeur place sur la pile l'adresse à laquelle devra reprendre l'exécution du programme principal après exécution du sous-programme. La pile est aussi utilisée par les langages évolués pour transmettre des paramètres aux procédures et aux fonctions et pour mémoriser des variables locales. Lors d'un accès à la pile, l'adresse est formée à partir du registre SS combiné avec le registre SP ou avec le registre BP.

ES Citons enfin, comme dernier registre de segment, le registre ES (Extra Segment) qui est à la libre disposition du programmeur qui peut s'en servir pour divers accès aux données. Il se révèle pariculièrement préciaux lorsqu'on copie des variables ou des buffers en dehors d'une même zone de 64 Ko. Dans ce cas en effet il faut gérer deux segments distincts entre lesquels il faut

transférer octet par octet ou mot par mot les données concernées : l'un des segments est le segment de départ ou segment source et l'autre le segment de destination.



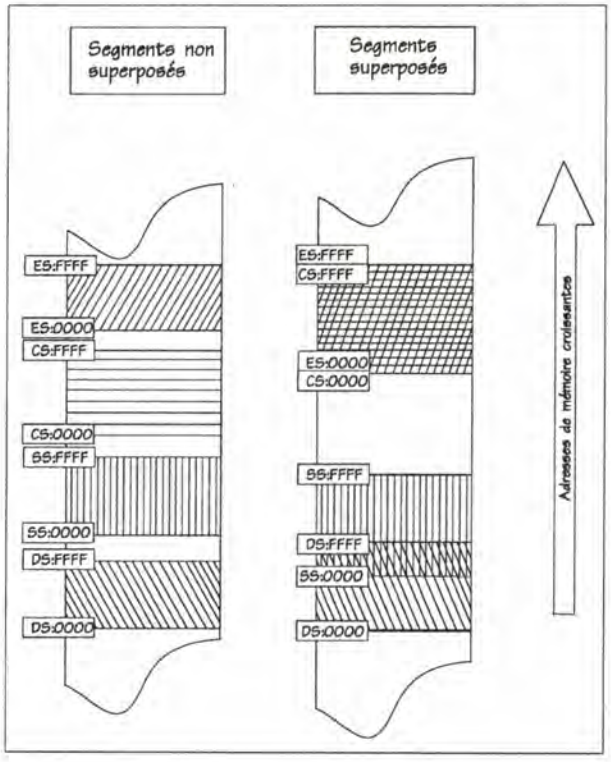
Copie d'une zone de mémoire à l'aide des registres DS et ES

Il est possible de mettre le segment de destination dans le registre ES tout en laissant dans DS le segment de départ. Le 8088 et ses dérivés possèdent même des instructions machine capables de recopier directement une zone de mémoire : ces instructions supposent que le segment de départ a été chargé en DS et le segment de destination en ES. Les offsets respectifs des zones de départ et de destination doivent être chargés en SI et DI. Autrement dit, en reprenant la notation introduite plus haut, la copie se fait de DS:SI en ES:DI.

Chevauchement des segments

Comme le montre la figure suivante, les segments référencés par les registres spécialisés peuvent se chevaucher partiellement, être identiques ou encore se trouver dans des zones tout à fait distinctes de la mémoire. Comme on aura rarement besoin de la totalité des 64 Ko d'un segment pour y stocker des données ou des instructions, la première

des trois possibilités citées est particulièrement pratique car elle permet de placer les données (registre DS) immédiatement à la suite des instructions en langage machine (registre CS), ce qui permet bien sûr d'économiser beaucoup de place mémoire.



Segments en situation ou non de chevauchement

Pointeurs NEAR et FAR

Ce qui s'appelle ici les adresses mémoire revient à la surface des langages évolués sous le nom de "pointeur". En Pascal et C, les pointeurs sont des variables destinées à mémoriser les adresses des objets qu'ils référencent. Ils se divisent en deux catégories : les pointeurs FAR (à accès long) et les pointeurs NEAR (à accès court).

Les pointeurs NEAR indiquent simplement l'offset d'un objet, ils ne comportent que 16 bits. Pour permettre l'accès recherché, le compilateur supplée un segment qui est chargé dans le registre de segement approprié. C'est pourquoi les pointeurs NEAR sont en général réservés aux accès portant sur des variables situées à l'intérieur du segment de données de 64 Ko installé par le compilateur.

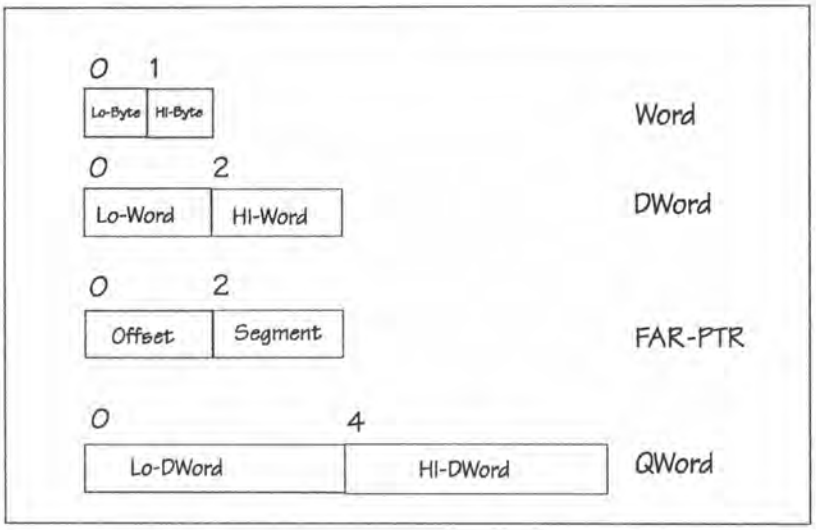
Les pointeurs FAR au contraire sont constitués d'un segment et d'un offset et sont donc mémorisés sur deux mots. Le mot inférieur contient l'offset et le mot situé juste à la suite le segment. En Turbo Pascal les pointeurs sont naturellement FAR, tandis qu'en C leur catégorie dépend du modèle mémoire mis en service. Voyez à ce sujet le chapitre 2.

Types de données et mémorisation

Les octets et les mots ne sont pas les seuls types de données que l'on rencontre en programmation système. On a également affaire à des doubles mots DWords qui interviennent lorsque les 16 bits d'un mot simple ne sont pas suffisants. Tel est le cas par exemple du compteurs de tops interne au BIOS qui au bout de dix heures franchit la limite de 65536.

Les processeurs de la famille 80xxx d'Intel mémorisent les doubles mots en disposant le mot de poids faible (bits 0 à 15) avant le mot de poids fort (bits 16 à 31). Ce procédé est appelé "Little Endian" et s'oppose au "Big Endian" qui intervertit l'ordre des mots et qui est exploité par les processeurs de la famille 68000 de Motorola (Apple McIntosh, Atari ST et Amiga).

La méthode Litte Endian est également appliquée, au-delà du stockage des doubles mots, au stockage des mots eux-mêmes. En effet l'octet de poids faible d'un mot est mémorisé avant l'octet de poids fort. Et l'histoire se répète de la même façon pour les quadruples mots (QWords) gérés par le coprocesseur arithmétique : le mot double de poids faible (bits 0 à 31) précède en mémoire lle mot double de poids fort (bits 32 à 63). La figure ci-après illustre ces permutations imbriquées :



Stockage de différents types de données au format Little Endian

1.5. Communication avec le matériel

Un programme communique avec le matériel, c'est-à-dire les circuits auxiliaires et les cartes d'extensions, par l'intermédiaire des ports d'entrée-sortie.

Un port est un contact avec l'extérieur, sur 8 ou 16 bits, identifié par une adresse comprise entre 0 et 65535. Il donne accès aux registres d'un circuit extérieur. En règle général, un périphérique occupe toute une série de ports.

Accès aux ports d'entrée-sortie

Pour communiquer avec les ports, le processeur se sert du bus de données et d'adresses, exactement comme dans le cas d'un accès à la mémoire. Il commence par envoyer sur une ligne spéciale du bus un signal destiné à prévenir tous les dispositifs reliés qu'il va s'adresser non pas à une mémoire mais un port. Il transfère ensuite l'adresse du port dans les 16 bits inférieurs du bus d'adresses et attend que l'un des appareils à l'écoute se déclare concerné, après quoi il envoie les données sur le bus de données.

Le même mécanisme se déroule en sens inverse pour une lecture. C'est alors la carte d'extension qui envoie le contenu du port au processeur, mais uniquement sur sa requête.

Pour piloter ainsi les entrées-sorties, les processeurs Intel disposent des instructions IN qui permettent d'envoyer des données du processeur vers le port ou inversement de charger dans le processeur des données en provenance d'un port. Le chapitre 2 explique comment effectuer la programmation correspondante avec un langage évolué.

Notez bien que ce n'est pas le système qui distribue les adresses des ports : chaque matériel est responsable de la zone d'adressage de ses ports. Il en résulte inévitablement des conflits entre différentes cartes qui occupent les mêmes adresses. C'est pourquoi la plupart des cartes d'extension sont munies de micro-interrupteurs DIP qui permettent de changer les adresses utilisées.

Standardisation des adresses des ports

Les adresses de port des circuits auxiliaires et de certaines interfaces standard ont été fixées à l'origine par IBM et font donc partie des paramètres standard de la micro-informatique. Il est vrai qu'il existe certaines différences entre le PC/XT d'une part et l'AT de l'autre, ce dernier disposant de plus de circuits rattachables. Le tableau suivant indique les adresses des ports attribués aux principaux composants matériels présentés dans ce livre, pour peu qu'ils soient utiles dans la programmation système sous DOS :

Circuit	PC/XT	AT
Contrôleur DMA (8237A-5)	000-00F	000-01F
Contrôleur d'interruption (8259A)	020-021	020-03F
Temporisateur	040-043	040-05F
Interface périphérique programmable (PPI 8255A-5)	060-063	néant
Clavier (8042)	néant	060-06F
Horloge en temps réel (MC146818)	néant	070-07F
Registre de page DMA	080-083	080-09F
Contrôleur d'interruption n2 (8259A)	néant	0A0-0BF
Contrôleur DMA n2 (8237A-5)	néant	0C0-0DF
Coprocesseur arithmétique	néant	0F0-0F1
Coprocesseur arithmétique	néant	0F8-0FF
Contrôleur de disque dur	320-32F	1F0-1F8
Manette de jeux (Joysticks)	200-20F	200-207
Unité d'extension	210-217	néant
2nde imprimante parallèle	néant	278-27F
Seconde interface série	2F8-2FF	2F8-2FF
Carte de prototype	300-31F	300-31F
Carte de réseau	néant	360-36F
1ère imprimante parallèle	378-37F	378-37F
Carte d'écran monochrome et 1re imprimante parallèle	3B0-3BF	3B0-3BF
Carte vidéo couleur/graphique	3D0-3DF	3D0-3DF
Contrôleur de disquette	3F0-3F7	3F0-3F7
Première interface série	3F8-3FF	3F8-3FF
(adresses exprimées en hexadécimal)		

1.6. Les interruptions

Dans la section 1.3.3, les interruptions ont été définies comme des mécanismes permettant à un périphérique d'interrompre brièvement le processeur dans l'exécution courante pour l'obliger à se brancher sur un sous-programme appelé gestionnaire d'interruption. Mais ce n'est là qu'une face des choses. Car les interruptions ne servent

pas seulement à piloter le matériel. Elles constituent aussi le moyen de communication central entre un programme et les fonctions de DOS ou du BIOS.

Les interruptions logicielles

Ce sont les interruptions dites logicielles qui sont déclenchées par les programmes lorsqu'ils ont besoin d'exécuter une fonction DOS, BIOS, de gestion de mémoire EMS, etc.. La fonction appelée est considérée par le processeur comme un sous-programme qui une fois achevé rend le contrôle à l'appelant.

Pour déclencher une fonction DOS ou BIOS par une interruption logicielle, le programme appelant n'a pas besoin de connaître l'adresse de la routine concernée, il lui suffit de connaître le numéro de l'interruption souhaitée. Ces numéros sont standardisés. C'est ainsi qu'indépendamment de la version de DOS, le service de ses fonctions est assuré par l'interruption 21h.

Le gestionnaire d'interruption est invoqué par l'intermédiaire d'une table de vecteurs d'interruption où se trouve l'adresse de la fonction recherchée. Le numéro de l'interruption constitue l'indice d'entrée dans la table qui est initialisée au démarrage du système de façon que les différents vecteurs d'interruption pointent sur les fonctions du BIOS.

L'avantage de ce mécanisme est patent : les fabricants de PC compatibles n'ont pas le droit de copier le BIOS de la ROM d'IBM. Mais ils peuvent implémenter en ROM les mêmes fonctionnalités qu'IBM avec une programmation différente. Les fonctions du BIOS seront déclenchées par les mêmes interruptions qu'avec un matériel IBM, les paramètres transmis à ces fonctions seront identiques et rangés dans les mêmes registres du processeur mais les routines responsables du traitement auront une autre forme.

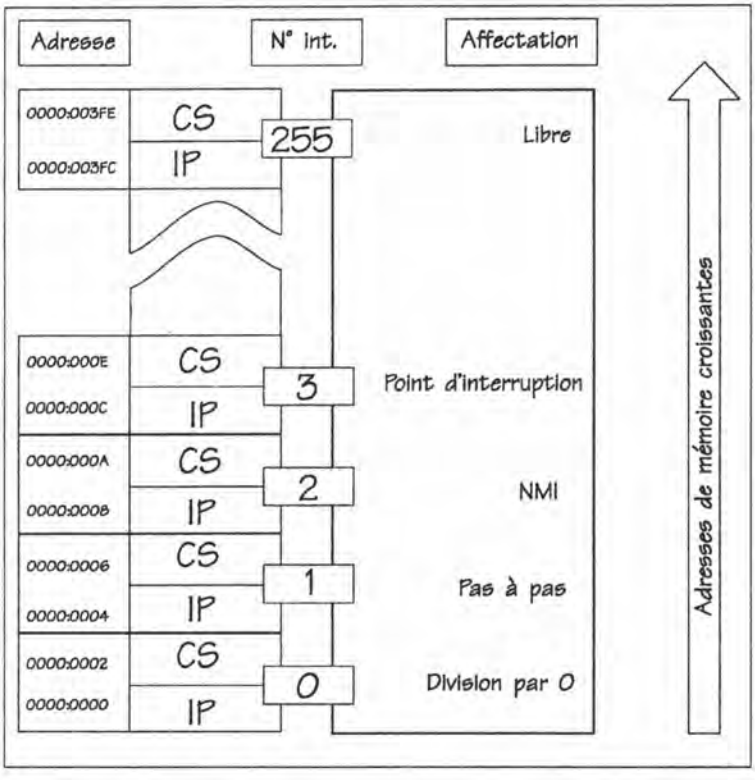
La notion d'interruption présente encore d'autres avantages qui seront évoqués au chapitre 2. Nous allons maintenant étudier la table des vecteurs d'interruption qui constitue la clé d'accès aux interruptions.

1.6.1. Structure et emplacement de la table des vecteurs d'interruption

Le fait que l'on parle de "table" montre qu'il existe plus d'une interruption dans un PC, sinon un simple pointeur suffirait. Il existe effectivement 256 interruptions pour un 8088 ou ses dérivés, c'est pourquoi la table des vecteurs d'interruption possède 256 entrées ou éléments. Les interruptions matérielles et logicielles sont mises sur le même plan, seul le type d'appel les distingue. Dans une interruption logicielle, c'est le programme qui fixe le numétro d'interruption tandis que dans une interruption matérielle c'est le périphérique qui s'en charge.

Chaque élément de la table des vecteurs d'interruption occupe deux mots successifs car il s'agit d'un pointeur FAR qui donne le segment et l'offset du gestionnaire associé. La table s'étend donc sur 1024 octets. Elle commence à la mémoire d'offset 0 et s'arrête à la limite du premier Ko. L'adresse où réside le gestionnaire associé à un vecteur se calcule en multipliant le numéro de l'interruption par quatre.

Comme la table des vecteurs est stockée en mémoire vive, elle peut être modifiée par n'importe quel programme. Les logiciels dits résidents et les drivers de périphériques ne manquent pas de faire usage de cette possibilité (voir chapitre 32).



Structure de la table des vecteurs d'interruption

1.6.2. Organisation de la table des vecteurs d'interruption standard

Le tableau suivant ne donne pas seulement les adresses des différents vecteurs d'interruption, il indique aussi les services rendus par chacun d'eux. La table est valable pour tous les PC et constitue un élément essentiel de ce qu'on appelle le "standard" de l'industrie. Un programme qui appelle une de ces interruptions est sûr de déclencher

partout le même effet. La plupart des interruptions et des fonctions concernées seront décrites tout au long de cet ouvrage.

Il faut noter cependant que de nombreux vecteurs ne sont exploitables que si le matériel correspondant est installé. Il en est ainsi par exemple de l'interruption 33h qui sert à activer les fonctions du driver de souris ou de l'interruption 5Ch qui sert d'interface avec les fonctions du NetBios lorsqu'un réseau est présent.

La mention "Réservé" signifie que l'interruption en question est exploitée par un des composants système (le plus souvent DOS) mais qu'elle n'a fait l'objet d'aucune documentation. On en connaît donc l'origine mais pas le fonctionnement.

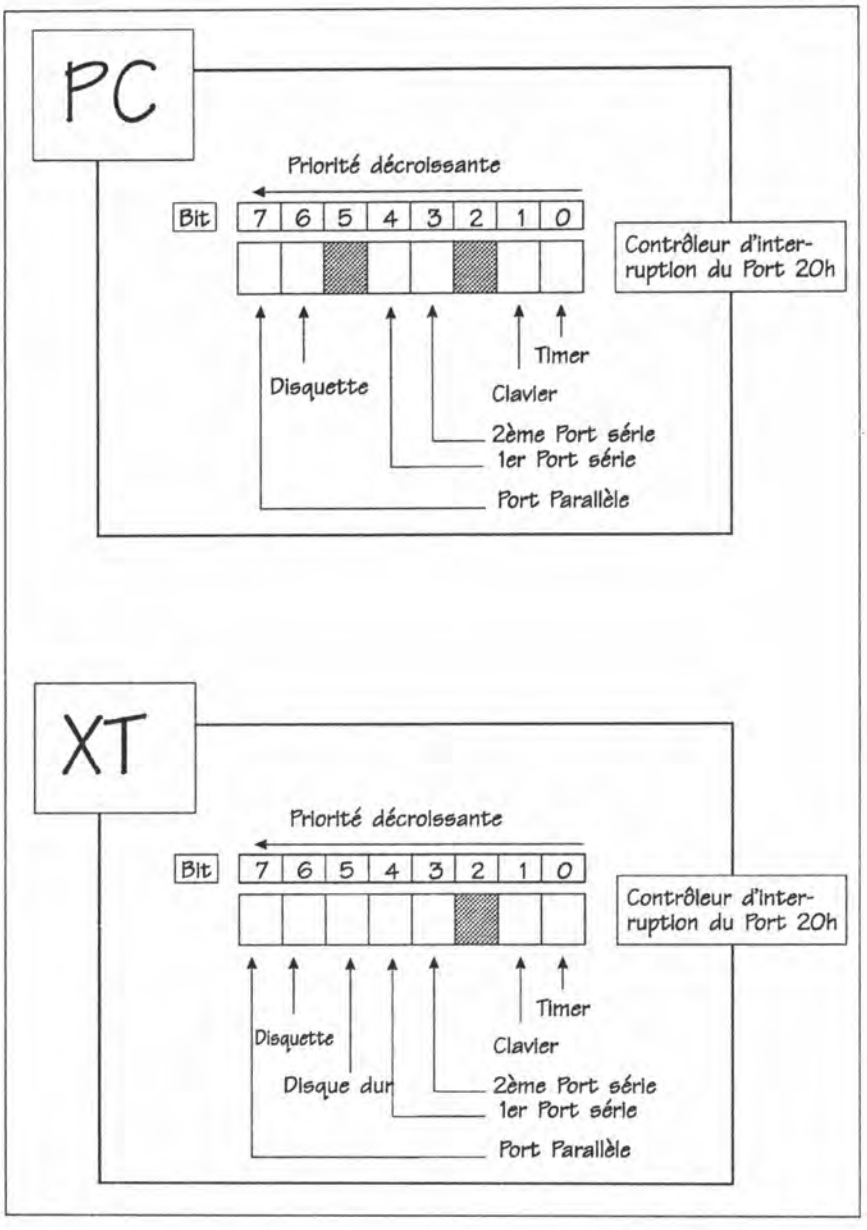
Nº	Adresse*	Fonction
00	000 - 003	CPU: Division par zéro
01	004 - 007	CPU: Pas à pas
02	008 - 00B	CPU: NMI (Défaut dans RAM)
03	00C - 00F	CPU: Point d'arrêt atteint
04	010 - 013	CPU: Débordement numérique
05	014 - 017	Copie d'écran
06	018 - 01B	Instruction inconnue (80286 seul.)
07	01D - 01F	Réservé
08	020 - 023	IRQ0: Timer (Appel 18,2 fois/sec.)
09	024 - 027	IRQ1: Clavier
0A	028 - 02B	IRQ2: deuxième 8259 (AT uniquement)
0B	02C - 02F	IRQ3: Interface série 2
0C	030 - 033	IRQ4: Interface série 1
0D	034 - 037	IRQ5: Disque dur
0E	038 - 03B	IRQ6: Disquette
0F	03C - 03F	IRQ7: Imprimante
10	040 - 043	BIOS: Fonctions vidéo
11	044 - 047	BIOS: Détermination configuration
12	048 - 04B	BIOS: Détermination taille RAM
13	04C - 04F	BIOS: Fonctions disquette/disque dur
14	050 - 053	BIOS: Accès à interface série
15	054 - 057	BIOS: Fonctions cassette ou étendues
16	058 - 05B	BIOS: Interrogation du clavier

Nº	Adresse*	Fonction
17	05C - 05F	BIOS: Accès à imprimante parallèle
18	060 - 063	Appel du BASIC en ROM
19	064 - 067	BIOS: Démarrage à chaud (ALT+CTRL+DEL)
1A	068 - 06B	BIOS: Lecture date et heure
1B	06C - 06F	Touche Break actionnée
1C	070 - 073	Appelé après chaque INT 08
1D	074 - 077	Adresse de la table de paramètres vidéo
1E	078 - 07B	Adresse de table paramètres disquette
1F	07C - 07F	Adresse des caractères graphiques
20	080 - 083	DOS: Terminaison du programme
21	084 - 087	DOS: Fonction de DOS
22	088 - 08B	Adresse de routine DOS fin du programme
23	08C - 08F	Adresse de routine CTRL-BREAK du DOS
24	090 - 093	Adresse de routine d'erreur du DOS
25	094 - 097	DOS: Lecture disquette/disque dur
26	098 - 09B	DOS: Ecriture sur disquette/disque dur
27	09C - 09F	DOS: Fin programme, laisser résident
28-3F	0A0 - 0FF	Réservé pour différentes fonctions non encore documentées du DOS
40	100 - 103	BIOS: Fonctions disquette
41	104 - 107	Adresse table des paramètres disque dur 1
42-45	108 - 117	Réservé
46	118 - 11b	Adresse table des paramètres disque dur 2
47-49	11c - 127	Librement définissable par le programme utilisateur
4A	128 - 12B	Heure alarme atteinte (AT seulement)
4B-67	12C - 19F	Librement définissable par le programme utilisateur
68-6F	1A0 - 1BF	Inutilisé

Nº	Adresse*	Fonction
70	1C0 - 1C3	IRQ08: Horloge temps réel (AT seul.)
71	1C4 - 1C7	IRQ09: (AT seulement)
72	1C8 - 1CB	IRQ10: (AT seulement)
73	1CC - 1CF	IRQ11: (AT seulement)
74	1D0 - 1D3	IRQ12: (AT seulement)
75	1D4 - 1D7	IRQ13: 80287 NMI (AT seulement)
76	1D8 - 1DB	IRQ14: Disque dur (AT seulement)
77	1DC - 1DF	IRQ15: (AT seulement)
78-7F	1E0 - 1FF	Inutilisé
80-F0	200 - 3C3	Utilisé à l'intérieur de l'interpréteur BASIC
F1-FF	3C4 - 3CF	Inutilisé
*	Toutes les adresses sont en hexadécimal	

1.6.3. Les interruptions matérielles

Les interruptions matérielles sont générées par les différents composants matériels du PC et parviennent au processeur par l'intermédiaire du contrôleur d'interruption. Nous allons examiner le fonctionnement de ce type d'interruption en tenant compte des différences existant entre le PC/XT et l'AT.


PC
Priorité décroissante
Bit
7 6 5 4 3 2 1 0
Contrôleur d'interruption du Port 20h
Disquette
Timer
Clavier
2ème Port série
1er Port série
Port Parallèle
XT
Priorité décroissante
Bit
7 6 5 4 3 2 1 0
Contrôleur d'interruption du Port 20h
Timer
Disquette
Clavier
Disque dur
2ème Port série
1er Port série
Port Parallèle

8 Interruptions matérielles pour le PC et l'XT

Ce sont les interruptions qui portent les numéros 8 à 15 qui sont considérées comme des interruptions matérielles en provenance du contrôleur d'interruptions. Il est théoriquement possible de déclencher ces interruptions comme des interruptions logicielles mais cette manière de procéder n'a pas beaucoup de sens. Le gestionnaire d'interruption associé est prévu pour interroger le matériel concerné et ne fonctionnerait pas correctement si l'appel n'en provient pas. On peut brancher jusqu'à 8 périphériques (sources d'interruptions) sur le contrôleur d'interruption d'un PC, par le moyen des lignes appelées IRQ0 à IRQ7. Le périphérique relié par IRQ0 possède la plus haute priorité, celui qui est relié par IRQ7 la plus faible. Si deux interruptions parviennent simultanément aux lignes IRQ3 et IRQ5, c'est d'abord IRQ3 qui sera pris en compte. Le numéro de l'interruption associée s'obtient en additionnant 8 au numéro de l'IRQ (=11 dans notre exemple).

Inhibition des interruptions matérielles

Dans certaines conditions un programme peut être amené à interdire l'exécution des interruptions matérielles. Pour qu'une telle interruption soit autorisée par le processeur, il faut que l'indicateur d'interruption qui se trouve dans le registre des indicateurs soit égal à 1. Si ledit indicateur a été mis à 0 par le programme, la demande du contrôleur d'interruption est ignorée. Vous trouverez davantage de détails sur ce thème au chapitre 2 qui évoque l'inhibition des interruptions matérielles et les circonstances qui le demandent.

Il est également possible de désactiver individuellement telle ou telle interruption mais il faut alors programmer le registre du masque d'interruption du contrôleur d'interruption.

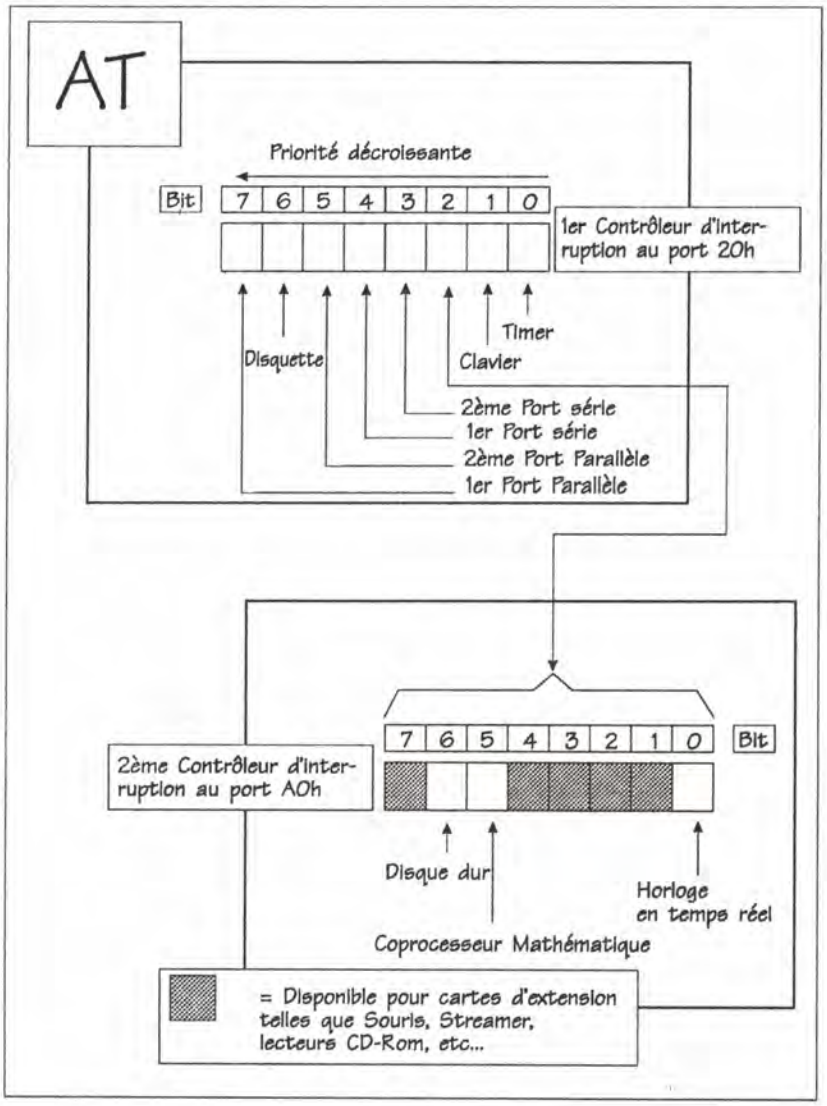
16 interruptions matérielles pour l'AT

Alors que le PC et l'XT s'en sortent avec 8 sources d'interruption gérées par un contrôleur 8259, ces ressources ne sont pas suffisantes pour un AT. Ce dernier est équipé de deux contrôleurs 8259 ce qui lui permet de gérer 16 sources d'interruption. Comme précédemment les interruptions portent des désignations, IRQ8 à IRQ15. Lorsqu'une demande d'interruption est issue de l'une des huit sources rattachées au deuxième contrôleur, ce dernier simule un IRQ2 adressé au premier contrôleur. Les interruptions associées au deuxième contrôleur sont donc plus prioritaires que celles des lignes IRQ4 et IRQ7 du premier contrôleur.

Lorsque la requête IRQ2 est émise, le gestionnaire de l'interruption 10 entre en scène. Il lit d'abord quelques registres du deuxième contrôleur d'interruption pour prendre connaissance du numéro de l'IRQ. En fonction de ce nombre l'une des interruptions

logicielles de 70h à 77h est déclenchée. Que cet appel ait été provoqué par un événement matériel n'a pas d'importance puisque le périphérique en question attend l'exécution de "son" gestionnaire d'interruption.

Ce processus consomme en fait la requête IRQ2 ce qui limite à 15 le nombre de sources d'interruptions qui peuvent effectivement être prises en compte.



Requêtes d'interruption et priorités dans un AT

1.7. Le matériel, le BIOS, le DOS

Pour clore ce chapitre, je voudrais montrer la façon dont les différents niveaux du matériel, du DOS et du BIOS sont imbriqués pour permettre aux programmes d'application d'accéder simplement aux périphériques du PC. Le clavier fournira un excellent sujet d'étude car les interruptions matérielles, les fonctions du BIOS et celles de DOS y interviennent de façon équilibrée. Suivons donc le chemin parcouru par un caractère entre sa frappe au clavier et le programme d'application qui en prend connaissance et l'affiche à l'écran.

La partie matérielle du clavier

Le clavier comporte un petit circuit spécialisé relié à l'unité centrale de l'ordinateur par un câble. Son rôle est de surveiller les touches et de signaler toute pression ou tout relâchement d'une touche. Il n'associe aucun caractère aux touches mais les désigne par un numéro. Les touches de commande comme Majuscule et Ctrl n'ont pas à cet égard de statut particulier et sont traitées comme les autres.

Si une touche est enfoncée, le circuit du clavier transmet son numéro au processeur sous la forme d'un code appelé Make-Code. Lorsque la touche est relâchée le même processus conduit à l'émission d'un Break-Code (ou Release-Code). La différence entre ces deux codes est minime : ils indiquent l'un et l'autre le numéro de la touche concernée (entre 0 et 127) mais dans le cas du Break-Code le bit 7 est à 1.

Pour initialiser la communication, le contrôleur du clavier envoie un signal d'interruption au contrôleur d'interruption par la ligne IRQ1. Pour peu que les interruptions matérielles ne soient pas inhibées et qu'aucune autre demande d'interruption ne se révèle plus prioritaire, le processeur exécute l'interruption 09H.

Le gestionnaire de clavier du BIOS

Nous en arrivons alors à un autre niveau, car derrière l'interruption 09h se cache une autre routine appelée le gestionnaire du clavier. Pendant l'appel du gestionnaire d'interruption, le circuit du clavier a envoyé le code de la touche actionnée au port 60h par l'intermédiaire du cordon du clavier. Le gestionnaire de clavier du BIOS le recueille à cet endroit et identifie le numéro de la touche enfoncée ou relâchée. Sa tâche est maintenant de transformer ce code en un code de caractère du jeu du PC (code dit ASCII). En effet le numéro de la touche n'est pas significatif pour un programme et par ailleurs il dépend du type de clavier utilisé. Seul le code ASCII est universel.

Cette tâche est cependant plus complexe qu'il n'y paraît à première vue car la routine du BIOS doit examiner si une des touches de commande telles que SHIFT ou ALT a

été actionnée. C'est ainsi que le code de la touche A devra le cas échéant être considéré soit comme une majuscule soit comme une minuscule.

Le buffer du clavier

Une fois le code ASCII établi, il est transcrit par le gestionnaire dans un buffer du BIOS dont la taille est de 16 octets. Ce buffer se trouve dans la partie basse de la mémoire vive. Si le buffer est plein, la routine émet un bip destiné à attirer l'attention de l'utilisateur sur son débordement. Le rôle du gestionnaire du BIOS est alors achevé et le processeur peut poursuivre le travail interrompu.

Ainsi les caractères ne sont pas directement livrés au programme en cours d'exécution, ils sont mis en attente dans le buffer du clavier du BIOS. Nous passons alors au niveau suivant qui relève toujours du BIOS.

L'interruption clavier du BIOS

Il s'agit de l'interruption 16h du BIOS qui recouvre 3 routines destinées à lire un caractère dans le buffer du clavier et à consulter l'état du clavier (quelles sont les touches de commande actuellement actives ?). Ces trois routines peuvent être appelées par un programme au moyen de l'instruction Int 16h. Cet événement peut survenir à n'importe quel moment et indépendamment de l'interruption matérielle du clavier avec laquelle on n'entre jamais en contact.

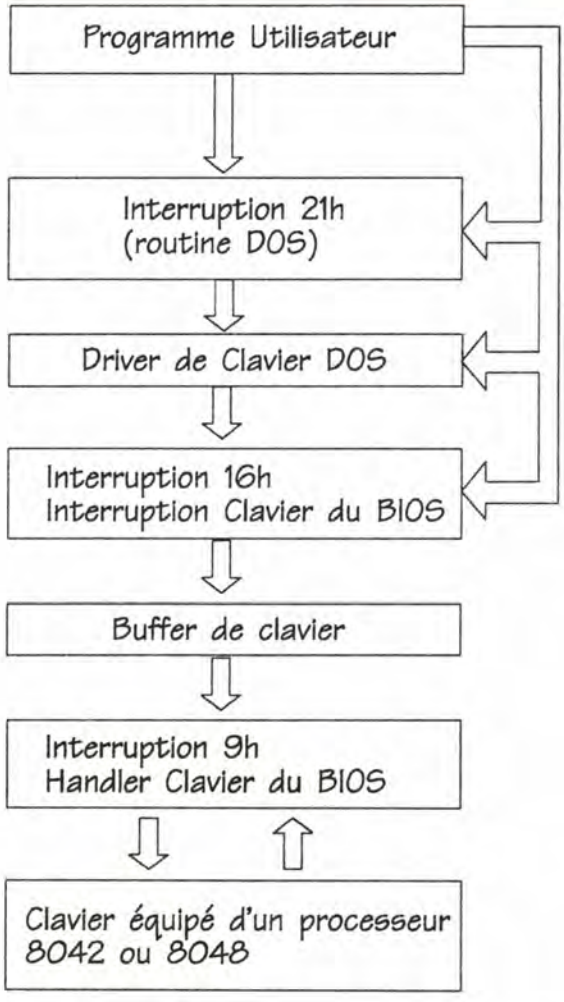
Mais un programme d'application n'est pas obligé de se compromettre directement avec l'interruption 16 h du BIOS pour lire une touche. Il peut très bien se servir de fonctions de DOS.

Le niveau de DOS

Nous voilà maintenant parvenus au niveau de DOS. Le système est d'abord représenté par les différentes routines du driver du clavier. Leur fonction consiste à exploiter l'interruption 16h du BIOS pour lire les caractères tapés et les stocker dans un buffer interne à DOS. Si le driver du clavier est le driver ANSI.SYS, il sera en outre capable de convertir certains codes (par exemple le code d'une touche de fonction) en d'autres codes ou en une séquence de caractères. C'est ainsi qu'il est possible de faire apparaître à l'écran l'instruction "DIR" en actionnant la touche F10.

Les fonctions d'un driver de périphérique peuvent être appelées directement à partir d'un programme d'application mais dans la pratique elles sont gérées par l'intermédiaire d'une fonction de DOS qui les chapeaute.

Nous sommes ainsi arrivés au dernier niveau avant le programme d'application : il s'agit des différentes fonctions d'accès au clavier de l'interruption 21h du DOS. Ces fonctions font appel au driver et restituent au programme d'application les caractères lus.



Les différents niveaux d'accès au clavier

2. Programmation système dans la pratique

Après ce premier chapitre bourré de principes et de théories, c'est maintenant le moment de passer à la pratique. Le but recherché est de faire de la programation système en Basic, Pascal et C. Car chaque langage dispose de ses propres instructions et fonctions pour accéder à la mémoire, lire les ports d'entrée-sortie ou déclencher des interruptions.

2.1. Programmation système en QuickBasic

Le Basic n'est pas le langage idéal pour faire de la programmation système, car il se maintient à une certaine distance du matériel informatique, beaucoup plus en tout cas que Pascal et surtout C qui passe parfois pour une sorte d'assembleur de haut niveau. La programmation système reste possible en Basic, même si elle ne peut pas avoir la même ambition qu'avec Pascal ou C. En Basic la notion de pointeur n'existe pas : cette lacune est tout à fait gênante dans de nombreux domaines d'application. C'est d'ailleurs la raison pour laquelle vous ne trouverez pas dans ce livre autant de programmes en Basic qu'en Pascal et C. Mais ce qui est réalisable, nous l'avons fait pour vous.

Ce chapitre présente les instructions et techniques qui s'imposent dans la programmation système. Nous avons pris en compte le compilateur QuickBasic de Microsoft, un standard dans le domaine du PC qui n'a pas à craindre de concurrence. Notez que la plupart des programmes ne pourront pas être exécutés en QBASIC, le nouvel interpréteur Basic fourni avec DOS 5.0. Car contrairement à QuickBasic, QBASIC n'a pas d'instruction pour appeler des interruptions.

2.1.1. Les types de données de QuickBasic

Lorsqu'on manipule des fonctions d'interruption, on a toujours affaire aux types de données gérés par le processeur car par définition ces fonctions sont écrites en langage machine et à ce niveau il n'existe aucun autre type de données. Tout développeur qui s'essaye à la programmation système en QuickBasic doit donc traduire en QuickBasic la structure des données du processeur. Le tableau suivant montre les types à utiliser :

Type QuickBasic	Mémorisé sous la forme
string*1	BYTE
integer	WORD
long	DWORD

Comme QuickBasic ne connaît pas les caractères isolés (char en Pascal et C) et ignore aussi les octets, il faut se satisfaire du type string*1, c'est-à-dire une chaîne de 1 octet que le compilateur traitera effectivement comme un seul octet. Mais cet octet-chaîne ne se laisse pas manipuler aussi simplement qu'un octet ordinaire. Une variable déclarée ainsi ne peut recevoir de valeur numérique que par le biais de la fonction CHR(), comme le montre l'exemple suivant :

```
DIM byte As STRING *1
byte= CHR$(35)   ' Ça marche
byte=5           ' Problème d'incompatibilité de types !
```

Inversement la valeur numérique doit être extraite au moyen de la fonction ASC :

```
DIM byte AS STRING*1
byte = CHR$(13)
IF ASC(byte)=13 THEN PRINT 13    'ÇA marche
IF byte=13 THEN PRINT 13       'Problème d'incompatibilité de types
```

Le travail avec les types de données integer et long donne également du souci lorsqu'on veut simuler les types WORD et DWORD. EN QuickBasic , les deux types sont en effet signés : le bit le plus à gauche détermine le signe. Lorsque ce bit est à 1, le nombre est considéré comme étant négatif.

Lorsqu'une fonction d'interruption renvoie un nombre du type WORD avec le bit 15 égal à 1 (Nombre > 32768), QuickBasic considère que ce nombre est négatif et son affichage à l'écran réserve des surprises. Le même problème se pose avec le type long mais il survient rarement en programmation système car les nombres doivent être immensément grands pour avoir un bit 31 égal à un.

Pour ce qui est du type integer, il est possible de s'en sortir en faisant intervenir une petite fonction de conversion en virgule flottante qui respecte le bit représentatif du signe. Voici par exemple la fonction MakeWord! qui est utilisée dans plusieurs programmes d'exemple :

```
FUNCTION MakeWord& (Nombre AS INTEGER)

IF Nombre < 0 THEN
  MakeWord = 65536& + Nombre
ELSE
  MakeWord = Nombre
END IF
END FUNCTION
```

Cette fonction attend comme argument l'entier de type integer dont le bit 15 est susceptible d'avoir la valeur 1. Elle renvoie un entier long garanti positif car c'est le bit 31 et non plus le bit 15 qui en fixe le signe.

Chaînes de caractères

Pour la plupart des fonctions du DOS et du BIOS une chaîne de caractères est une suite d'octets qui contiennent des codes ASCII et qui se terminent par un caractère "nul" de code 0. Dans la littérature système ce type de chaîne est parfois appelé ASCIIZ, le Z évoquant le zéro terminal. En Basic, les chaînes de caractères sont normalement mémorisées sous un autre format. Il faut d'ailleurs faire la distinction entre chaînes de longueur variable et chaînes de longueur fixe. La programmation système ne met en service que des chaînes de longueur fixe car leur adresse mémoire est plus facilement maîtrisable. Cette adresse se révèle indispensable lorsqu'une chaîne doit être transmise à une fonction de DOS ou du BIOS. Si vous déclarez une chaîne de longueur fixe en utilisant par exemple l'instruction :

```
dim s as string * 20
```

QuickBasic réserve 20 octets en mémoire centrale, avec un contenu a priori indéfini.

Lorsqu'on écrit une affectation du genre :

```
S="Micro-ordinateur"
```

la zone réservée est remplie avec les codes ASCII des différents caractères de la chaîne et complétée par des blancs.

Le caractère nul ne peut donc pas être simplement accroché par une instruction du genre :

```
s = s + chr$(0)
```

Il faut d'abord rechercher la fin de la chaîne en installant une boucle FOR NEXT puis y introduire le caractère nul par la fonction MID$:

```
DIM s AS STRING * 20
DIM i AS INTEGER

s = ""
INPUT "Tapez votre chaîne SVP: "; s

i = LEN(s) - 1
WHILE (i > 0) AND (MID$(s, i, 1) = " ")
  i = i - 1
WEND
IF i = 0 THEN i = 1
MID$(s, i) = CHR$(0)
```

Si vous connaissez a priori la chaîne c'est-à-dire si elle n'est pas tapée librement par l'utilisateur, les choses se simplifient car l'affectation peut inclure directement le caractère nul :

```
S="Micro-ordinateur"+chr$(0)
```

Structures et tableaux

Tout comme les programmes d'application, le DOS et le BIOS gèrent beaucoup d'informations rangées dans des structures et des tableaux qui doivent être reproduits dans le langage de programmation utilisé. Si vous avez déjà travaillé avec ces formes de données, vous serez en terrain familier.

La figure suivante décrit un exemple de structure renvoyée par certaines fonctions DOS. Il s'agit d'informations servant à parcourir les répertoires à la recherche de fichiers :

Structure d'une entrée de répertoire retournée par les fonctions 4Eh et 4Fh de DOS

Adr.	Contenu	Type
+00h	réservé	21 BYTE
+15h	Attribut du fichier	1 BYTE
+16h	Heure de la dernière mise à jour	1 WORD
+18h	Date de la dernière mise à jour	1 WORD
+1Ah	Taille du fichier	1 DWORD
+1Eh	Nom du fichier et extension séparés par un point sans indication de chemin d'accès terminaison = caractère nul	13 BYTE

Longueur : 43 octets

L'extrait de programme suivant montre comment cette structure peut être exprimée en QuickBasic :

```
TYPE DirStruct
   Reserve AS STRING * 21
   Attrib AS STRING * 1
   Time AS INTEGER
   Date AS INTEGER
   Size AS LONG
   DatName AS STRING * 13
END TYPE
```

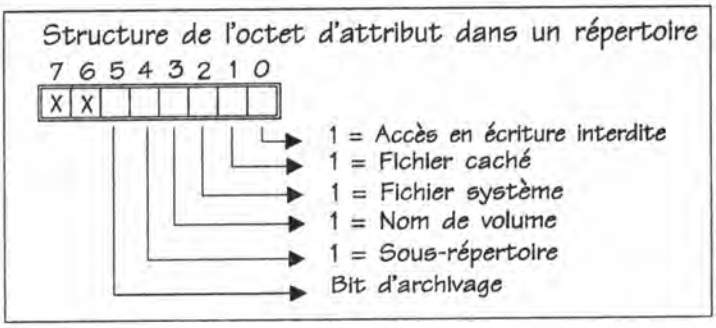
La zone réservée en tête de structure est reproduite sous la forme d'une chaîne de longueur fixe : c'est la manière la plus simple de réserver un nombre déterminé d'octets.

Les autres composants de la structure de DOS sont traduits en QuickBasic selon la règle suivante : les octets (BYTE) deviennent des string*1, les mots (WORD) des entiers simples

de type integer, les mots doubles (DWORD) des entiers longs. Les noms des champs n'ont aucune importance et peuvent être choisis librement car ils n'influencent en rien la structure.

Accès aux bits

Les structures comportent parfois des champs de bits où des bits isolés ou des groupes de bits ont une certaine signification. Par exemple l'octet représentant l'attribut d'un fichier dans la structure de répertoire présentée plus haut est en fait un champ de bits. Comme le montre la figure suivante, chaque bit indique individuellement si le fichier est protégé en écriture, si c'est un fichier système, ou s'il s'agit d'un sous-répertoire. Comment prendre connaissance de cette information ?



Pour accéder à un bit il faut connaître sa valeur comme puissance de 2. Le bit 0 a la valeur 1, le bit 1 la valeur 2^1=2, le bit 2 la valeur 2^2=4 et ainsi de suite jusqu'au bit 7 dont la valeur est 2^7=128. Pour savoir si l'attribut étudié désigne un sous-répertoire, il faut faire intervenir la valeur du bit 4, soit le nombre 16.

Notre préoccupation va être de mettre à 0 tous les autres bits de l'attribut de façon que seul le bit 4 reste présent pour être testé. Il suffit de combiner l'attribut avec un "masque" par l'opérateur binaire AND : tous les bits qui ne seront pas à 1 dans le masque seront mis à 0 dans le résultat. En QuickBasic on écrira :

```
Attribut AND 16
```

Le résultat de cette opération sera 16 si le bit 4 est à 1, et 0 dans tous les autres cas.

Avec une instruction IF, on aura :

```
IF ( ( Attribut AND 16 ) <> 0 ) THEN
  ' ceci est un sous-répertoire
ELSE
  ' ceci n'est pas un sous-répertoire
ENDIF
```

Lorsqu'on désire tester plusieurs bits simultanément, l'exercice est un peu plus compliqué. Il faut alors faire la somme des puissances de 2 représentées par les différents bits. Supposons que vous cherchiez à savoir si un fichier donné est un fichier système caché. Les indicateurs correspondants sont les bits 1 et 2 qui ont pour valeur totale $2^1 + 2^2=6$.

L'expression

```
Attribut AND 6
```

retourne le contenu des deux bits qui nous intéressent. Mais attention! Si vous reprenez un test IF semblable au précédent, il ne donnera pas forcément le résultat escompté car :

```
( Attribut AND 6 ) <> 0
```

est déjà TRUE si un seul des deux bits est à 1 et si de ce fait le résultat de l'opération AND est différent de 0. Pour vérifier si les deux bits sont simultanément à 1, il faut écrire :

```
IF ( ( Attribut AND 6 ) = 6 ) THEN
   ' c'est un fichier système caché
ELSE
  ' ce n'est pas un fichier système caché
ENDIF
```

Le programmeur ne se contente pas toujours de lire des bits, parfois il est obligé de les fixer d'autorité, notamment pour communiquer certains champs de bits à des fonctions de DOS ou du BIOS. Là encore c'est la valeur des bits calculée en puissances de deux qui intervient mais cette fois dans des opérations OR. Pour mettre à 1 le bit 3 de l'octet qui représente un attribut de fichier, on écrira en QuickBasic :

```
Attribut = Attribut OR 8
```

S'il faut fixer plusieurs bits à 1, leurs puissances devront être additionnées, par exemple

```
Attribut = Attribut OR (8+16)
```

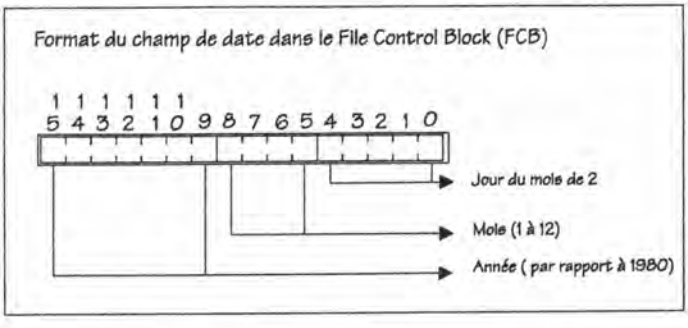
Mais comment faire pour mettre à 0 des bits ? On refait alors appel à un opérateur AND mais exploité d'une manière différente car il s'agit de sélectionner uniquement le bit visé. Les règles de la logique binaire nous apprennent qu'il faut alors inverser la valeur associée au bit en utilisant l'opérateur NOT. Pour mettre 0 le bit 5, on écrira :

```
Attribut = Attribut AND NOT (32)
```

Pour mettre à 0 plusieurs bits en même temps, on pratique l'addition des puissances de deux correspondantes, par exemple :

```
Attribut = Attribut AND NOT (32+8)
```

Un champ binaire n'est pas toujours constitué de bits isolés, il peut être fait de groupes de bits qui expriment une valeur. A titre d'exemple, examinons le champ date d'une entrée de répertoire. Il contient trois groupes de bits qui indiquent la date de création ou de dernière mise à jour de chaque fichier. Pour exploiter cette information il est inutile de tester individuellement les bits, il faut obtenir la valeur représentée par les trois groupes.



Le jour est facile à calculer avec les méthodes précédemment étudiées et l'opérateur AND :

```
Jour = Date AND ( 1 + 2 + 4 + 8)
```

Pour calculer le mois, l'opérateur AND ne suffit plus. Le goupe de bits doit en effet être de décalé de cinq positions vers la droite pour donner le numéro du mois. En QuickBasic le décalage de n positions vers la droite peut se faire au moyen d'une division par 2^n. Le mois et l'année s'obtiennent donc de la façon suivante :

```
Mois = ( Date AND ( 32 + 64 + 128 + 256 ) ) \ 32  '2 puissance 5 = 32
Annee = ( Date AND ( 512+1024+2048+4096+8192+16384+32768 ) ) \ 512
```

La deuxième affectation risque de vous réserver quelques surprises en matière de signe. C'est pourquoi nous ferons intervenir la fonction MakeWord :

```
Annee = ( MakeWord(Date) AND 65024& ) \ 512
```

Pour déplacer des bits non pas vers la droite mais vers la gauche, on remplace la division par une multiplication. . Si on décide ainsi de reconstituer un champ date à partir du jour, du mois et de l'année, il faudra écrire :

```
Date = Jour + ( Mois * 32 ) + ( Annee * 512 )
```

2.1.2. Appel des interruptions

Pour déclencher des interruptions logicielles, QuickBasic propose deux instructions appelées Interrupt et InterruptX qui donnent accès aux 256 interruptions du processeur Intel. Notez cependant que ces instructions ne font pas vraiment partie du compilateur proprement dit : elles se trouvent dans la bibliothèque Quick QB.QLB. Vous devrez donc lancer QuickBasic avec le paramètre /L si vous voulez les exploiter.

Les instructions dont il est question permettent notamment de déclencher les interruptions &H21, ce qui ouvre la porte aux 200 fonctions de l'API du DOS. API veut dire "Application Program Interface" et désigne l'ensemble des fonctions que le DOS met à la disposition des programmes d'application.

Pour faciliter le travail avec ces instructions, il existe un fichier Include qui peut être incorporé en début de programme par :

```
REM $INCLUDE: 'QB.BI'
```

La syntaxe des deux instructions est conforme aux modèles :

```
CALL INTERRUPT(NumInterrupt, Inreg, Outreg)
CALL INTERRUPTX(NumInterrupt, Inreg, Outreg)
```

Accès aux registres du processeur

Les arguments Inreg et Outreg communiqués à l'instruction d'interruption sont du type RegType qui représente une structure définie à l'intérieur de QB.BI. Elle est utilisée pour rendre accessible les registres du processeur à un programme écrit en Basic. Avant de s'exécuter, l'instruction INTERRUPT charge dans les registres du processeur les données mises au préalable dans la structure Inreg. Selon un mécanisme symétrique, à l'issue de l'interruption, le contenu des registres du processeur est transféré dans les registres Outreg. Il faut donc remplir correctement la structure Inreg avant d'appeler INTERRUPT. On trouvera les résultats de la fonction d'interruption dans les registres Outreg.

Le type RegType reflète fidèlement les registres du processeur, selon la syntaxe :

```
TYPE RegType
   ax AS INTEGER
   bx AS INTEGER
   CX AS INTEGER
   dx AS INTEGER
   bp AS INTEGER
   si AS INTEGER
   di AS INTEGER
flags AS INTEGER
END TYPE
```

Notez bien que ce sont les registres de 16 bits qui sont représentés. Pour accéder à un registre de 8 bits, il faut passer par le registre de 16 bits correspondant. Ainsi dans la séquence suivante, le nombre &h1B est chargé dans le registre Ah grâce à une multiplication par 256 qui décale les bits de 8 positions vers la gauche :

```
DIM Regs AS RegType
Regs.AX = &h1B * 256
CALL INTERRUPT( &hxyz, Regs, Regs )   'Appel de l'interruption
```

En même temps AL est mis à zéro. Si cet effet est indésirable, il faut adopter une voie plus prudente en insérant la valeur souhaitée à l'aide d'une opération OR :

```
DIM Regs AS RegType
Regs.AX = Regs.AX OR ( &h1B * 256 )
CALL INTERRUPT( &hxyz, Regs, Regs )   'Appel de l'interruption
```

Pour fixer le contenu du registre AL, il suffit d'écrire la valeur souhaitée dans AX, pour peu que AH doive être nul. Mais si AH contient déjà une valeur à préserver, il faut encore faire appel à un opérateur OR :

```
DIM Regs AS RegType
Regs.AX = &h1B              'Charge &h1B en AL et 0 en AH
Regs.AX = Regs.AX OR &h1B   'Ici  AH reste inchangé
CALL INTERRUPT( &hxyz, Regs, Regs )   'Appel de l'interruption
```

Ces principes s'appliquent évidemment à tous les registres généraux quels qu'ils soient. Il est également très simple de lire le contenu d'un registre de 8 bits à la suite d'une interruption. Si c'est l'octet de poids fort qui vous intéresse, vous devrez diviser le contenu du registre de 16 bits par 256. Si c'est au contraire l'octet de poids faible, vous pouvez éliminer les bits de gauche en mettant en service une opération AND :

```
DIM Regs AS RegType
CALL INTERRUPT( &hxyz, Regs, Regs )   'Appel de l'interruption
PRINT "AH = "; MakeWord(Regs.AX) \ 256
PRINT "AL = "; Regs.AX AND &HFF
```

Prise en compte des registres de segment

Si vous êtes attentif, vous avez sans doute remarqué que les registres de segment ne sont pas mentionnés dans le type RegType. Il existe pourtant des fonctions de DOS et du BIOS qui attendent des arguments dans les registres DS et ES ou qui renvoient des résultats dans ces segments. C'est pourquoi Quickbasic propose l'instruction INTERRUPTX qui fonctionne exactement comme INTERRUPT mais avec des structures du type RegTypeX. Comme vous le devinez, RegTypeX est semblable à RegType mais contient deux champs supplémentaires pour héberger les contenus de ES et DS :

```
TYPE RegTypeX
   ax AS INTEGER
   bx AS INTEGER
   CX AS INTEGER
   dx AS INTEGER
   bp AS INTEGER
   si AS INTEGER
   di AS INTEGER
flags AS INTEGER
   ds AS INTEGER
   es AS INTEGER
END TYPE
```

Lecture des indicateurs du registre des indicateurs

Dans beaucoup de cas ce ne sont pas seulement les registres généraux AX,BX,CX, etc... mais aussi les indicateurs du registre des indicateurs qui renvoient des informations au programme appelant. Les fonctions de DOS font largement usage de l'indicateur de retenue qui signale une erreur lorsqu'il est à 1 après exécution de la fonction.

Après avoir déclenché une instruction INTERRUPT ou INTERRUPTX vous pouvez accéder aux différents indicateurs du processeurs en exploitant la variable FLAGS de la structure Outreg. Le contenu individuel de chaque indicateur est obtenu par une combinaison AND appropriée, selon le tableau suivant :

Indicateur	Position	Puissance de deux
Retenue (Carry)	0	1
Parité	2	4
Auxiliaire	4	16
Zéro	6	64
Signe	7	128
Débordement (Overflow)	11	2048

Pour tester l'indicateur de retenue, on écrira par exemple :

```
DIM Regs AS RegType
CALL INTERRUPT( &hxyz, Regs, Regs )   'Appel de l'interruption
IF ( Regs.Flags AND 1 ) <> 0 THEN PRINT "Erreur"
```

2.1.3. Du bon usage des buffers

De nombreuses fonctions utilisent comme argument un pointeur sur un buffer qui sert de zone d'échange pour lire ou écrire des données. Ces pointeurs sont toujours FAR: ils sont constitués d'un segment et d'un offset parce que les données ne se trouvent pas forcément dans le segment du programme courant, elles peuvent être dispersées n'importe où dans la mémoire.

Transmission des pointeurs aux fonctions d'interruption

Prenons comme exemple la fonction numéro &H09 de DOS qui affiche une chaîne de caractères à la position courante du curseur. Comme toutes les fonctions DOS elle s'attend à trouver son numéro d'appel (&H09) dans le registre AH. Par ailleurs l'adresse du buffer contenant la chaîne à afficher doit être mémorisée dans les registres DS:DX. DS est destiné à la partie segment et DX à l'offset.

Créer une chaîne n'est pas difficile en QuickBasic mais comment trouver l'adresse du buffer dont nous avons besoin ? En réalité, c'est très simple car QuickBasic dispose de deux fonctions VARSEG et VARPTR qui renvoient justement le segment et l'adresse d'une variable. Le programmme suivant montre comment ces fonctions peuvent être mises en service dans le cadre d'un appel à la fonction &H09 de DOS :

```
'$INCLUDE: 'QB.BI' 'Fichier d'inclusion pour les appels d'interruptions

DIM s AS STRING * 20
DIM RegsX AS RegTypeX

CLS
s = "Micro-ordinateur" + "$"
RegsX.AX = &H900                    'Numéro de la fonction
RegsX.DS = VARSEG(s)                'Segment de s
RegsX.DX = VARPTR(s)                'Offset de s
CALL interruptx(&H21, RegsX, RegsX)
```

Réception de pointeurs renvoyés par les fonctions d'interruption

Nous allons étudier un programme symétrique du précédent qui s'appelle GetMedia. Il appelle la fonction 0x1B de DOS qui n'attend aucun pointeur mais en retourne un dans les registres DS:BX. Ce pointeur référence un octet qui contient le code d'identification du support (media-ID) du lecteur actif. Ce code, compris entre 0xF0 et 0xFF, décrit de façon interne à DOS le type du lecteur en service. La valeur 0xF8 (248) caractérise les disques durs de toutes sortes.

Comme QuickBasic ne connaît pas les pointeurs FAR, il faut utiliser l'instruction PEEK() pour lire le code du support. Mais PEEK() ne fonctionne qu'avec un offset, le segment concerné étant toujours le segment courant. Heureusement le segment courant peut être facilement fixé par l'instruction DEF SEG, comme le montre le programme suivant :

```
'$INCLUDE: 'QB.BI'   'Fichier d'inclusion pour les appels d'interruption

DIM RegsX AS RegTypeX
DIM MediaID AS INTEGER

CLS
RegsX.AX = &H1B00                          'Numéro de la fonction
CALL interruptx(&H21, RegsX, RegsX)
DEF SEG = RegsX.DS                         'Fixe le segment
MediaID = PEEK(RegsX.BX)                   'Lit le code d'identification
PRINT "Code du support: "; MediaID
```

2.2. Programmation système en Pascal

Dès qu'on prononce le nom de ce langage, on est obligé de rajouter "Turbo", tant il est vrai que Turbo Pascal domine le marché. La tentative de Microsoft pour briser le monopole de Borland en introduisant QuickPascal a tourné court. Pourtant Turbo Pascal n'est certainement pas le point final du développement de Pascal : l'apparition de produits tels que Pascal+, le compilateur de Stony Brook, le montre clairement. Complètement compatible avec les sources de Turbo Pascal 6.0, le code généré tourne souvent deux fois plus vite, en prenant moins de place.

Pour le moment la qualité du code ne joue aucun rôle car il s'agit d'indiquer quel est le compilateur qui a servi à réaliser les innombrables programmes d'exemple. A cet égard je peux vous rassurer : bien que les programmes aient été développés en Turbo Pascal 6.0, ils fonctionnent également avec Turbo Pascal 5.5. L'extension orientée objets n'a pas été exploitée car les réalisations étudiées ici concernent la pure programmation système. Mais cela ne veut pas dire que ce type de programmation ne puisse pas tirer profit d'un environnement d'objets, bien au contraire.

Pensez par exemple à un programme destiné à gérer plusieurs interfaces série. Les routines sont toujours les mêmes, seules les adresses des ports sont différentes. Il est facile d'imaginer un type d'objet qui gère une interface série en mémorisant dans une variable objet l'adresse du port. Pour gérer plusieurs interface, il suffirait de créer plusieurs instances de ce type d'objet en leur affectant l'adresse du port qui convient.

Mais assez de préambules : concentrons-nous sur la programmation système en Turbo Pascal.

```
    i : integer;

begin
  write ( 'Votre chaîne: ' );
  readln( ASCIIZ );
  ASCIIZ := ASCIIZ + chr(0);
  for i := 0 to ord( ASCIIZ[0] ) do
    begin
      write( i:2, '   ', ord( ASCIIZ[i] ):3 );
      if ( ASCIIZ[i] > ' ' ) then
        write( '  ', ASCIIZ[i] );
      writeln;
    end;
end.
```

Nous reproduisons ci-dessous l'affichage déclenché par le programme précédent. Ce programme demande que l'on tape une chaîne de caractères puis il ajoute un caractère nul et affiche la chaîne caractère par caractère. Notez bien l'octet introductif qui mémorise la longueur de la chaîne.

```
Votre chaîne: Chaîne ASCIIZ       <---- chaîne saisie
 0    14                          <---- octet définissant la longueur
 1    67  C
 2   104  h
 3    97  a
 4   140  î
 5   110  n
 6   101  e
 7    32
 8    65  A
 9    83  S
10    67  C
11    73  I
12    73  I
13    90  Z
14     0                <---- caractère nul terminal
```

Structures et tableaux

Tout comme les programmes d'application, DOS et le BIOS gèrent de nombreuses informations en s'aidant de structures et de tableaux que l'on est parfois obligé de reproduire dans la programmation système en Pascal. Si vous êtes un habitué des "arrays" (tableaux) et des "records" (enregistrements) vous n'aurez pas de mal à appliquer votre expérience. Ce qui est important c'est que le compilateur range les données dans le bon ordre, qu'elles commencent à une frontière de mot (Les processeurs 80xxx sont plus efficaces lorsque les données à traiter sont situées à des adresses paires, autrement dit lorsqu'elles sont alignées sur des mots-mémoire), et qu'aucun espace libre ne soit laissé entre elles.

2.2.1. Les types de données de Turbo Pascal

Comme c'est le cas de la plupart des compilateurs, les types de données de Turbo Pascal correspondent plus ou moins à ceux du processeur pour permettre un traitement plus simple et plus rapide. Le tableau suivant montre comment Turbo Pascal mémorise ses différents types de données :

CHAR	BYTE
BYTE	BYTE
BOOLEAN	BYTE
INTEGER	WORD
WORD	WORD
LONGINT	DWORD
POINTEUR	DWORD

En Turbo Pascal les pointeurs sont en principe FAR, aussi bien les pointeurs sur les données que les pointeurs dits procéduraux qui pointent sur du code de programme.

Chaînes de caractères

La manière dont les chaînes sont stockées en Pascal ne correspond pas aux attentes de la plupart des fonctions du DOS et du BIOS. Pour ces dernières en effet une chaîne de caractères est une suite d'octets qui contiennent des codes ASCII terminés par un caractère "nul" de code 0. Dans la littérature système ce type de chaîne est parfois appelé ASCIIZ, le Z évoquant le zéro terminal.

Turbo Pascal mémorise aussi la suite des codes ASCII mais elle ne se termine pas par un caractère nul. Par contre elle est introduite par un octet supplémentaire qui mémorise la longueur de la chaîne. Ce procédé de stockage facilite le traitement des chaînes mais il n'est pas compatible avec les mécanismes du DOS et du BIOS.

Mais comme le montre le listing suivant, il est facile de convertir les chaînes TP en chaînes ASCII en y accrochant un caractère nul. Lorsqu'on transmettra ce type de chaîne à une fonction du DOS ou du BIOS, on prendra soin de définir l'adresse du premier caractère avec un décalage de 1 (string[1]), pour ne pas envoyer l'octet de longueur (string[0]). Le paragraphe suivant consacré aux interruptions montrera l'usage de cette technique.

```
program ASCIIZDemo;

var ASCIIZ : string[100];
```

Le compilateur Turbo Pascal comporte une options d'alignement ALIGN DATA et la directive associée {$A} mais l'alignement en question ne concerne pas les tableaux et les enregistrements.

La figure suivante décrit un exemple de structure renvoyée par des fonctions DOS. Il s'agit d'informations servant à parcourir les répertoires à la recherche de fichiers :

Structure d'une entrée de répertoire retournée par les fonctions 4Eh et 4Fh de DOS		
Adresse	Contenu	**Type**
+00h	réservé	21 BYTE
+15h	Attribut du fichier	1 BYTE
+16h	Heure de la dernière mise à jour	1 WORD
+18h	Date de la dernière mise à jour	1 WORD
+1Ah	Taille du fichier	1 DWORD
+1Eh	Nom du fichier et extension séparés par un point sans indication de chemin d 'accès terminaison = caractère nul	13 BYTE
Longueur : 43 octets		

L'extrait de programme suivant montre comment cette structure peut être exprimée en Turbo Pascal :

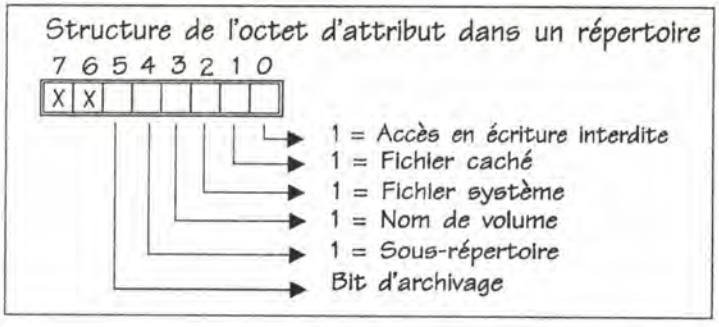
```
type DirBufTyp = record { Structure renvoyée par les Fonctions $4E et
$4F }
                   Reserve   : array [1..21] of char;
                   Attr      : byte;
                   Time      : integer;
                   Date      : integer;
                   Size      : longint;
                   Name      : array [1..13] of char
                 end;
```

La zone réservée en tête de structure est rendue par un tableau qui n'est pas forcément un tableau de caractères CHAR mais peut aussi être un tableau d'octets BYTE. Il faut simplement veiller à ce que les champs en Pascal respectent l'adresse relative à l'origine de la structure de DOS.

Les autres composants de la structure de DOS sont traduits en Turbo Pascal selon la règle suivante : les octets restent des BYTEs, les mots (WORD) deviennent des entiers simples de type integer, les mots doubles (DWORD) des entiers longs longint. Les noms des champs n'ont aucune importance et peuvent être choisis librement car ils n'influencent en rien la structure.

Accès aux bits

Les structures comportent parfois des champs de bits, où des bits isolés ou des groupes de bits ont une certaine signification. Par exemple l'octet représentant l'attribut d'un fichier dans la structure de répertoire présentée plus haut est en fait un champ de bits. Comme le montre la figure suivante, chaque bit indique individuellement si le fichier est protégé en écriture, si c'est un fichier système, ou si on est en présence d'un sous-répertoire. Comment prendre connaissance de cette information ?



Pour accéder à un bit il faut connaître sa valeur comme puissance de 2. Le bit 0 a la valeur 1, le bit 1 la valeur 2^1=2, le bit 2 la valeur 2^2=4 et ainsi de suite jusqu'au bit 7 dont la valeur est 2^7=128. Pour savoir si l'attribut étudié désigne un sous-répertoire, il faut faire intervenir la valeur du bit 4, soit le nombre 16.

Notre préoccupation va être de mettre à 0 tous les autres bits de l'attribut de façon que seul le bit 4 reste présent pour être testé. Il suffit de combiner l'attribut avec un "masque" par l'opérateur binaire AND : tous les bits qui ne seront pas à 1 dans le masque seront mis à 0 dans le résultat. En Turbo Pascal on écrira :

```
Attribut and 16
```

Le résultat de cette opération sera 16 si le bit 4 est à 1, et 0 dans tous les autres cas.

Avec une instruction If, on aura :

```
If  Attribut and 16  <> 0  then
  { ceci est un sous-répertoire }
else
  { ceci n'est pas un sous-répertoire }
```

Lorsqu'on désire tester plusieurs bits simultanément, l'exercice est un peu plus compliqué. Il faut alors faire la somme des puissances de 2 représentées par les différents bits. Supposons que vous cherchiez à savoir si un fichier donné est un fichier système caché. Les indicateurs correspondants sont les bits 1 et 2 qui ont pour valeur totale $2^1 + 2^2$=6.

L'expression

```
Attribut and 6
```

retourne le contenu des deux bits qui nous intéressent. Mais attention! Si vous reprenez un test If semblable au précédent, il ne donnera pas forcément le résultat escompté car :

```
Attribut and 6 <> 0
```

est déjà TRUE si un seul des deux bits est à 1 et si de ce fait le résultat de l'opération AND est différent de 0. Pour vérifier si les deux bits sont simultanément à 1, il faut écrire :

```
If Attribut and 6 = 6 then
   { c'est un fichier système caché }
    else
   { ce n'est pas un fichier système caché }
```

Le programmeur ne se contente pas toujours de lire des bits, parfois il est obligé de les fixer d'autorité, notamment pour communiquer certains champs de bits à des fonctions de DOS ou du BIOS. Là encore c'est la valeur des bits calculée en puissances de deux qui intervient mais cette fois dans des opérations OR. Pour mettre à 1 le bit 3 de l'octet qui représente un attribut de fichier, on écrira en Turbo Pascal :

```
Attribut = Attribut or 8
```

S'il faut fixer plusieurs bits à 1, leurs puissances devront être additionnées, par exemple

```
Attribut := Attribut or (8+16);
```

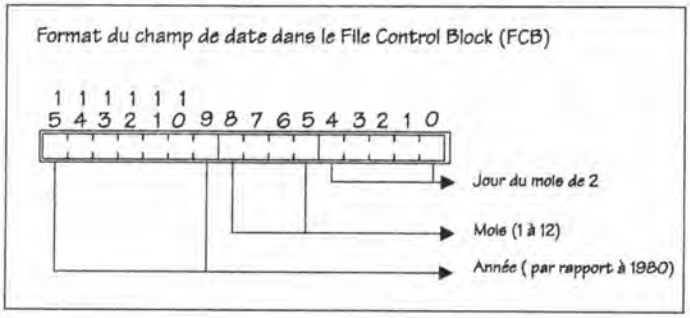
Mais comment faire pour mettre à 0 des bits ? On refait alors appel à un opérateur AND mais exploité d'une manière différente car il s'agit de sélectionner uniquement le bit visé. Les règles de la logique binaire nous apprennent qu'il faut alors inverser la valeur associée au bit en utilisant l'opérateur NOT. Pour mettre à 0 le bit 5, on écrira :

```
Attribut := Attribut and not (32);
```

Pour mettre à 0 plusieurs bits en même temps, on pratique l'addition des puissances de deux correspondantes, par exemple :

```
Attribut := Attribut and not (32+8);
```

Un champ binaire n'est pas toujours constitué de bits isolés, il peut être fait de groupes de bits qui expriment une valeur. A titre d'exemple, examinons le champ date d'une entrée de répertoire. Il contient trois groupes de bits qui indiquent la date de création ou de dernière mise à jour de chaque fichier. Pour exploiter cette information il est inutile de tester individuellement les bits, il faut obtenir la valeur représentée par les trois groupes.



Le jour est facile à calculer avec les méthodes précédemment étudiées et l'opérateur AND :

```
Jour := Date and ( 1 + 2 + 4 + 8);
```

Pour calculer le mois, l'opérateur AND ne suffit plus. Le goupe de bits doit en effet être décalé de cinq positions vers la droite avant de donner le numéro du mois. En Turbo Pascal le décalage de n positions vers la droite se fait au moyen de l'opérateur SHR. Le mois et l'année s'obtiennent donc de la façon suivante :

```
Mois := ( Date and ( 32 + 64 + 128 + 256 ) ) shr 5;
Annee := ( Date and ( 512+1024+2048+4096+8192+16384+32768 ) ) shr 9;
```

Pour déplacer des bits non pas vers la droite mais vers la gauche , on utilise l'opérateur symétrique SHL. Si on décide ainsi de reconstituer un champ date à partir du jour, du mois et de l'année, on écrira :

```
Date := Jour + ( Mois shl 5 ) + ( Annee shl 9 );
```

2.2.2. Appel des interruptions

Pour déclencher des interruptions logicielles, Turbo Pascal propose deux instructions appelées INTR et MSDOS qui sont définies dans l'unité DOS. Cette unité comporte également quelques déclarations de types et de constantes nécessitées par les appels.

La syntaxe de INTR est la suivante :

```
Intr( NumInterrupt: byte, Regs: Registers);
```

Le paramètre NumInterupt désigne le numéro de l'interruption déclenchée. Ce numéro peut prendre toute valeur comprise entre 0 et 255 : c'est donc l'intégralité des interruptions tant matérielles (Il n'est généralement pas raisonnable de déclencher une interruption matérielle par ce moyen, les risques de déséquilibrage du système, autre-

- ✓ Regs.ax,
- ✓ Regs.bx
- ✓ Regs.cx
- ✓ Regs.ah
- ✓ Regs.dl, etc...

S'il faut donner à DL la valeur $D3 avant de déclencher une certaine interruption, on écrira simplement :

```
Regs.DL:=$D3;
```

Vous pouvez exploiter des expressions plus complexes car Regs.DL est une variable tout à fait ordinaire, et il en est de même des autres contenus de registres. Préalablement à un ordre d'interruption INTR ou MSDOS ne chargez que les registres qui servent effectivement d'arguments dans l'interruption car les autres seront royalement ignorés.

Lecture des indicateurs du registre des indicateurs

Dans beaucoup de cas ce ne sont pas seulement les registres généraux AX,BX,CX, etc... mais aussi les indicateurs du registre des indicateurs qui renvoient des informations au programme appelant. Les fonctions de DOS font un large usage de l'indicateur de retenue qui signale une erreur lorsqu'il est à 1 après exécution de la fonction.

Pour faciliter l'évaluation des différents indicateurs, l'unité DOS définit différentes constantes qui en reflètent la valeur.

Constante	Position	Puissance de deux
FCarry	0	1
FParity	2	4
FAuxiliary	4	16
FZero	6	64
FSign	7	128
FOverflow	11	2048

Pour tester un de ces bits, vous pouvez vous servir de l'opérateur AND comme nous l'avons vu précédemment. Voici par exemple une expression booléenne qui prend la valeur TRUE si l'indicateur de retenue est à 1 :

```
Erreur:=((Regs.Flags and FCarry)<>0);
```

ment dit les chances de plantage, étant très importants) que logicielles qui est ainsi accessible.

La procédure MSDOS est une variante de la procédure INTR et s'écrit :

```
MsDos(Regs: Registers);
```

Le numéro d'interruption n'est pas précisé car il s'agit implicitement du numéro $21 qui permet d'appeler plus de 200 fonctions de l'API de DOS. API veut dire "Application Program Interface" et désigne l'ensemble des fonctions que DOS met à la disposition des programmes d'application.

Accès aux registres du processeur

Les deux procédures INTR et MSDOS attendent un argument de type REGISTERS défini dans l'unité DOS. Avant d'exécuter l'interruption demandée, ces procédures chargent dans les registres du processeur les données mises au préalable dans l'enregistrement REGISTERS. Selon un mécanisme symétrique, à l'issue de l'interruption, le contenu des registres du processeur est transféré dans le même enregistrement.

Pour faciliter l'accès aux registres, REGISTERS est défini comme un enregistrement variable permettant de prendre en considération des valeurs sur 8 bits et sur 16 bits :

```
type Registers = record
                   case integer of
                     0 : (AX, BX, CX, DX, BP, SI, DI, DS, ES, Flags : Word);
                     1 : (AL, AH, BL, BH, CL , CH, DL, DH : byte);
                   end;
```

Les registres de 16 bits allant de AX à ES sont représentés par des variables de même nom ayant le type WORD. Il en est de même des registres de 8 bits (AL, AH, BL, BH...) qui sont représentés par des variables de type BYTE.

L'introduction d'un enregistrement variable se justifie par le rôle important joué par les registres de 8 bits dans la gestion des interruptions. En mémoire centrale, les registres de 8 bits et de 16 bits occupent la même zone, ils se recouvrent donc complètement : chaque paire de "8 bits" se situe à la même place que le "16 bits" associé. C'est ainsi que AL et AH se partagent l'emplacement mémoire de AX, BL et BH se tiennent à l'emplacement de BX, et ainsi de suite pour CL/CH et DL/DH.

Notez bien l'ordre dans lequel sont donnés les registres de 8 bits : il doit refléter le format du registre de 16 bits superposé. Comme l'octet de poids faible précède en mémoire l'octet de poids fort, le registre -L doit être déclaré avant le registre -H associé.

Si REGS est une variable de type REGISTERS, les différents registres du processeur peuvent être adressés de la façon suivante :

Réception de pointeurs renvoyés par les fonctions d'interruption

Nous allons étudier un programme symétrique du précédent qui s'appelle GetMedia. Il appelle la fonction $1B de DOS qui n'attend aucun pointeur mais en retourne un dans les registres DS:BX. Ce pointeur référence un octet qui contient le code d'identification du support (media-ID) du lecteur courant. Ce code, compris entre 0xF0 et 0xFF, décrit de façon interne à DOS le type du lecteur en service. La valeur 0xF8 (248) caractérise les disques durs de toutes sortes.

Au début du programme le type MediaPtr est défini comme pointeur sur un octet. En Turbo Pascal les pointeurs sont toujours FAR, aussi sommes-nous absolument sûrs d'avoir bien créé un pointeur FAR. La variable MP est définie avec ce type pour recevoir à l'issue de l'interruption le contenu des registres DS:BX. On met en service à cet effet la fonction PTR de Turbo Pascal qui forme un pointeur générique (Les pointeurs génériques peuvent pointer sur n'importe quel objet mémoire, ils ne sont pas liés à des types de variables précis) en partant d'un segment et d'un offset.

Une fois qu'on dispose du pointeur correctement rempli, on peut l'exploiter dans les conditions habituelles, ici à l'intérieur d'une instruction WRITELN :

```
program GetMedia;

uses DOS;

type MediaPtr = ^byte;

var Regs : Registers;
    MP   : MediaPtr;

begin
  Regs.AH := $1B;
  MsDos( Regs );
  MP := ptr( Regs.DS, Regs.BX );
  writeln( 'Code du support du lecteur actif: ', MP^ );
end.
```

Accès à la mémoire par MEM et MEMW

Pour être honnête, il faut avouer que cet exemple aurait pu se programmer plus simplement. Pour accéder à des octets, des mots ou des entiers longs (DWord), Turbo Pascal dispose de trois tableaux prédéfinis appelés mem, memw et meml. Les éléments de ces tableaux sont référencés d'une manière toute particulière : à l'intérieur des crochets on met le segment et l'offset séparés par deux-points.

2.2.3. Du bon usage des buffers

De nombreuses fonctions utilisent comme argument un pointeur sur un buffer qui sert de zone d'échange pour lire ou écrire des données. Ces pointeurs sont toujours FAR : ils sont constitués d'un segment et d'un offset parce que les données ne se trouvent pas forcément dans le segment du programme courant, elles peuvent être dispersées n'importe où dans la mémoire.

Transmission des pointeurs aux fonctions d'interruption

Prenons comme exemple la fonction numéro $09 de DOS qui affiche une chaîne de caractères à la position courante du curseur. Comme toutes les fonctions DOS elle s'attend à trouver son numéro d'appel ($09) dans le registre AH. Par ailleurs l'adresse du buffer contenant la chaîne à afficher doit être mémorisée dans les registres DS:DX. DS est destiné à la partie segment et DX à l'offset.

Créer une chaîne n'est pas difficile en Turbo Pascal mais comment trouver l'adresse du buffer dont nous avons besoin ? En réalité, c'est très simple car Turbo Pascal dispose de deux fonctions SEG() et OFS() qui renvoient justement le segment et l'offset de n'importe quel objet en mémoire. Peu importe s'il s'agit d'une variable locale ou globale, ou d'une constante typée.

Le programmme suivant montre comment ces fonctions peuvent être mises en service dans le cadre d'un appel à la fonction $09 de DOS. Notez bien que contrairement à la plupart des autres fonctions de DOS, la fonction $09 demande que la chaîne traitée se termine par un $ (et non par un caractère nul) :

```
program DOSPrint;

uses DOS;

var Regs    : Registers;
    Message : string[20];

begin
  Message := 'DOSPrint' + '$';

  Regs.AH := $09;
  Regs.DS := seg( Message[1] );
  Regs.DX := ofs( Message[1] );
  MsDos( regs );
end.
```

Les programmes en C de cet ouvrage ont été conçus de manière à fonctionner avec tous les compilateurs qui viennent d'être mentionnés. Il n'est cependant pas exclu que l'un ou l'autre émette de temps en temps un message d'avertissement sans importance. Mais cette situation devrait être rarissime car sous Microsoft les programmes ont été compilés avec le niveau d'avertissement W2 tandis que sous Borland les paramètres par défaut de l'environnement de développement ont été repris tels quels.

Les programmes présentent cependant des différences dues à des bibliothèques dissemblables. C'est pourquoi ils présenteront parfois l'aspect suivant , comme c'est le cas par exemple du programme DIRC2.C du chapitre 20. Les différences sont interceptées par des macros :

```
#ifdef __TURBOC__                          /* Compilateur Turbo C? */
  #define DIRSTRUCT                      struct ffblk
  #define FINDFIRST( path, buf, attr )   findfirst( path, buf, attr )
  #define FINDNEXT( buf )                findnext( buf )
  #define NAME                           ff_name
  #define ATTRIBUT                       ff_attrib
  #define TIME                           ff_ftime
  #define DATE                           ff_fdate
  #define SIZE                           ff_fsize
#else                                        /* Non, Microsoft C */
  #define DIRSTRUCT                      struct find_t
  #define FINDFIRST( path, buf, attr )   _dos_findfirst(path, attr, buf)
  #define FINDNEXT( buf )                _dos_findnext( buf )
  #define NAME                           name
  #define ATTRIBUT                       attrib
  #define TIME                           wr_time
  #define DATE                           wr_date
  #define SIZE                           size
#endif
```

Les caractéristiques de l'orientation objets de C++ n'ont pas été exploitées ici pour deux raisons très précises. La première tient au retard de Microsoft qui à l'heure de la publication de cet ouvrage n'a toujours pas réussi à mettre au point un compilateur C++. La deuxième raison est due à la volonté de ne pas occulter la substance profonde des programmes par des définitions de classes et de hiérarchies d'objets.

Mais cela ne veut pas dire que ce type de programmation ne puisse pas tirer profit d'un environnement d'objets, bien au contraire.

Pensez par exemple à un programme destiné à gérer plusieurs interfaces série. Les routines sont toujours les mêmes, seules les adresses des ports sont différentes. Il est facile d'imaginer un type d'objet qui gère une interface série en mémorisant dans une variable objet l'adresse du port. Pour gérer plusieurs interface, il suffirait de créer plusieurs instances de ce type d'objet en leur affectant l'adresse du port qui convient.

Après tous ces préambules, il est temps de se concentrer sur la programmation système en C.

Dans le cas qui nous occupait, nous aurions pu récupérer le code du support de la façon suivante :

```
mem [Regs.DX : Regs.BX]
```

Mais cette simplification ne doit pas faire illusion : mem, memw et meml atteignent rapidement leurs limites lorsque les pointeurs à traiter référencent des structures complexes. Il faut alors inévitablement passer par la déclaration d'un enregistrement et du pointeur associé.

2.2.4. Accès aux ports d'entrée-sortie

Turbo Pascal voit également les ports d'entrée-sortie comme des tableaux prédéfinis. Il existe en fait deux tableaux appelés PORT pour les ports de 8 bits et PORTW pour les ports de 16 bits. La seule différence réside dans la possibilité de traiter des données sur 16 bits avec PORTW tandis que PORT ne gère que des nombres de 8 bits. Le choix de l'un ou l'autre de ces tableaux dépendra des caractéristiques matérielles de la carte que l'on désire commander. Si c'est une carte 16 bits, on choisira PORTW, sinon on en est réduit à prendre PORT.

Ces tableaux utilisent la syntaxe ordinaire des tableaux courants de Turbo Pascal. Par exemple pour lire le contenu du port $3C4 (contrôleur graphique d'une carte EGA ou VGA), on écrira :

```
XByte := port[ $3C4 ];
XWord := portw[ $3C4 ];
```

L'émission d'un octet ou d'un mot est tout aussi simple:

```
port[ $3C4 ] := XByte;
portw[ $3C4 ] := XWord;
```

Vous trouverez des exemples d'application de ces instructions au chapitre XX, dans le cadre de la programmation des cartes EGA et VGA.

2.3. Programmation système en C

Contrairement au cas de Pascal, le marché des compilateurs C se caractérise par une rivalité entre Microsoft et Borland. Les deux maisons envoient plusieurs chevaux dans la course : Microsoft bichonne son QuickC et le gros compilateur MSC, tandis que Borland accorde tous ses soins à Turbo C++ et à Borland C++ qui sont des systèmes complets pour développer des programmes en C et en C++. Turbo C n'existe plus officiellement mais heureusement les deux compilateurs C++ conservent la compatibilité avec le standard établi précédemment.

2.3.1. Les types de données de C

Comme c'est le cas de la plupart des compilateurs, les types de données de Turbo Pascal correspondent plus ou moins à ceux du processeur pour permettre un traitement plus simple et plus rapide. Le tableau suivant montre comment les compilateurs C mémorisent leurs différents types de données :

Type en C	Stockage interne
unsigned char	BYTE
char	BYTE
int	WORD
unsigned int	WORD
near *void	WORD
long	DWORD
far *void	DWORD

Il n'existe pas de type BYTE et WORD en C. C'est pourquoi vous trouverez souvent au début des programmes en C présentés dans ce livre des instructions typedef du genre :

```
typedef unsigned char BYTE;
typedef unsigned int WORD;
```

Ces deux lignes définissent les deux types essentiels en programmation système.

En C on ne peut pas dire a priori si les pointeurs sont NEAR ou FAR : cela dépend du modèle mémoire utilisé. Les programmes de ce livre ont été développés avec le modèle SMALL qui fonctionne exclusivement avec des pointeurs NEAR. Lorsque la programmation système requiert des pointeurs FAR, on applique le modificateur far à la déclaration de la variable concernée :

```
int far *p;  /* p est un pointeur far sur un entier */
```

QuickC n'aime pas travailler avec des pointeurs FAR tant que l'option Options/Compiler Flags/Pointer Check est enclenchée. N'oubliez pas de désactiver cette option pour que les programmes de démonstration présentés dans ce livre s'exécutent sans problème.

Chaînes de caractères

Pour la plupart des fonctions de DOS et du BIOS, une chaîne de caractères est une suite de codes ASCII terminés par un caractère "nul" de code 0. Dans la littérature système ce type de chaîne est parfois appelé ASCIIZ, le Z évoquant le zéro terminal.

Le langage C adopte le même format de stockage, c'est pourquoi l'usage de C en programmation système est souvent plus facile que celui de Pascal qui exploite un autre format.

Structures et tableaux

Tout comme les programmes d'application, DOS et le BIOS gèrent de nombreuses informations en s'aidant de structures et de tableaux que l'on est parfois obligé de reproduire dans la programmation système en C. Si vous êtes un habitué des tableaux et des structures vous n'aurez pas de mal à appliquer votre expérience. Ce qui est important c'est que le compilateur range les données dans le bon ordre, qu'elles commencent à une frontière de mot (Les processeurs 80xxx sont plus efficaces lorsque les données à traiter sont situées à des adresses paires, autrement dit lorsqu'elles sont alignées sur des mots-mémoire), et qu'aucune espace libre ne soit laissé entre elles.

Tous les compilateurs C pour PC gèrent un paramètre qui permet de "compacter" les structures. Dans le cas de Microsoft C, le paramètre s'écrit /ZP et a pour effet de ne pas tolérer d'espace libre entre les composants d'une structure. L'environnement de développement intégré de Borland possède une commande Options/Compiler/Code génération qui déclenche une boîte de dialogue dans laquelle il est possible de fixer l'option "WORD-ALignement". Veillez à ce que cette option soit désactivée pour que le compilateur n'étire pas les structures.

La figure suivante décrit un exemple de structure renvoyée par des fonctions DOS. Il s'agit d'informations servant à parcourir les répertoires à la recherche de fichiers :

Structure d'une entrée de répertoire retournée par les fonctions 4Eh et 4Fh de DOS

Adresse	Contenu	Type
+00h	réservé	21 BYTE
+15h	Attribut du fichier	1 BYTE
+16h	Heure de la dernière mise à jour	1 WORD
+18h	Date de la dernière mise à jour	1 WORD
+1Ah	Taille du fichier	1 DWORD

Structure d'une entrée de répertoire retournée par les fonctions 4Eh et 4Fh de DOS		
Adresse	**Contenu**	**Type**
+1Eh	Nom du fichier et extension séparés par un point sans indication de chemin d'accès terminaison = caractère nul	13 BYTE
Longueur : 43 octets		

L'extrait de programme suivant montre comment cette structure peut être exprimée en C :

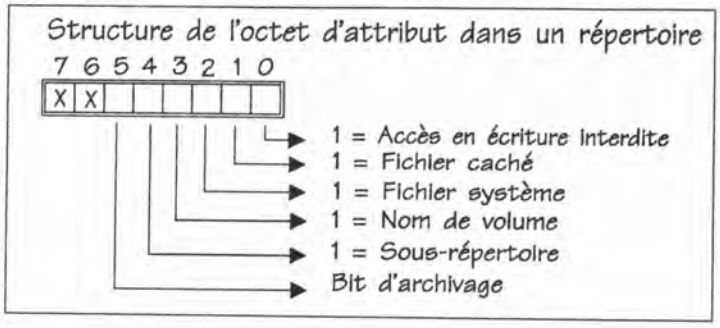
```
typedef unsigned char BYTE;                    /* création du type BYTE */
typedef struct {          /* Structure DIR des fonctions 4Eh et 4Fh */
                BYTE          Reserve[21];
                BYTE          Attribut;
                unsigned int  Time;
                unsigned int  Date;
                unsigned long Size;
                char          Name[13];
               } DIRSTRUCT;
```

La zone réservée en tête de structure est rendue par un tableau qui n'est pas forcément un tableau de caractères CHAR mais peut aussi être un tableau de BYTEs. Il faut simplement veiller à ce que les champs en C respectent l'adresse relative à l'origine de la structure de DOS.

Les autres composants de la structure de DOS sont traduits en C selon la règle suivante : les octets deviennent des BYTEs, les mots (WORD) des types unsigned int, les mots doubles (DWORD) des types unsigned long. Les noms des champs n'ont aucune importance et peuvent être choisis librement car ils n'influencent en rien la structure.

Accès aux bits

Les structures comportent parfois des champs de bits, où des bits isolés ou des groupes de bits ont une certaine signification. Par exemple l'octet représentant l'attribut d'un fichier dans la structure de répertoire présentée plus haut est en fait un champ de bits. Comme le montre la figure suivante, chaque bit indique individuellement si le fichier est protégé en écriture, si c'est un fichier système, ou si on est en présence d'un sous-répertoire. Comment prendre connaissance de cette information ?



Pour accéder à un bit il faut connaître sa valeur comme puissance de 2. Le bit 0 a la valeur 1, le bit 1 la valeur 2^1=2, le bit 2 la valeur 2^2=4 et ainsi de suite jusqu'au bit 7 dont la valeur est 2^7=128. Pour savoir si l'attribut étudié désigne un sous-répertoire, il faut faire intervenir la valeur du bit 4, soit le nombre 16.

Notre préoccupation va être de mettre à 0 tous les autres bits de l'attribut de façon que seul le bit 4 reste présent pour être testé. Il suffit de combiner l'attribut avec un "masque" par l'opérateur binaire AND : tous les bits qui ne seront pas à 1 dans le masque seront mis à 0 dans le résultat. En C on écrira :

```
Attribut & 16
```

Le résultat de cette opération sera 16 si le bit 4 est à 1, et 0 dans tous les autres cas.

Avec une instruction If, on aura :

```
If  ((Attribut & 16 ) != 0)
  /* ceci est un sous-répertoire */
else
  { ceci n'est pas un sous-répertoire }
```

Lorsqu'on désire tester plusieurs bits simultanément, l'exercice est un peu plus compliqué. Il faut alors faire la somme des puissances de 2 représentées par les différents bits. Supposons que vous cherchiez à savoir si un fichier donné est un fichier système caché. Les indicateurs correspondants sont les bits 1 et 2 qui ont pour valeur totale $2^1 + 2^2$=6.

L'expression

```
Attribut & 6
```

retourne le contenu des deux bits qui nous intéressent. Mais attention! Si vous reprenez un test If semblable au précédent, il ne donnera pas forcément le résultat escompté car :

```
(Attribut & 6) != 0
```

est déjà TRUE si un seul des deux bits est à 1 et si de ce fait le résultat de l'opération AND est différent de 0. Pour vérifier si les deux bits sont simultanément à 1, il faut écrire :

```
If (Attribut & 6)== 6 then
   { c'est un fichier système caché }
    else
   { ce n'est pas un fichier système caché }
```

Le programmeur ne se contente pas toujours de lire des bits, parfois il est obligé de les fixer d'autorité, notamment pour communiquer certains champs de bits à des fonctions de DOS ou du BIOS. Là encore c'est la valeur des bits calculée en puissances de deux qui intervient mais cette fois dans des opérations OR. Pour mettre à 1 le bit 3 de l'octet qui représente un attribut de fichier, on écrira en C :

```
Attribut = Attribut | 8 ;   ou plus brièvement:
Attribut |= 8 ;
```

S'il faut fixer plusieurs bits à 1, leurs puissances devront être additionnées, par exemple

```
Attribut = Attribut | (8+16) ;
```

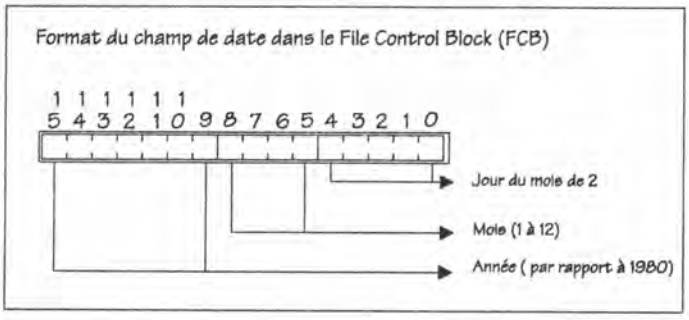
Mais comment faire pour mettre à 0 des bits ? On refait alors appel à un opérateur AND mais exploité d'une manière différente car il s'agit de sélectionner uniquement le bit visé. Les règles de la logique binaire nous apprennent qu'il faut alors inverser la valeur associée au bit en utilisant l'opérateur NOT (symbolisé en C par !). Pour mettre à 0 le bit 5, on écrira :

```
Attribut = Attribut & !32;
```

Pour mettre à 0 plusieurs bits en même temps, on pratique l'addition des puissances de deux correspondantes, par exemple :

```
Attribut = Attribut & !(32+8);
```

Un champ binaire n'est pas toujours constitué de bits isolés, il peut être fait de groupes de bits qui expriment une valeur. A titre d'exemple, examinons le champ date d'une entrée de répertoire. Il contient trois groupes de bits qui indiquent la date de création ou de dernière mise à jour de chaque fichier. Pour exploiter cette information il est inutile de tester individuellement les bits, il faut obtenir la valeur représentée par les trois groupes.



Le jour est facile à calculer avec les méthodes précédemment étudiées et l'opérateur AND :

```
Jour = Date & ( 1 + 2 + 4 + 8) ;
```

Pour calculer le mois, l'opérateur AND ne suffit plus. Le goupe de bits doit en effet être décalé de cinq positions vers la droite avant de donner le numéro du mois. En C le décalage de n positions vers la droite se fait au moyen de l'opérateur >>. Le mois et l'année s'obtiennent donc de la façon suivante :

```
Mois = ( Date & ( 32 + 64 + 128 + 256 ) ) >> 5;
Annee = ( Date & ( 512+1024+2048+4096+8192+16384+32768 ) ) >> 9;
```

Pour déplacer des bits non pas vers la droite mais vers la gauche , on utilise l'opérateur symétrique <<. Si on décide ainsi de reconstituer un champ date à partir du jour, du mois et de l'année, on écrira :

```
Date = Jour + ( Mois << 5 ) + ( Annee << 9 );
```

2.3.2. Appel des interruptions

Pour déclencher les interruptions logicielles les compilateurs de Borland et de Microsoft disposent tous des fonctions int86(), int86x(), intdos(), intdosx(). Les fonctions int86() et int86x() permettent d'appeler les 256 interruptions du processeur Intel, alors que intdos() et intdos() se concentrent uniquement sur l'interruption 0x21 qui ouvre la porte à plus de 200 fonctions de l'API du DOS. API veut dire "Application Program Interface" et désigne l'ensemble des fonctions que le DOS met à la disposition des programmes d'application.

Les déclarations des fonctions d'interruption se trouvent, chez l'un et l'autre compilateur, dans le fichier d'en-tête DOS.H qui doit être inclus dans le programme en C :

```
int intdos(union REGS *inregs, union REGS *outregs);
int intdosx(union REGS *inregs, union REGS *outregs, struct SREGS
```

```
*sreg);
int int86(int, union REGS *inregs, union REGS *outregs);
int int86x(int, union REGS *inregs, union REGS *outregs, struct SREGS
*sreg);
```

Accès aux registres du processeur

Comme vous le voyez à travers leurs déclarations, les fonctions attendent comme arguments des pointeurs sur des structures de type REGS (SREGS pour les fonctions avec un X). Ces structures imitent les registres du processeur. Avant d'exécuter l'interruption demandée, les fonctions chargent dans les registres du processeur les données mises au préalable dans le premier argument inregs. Selon un mécanisme symétrique, à l'issue de l'interruption, le contenu des registres du processeur est transféré dans le deuxième argument outreg.

Pour faciliter l'accès aux registres, le type REGS est défini comme une union de deux structures en recouvrement, de types WORDREGS et BYTEREGS, ce qui permet de prendre en considération des valeurs sur 8 bits et sur 16 bits :

```
union REGS {
           struct WORDREGS x;
           struct BYTEREGS h;
          };

struct WORDREGS {
                 unsigned int ax;
                 unsigned int bx;
                 unsigned int cx;
                 unsigned int dx;
                 unsigned int si;
                 unsigned int di;
                 unsigned int cflag;
                };

struct BYTEREGS {
                 unsigned char al, ah;
                 unsigned char bl, bh;
                 unsigned char cl, ch;
                 unsigned char dl, dh;
                };
```

Les registres de 16 bits allant de AX à DI sont représentés dans WORDREGS par des variables de même nom ayant le type unsigned int. Il en est de même des registres de 8 bits (AL, AH, BL, BH...) qui sont représentés par des variables de type unsigned char dans BYTEREGS.

L'introduction d'une structure alternative se justifie par le rôle important joué par les registres de 8 bits dans la gestion des interruptions. En mémoire centrale, les registres

de 8 bits et de 16 bits occupent la même zone, ils se recouvrent donc complètement : chaque paire de "8 bits" se situe à la même place que le "16 bits" associé. C'est ainsi que AL et AH se partagent l'emplacement mémoire de AX, BL et BH se tiennent à l'emplacement de BX, et ainsi de suite pour CL/CH et DL/DH.

Notez bien l'ordre dans lequel sont donnés les registres de 8 bits : il doit refléter le format du registre de 16 bits superposé. Comme l'octet de poids faible précède en mémoire l'octet de poids fort, le registre -L doit être déclaré avant le registre -H associé.

Si pregs est une variable de type REGS, les différents registres du processeur peuvent être adressés de la façon suivante :

- ✓ pregs.x.ax,
- ✓ pregs.x.bx
- ✓ pregs.x.cx
- ✓ pregs.h.ah
- ✓ pregs.h.dl, etc...

S'il faut donner à DL la valeur $D3 avant de déclencher une certaine interruption, on écrira simplement :

```
pregs.h.dl:=0xD3;
```

Vous pouvez exploiter des expressions plus complexes car pregs.h.dl est une variable tout à fait ordinaire, et il en est de même des autres contenus de registres. Préalablement à un ordre d'interruption, ne chargez que les registres qui servent d'arguments dans l'interruption désignée car les autres seront royalement ignorés.

Prise en compte des registres de segment

Comme le montrent les définitions de BYTEREGS et WORDREGS, ces structures ne prennent en compte que les registres généraux, laissant de côté les registres de segment qui n'interviennent que rarement dans les appels de fonctions. En cas de besoin, vous pouvez cependant recourir aux fonctions int86x() et intdos() qui attendent non seulement deux pointeurs sur des variables de type REGS mais aussi un pointeur sur une variable de type SREGS. Avant de déclencher l'interruption demandée, les deux fonctions chargent les registres de segment avec les valeurs mises dans la variable de type SREGS et inversement au retour de l'interruption le contenu des registres du processeur y est transféré.

La définition de SREGS est la suivante :

```
struct SREGS {
              unsigned int es;
              unsigned int cs;
```

```
    unsigned int ss;
    unsigned int ds;
};
```

Lecture des indicateurs du registre des indicateurs

Dans beaucoup de cas ce ne sont pas seulement les registres généraux AX,BX,CX, etc... mais aussi les indicateurs du registre des indicateurs qui renvoient des informations au programme appelant. Les fonctions de DOS font un large usage de l'indicateur de retenue qui signale une erreur lorsqu'il est à 1 après exécution de la fonction.

Pour faciliter l'évaluation de l'indicateur de retenue, la structure WORDREGS contient un champ dénommé CFLAG qui au retour de l'interruption est chargé avec la valeur de l'indicateur. Lorsque l'indicateur est armé, le champ a la valeur 1, sinon il est nul. Sa valeur initiale n'est pas prise en compte, ce n'est qu'au retour de l'interruption que cet indicateur joue un rôle.

Le programme suivant montre qu'il est facile de tester si l'indicateur de retenue est positionné à 1 à l'issue d'une interruption. Vous constaterez également qu'une seule et même variable est indiquée à la fois comme paramètre inregs et outregs. Cette variable reçoit les valeurs d'entrée et fournit le résultat testé en sortie :

```
#include <dos.h>

void test( void )
{
 union REGS pregs;

 pregs.h.ah = 0x13;          /* Numéro de fonction quelconque*/
 pregs.h.dl = 0;             /* Valeur d'entrée */
 intdos( &pregs, &pregs );
 if ( pregs.x.cflag )
  ;                          /* Indicateur de retenue positionné à 1 */
 else
  ;                          /* Indicateur de retenue positionné à 0 */
}
```

L'indicateur de retenue n'est pas le seul indicateur existant. Le problème se complique pour les autres car certaines fonctions du BIOS (rares, il est vrai) exploitent l'indicateur de zéro pour restituer une information. Les fonctions int...() de Microsoft ne permettent pas de progresser sur ce plan. Mais les développeurs de Borland ont été suffisamment malins pour étendre la structure WORDREGS en y ajoutant une variable FLAGS qui au retour de l'interruption reflète l'état des indicateurs.

```
struct WORDREGS {                    /* Borland uniquement ! */
                 unsigned int ax, bx, cx, dx, si, di, cflag, flags;
                };
```

Pour tester l'un des indicateurs du registre des indicateurs, vous pouvez donc, si vous utilisez un compilateur Borland, combiner par AND la variable flags avec la puissance de deux de l'indicateur souhaité. Pour mémoire, voici la table des puissances de deux associées aux différents indicateurs :

Indicateur	Position	Puissance de deux
Retenue (Carry)	0	1
Parité	2	4
Auxiliaire	4	16
Zéro	6	64
Signe	7	128
Débordement (Overflow)	11	2048

2.3.3. Du bon usage des buffers

De nombreuses fonctions utilisent comme argument un pointeur sur un buffer qui sert de zone d'échange pour lire ou écrire des données. Ces pointeurs sont toujours FAR : ils sont constitués d'un segment et d'un offset parce que les données ne se trouvent pas forcément dans le segment du programme courant, elles peuvent être dispersées n'importe où dans la mémoire.

Transmission des pointeurs aux fonctions d'interruption

Prenons comme exemple la fonction numéro 0x09 de DOS qui affiche une chaîne de caractères à la position courante du curseur. Comme toutes les fonctions DOS elle s'attend à trouver son numéro d'appel (0x09) dans le registre AH. Par ailleurs l'adresse du buffer contenant la chaîne à afficher doit être mémorisée dans les registres DS:DX. DS est destiné à la partie segment et DX à l'offset.

Créer une chaîne n'est pas difficile en C mais comment trouver l'adresse du buffer dont nous avons besoin ? En réalité, c'est très simple car les compilateurs C de Borland et de Microsoft disposent tous deux des macros FP_SEG() et FP_OFF() qui renvoient justement une adresse de segment et un offset. Malheureusement les implémentations diffèrent chez les deux sociétés, ce qui ne manque pas de poser problème :

Borland:

```
#define FP_SEG(fp) ((unsigned)(void _seg *)(void far *)(fp))
#define FP_OFF(fp) ((unsigned)(fp))
```

Microsoft:

```
#define FP_SEG(fp) (*((unsigned _far *)&(fp)+1))
#define FP_OFF(fp) (*((unsigned _far *)&(fp)))
```

Dans la définition de Borland, les deux macros sont appelées directement avec la variable dont on recherche le segment et l'offset. Dans la version de Microsoft, il faut leur communiquer un pointeur FAR qui référence la variable.

Le programme suivant montre comment ces fonctions peuvent être mises en service dans le cadre d'un appel à la fonction 0x09 de DOS. Notez bien que contrairement à la plupart des autres fonctions de DOS, la fonction 0x09 demande que la chaîne traitée se termine par un $ (et non par un caractère nul) :

Voici d'abord la version Borland :

```
#include <dos.h>        /* Version BORLAND */

void main( void )
{
 union REGS pregs;
 struct SREGS sregs;
 char message[20] = "DOSPrint$";

 pregs.h.ah = 0x09;
 sregs.ds = FP_SEG( message );          /* Recherche l'adresse */
 pregs.x.dx = FP_OFF( message );        /* par la variable  */
 intdosx( &pregs, &pregs, &sregs );
}
```

Vous voyez que l'adresse de "message" est obtenue en indiquant directement le nom de la variable aux macros FP_SEG() et FP_OFF(). Avec les compilateurs de Microsoft, il n'est pas possible de procéder ainsi. Il faut intercaler une variable qui référence la chaîne sous forme de pointeur FAR :

```
#include <dos.h>        /* Version MICROSOFT */

void main( void )
{
 union REGS pregs;
 struct SREGS sregs;
 char message[20] = "DOSPrint$";
 void far *mesptr = message;        /* Pointeur FAR sur la chaîne*/

 pregs.h.ah = 0x09;
 sregs.ds = FP_SEG( mesptr );           /* Recherche l'adresse */
 pregs.x.dx = FP_OFF( mesptr );         /* par le pointeur FAR */
 intdosx( &pregs, &pregs, &sregs );
}
```

Réception de pointeurs renvoyés par les fonctions d'interruption

Nous allons étudier un programme symétrique du précédent qui s'appelle GetMedia. Il appelle la fonction 0x1B de DOS qui n'attend aucun pointeur mais en retourne un dans les registres DS:BX. Ce pointeur référence un octet qui contient le code d'identification du support (media-ID) du lecteur courant. Ce code, compris entre 0xF0 et 0xFF, décrit de façon interne à DOS le type du lecteur en service. La valeur 0xF8 (248) caractérise les disques durs de toutes sortes.

L'adresse du code du support est renvoyée sous forme de pointeur FAR dans les registres DS:BX. Mais comment en déduire un pointeur FAR en C ? La solution des compilateurs BORLAND est une macro MK_FP() définie dans le fichier d'inclusion DOS.H. Cette macro attend deux paramètres, le segment et l'offset du pointeur à former. Ce dernier constitue le résultat de la macro et présente le type void far*.

Les compilateurs Microsoft ne connaissent pas cette macro mais il est facile de se la bricoler soi-même, comme le montre le listing ci-dessous, qui définit MK_FP si elle fait défaut.

Après l'interruption, le pointeur FAR constitué par MK_FP() est affecté à la variable mp. Dans la fonction printf() située à la fin du programme le code du support est déréférencé au moyen de ce pointeur et affiché à l'écran.

```
#include <dos.h>
#include <stdio.h>

#ifndef MK_FP                    /* La macro MK_FP est-elle définie ? */
  #define MK_FP(seg,ofs) ((void far *) ((unsigned long)
(seg)<<16|(ofs)))
#endif

void main( void )
{
 union REGS pregs;
 struct SREGS sregs;
 unsigned char far *mp;

 pregs.h.ah = 0x1B;
 intdosx( &pregs, &pregs, &sregs );
 mp = MK_FP( sregs.ds, pregs.x.bx );
 printf( "Code du support du lecteur actif: %d\n ", *mp );
}
```

Si vous examinez attentivement la définition de la macro MK_FP(), vous observerez que malgré le grand nombre de parenthèses et de mots-clés son mécanisme reste simple. L'adresse de segment est convertie en un entier long, décalée de 16 bits vers la gauche, après quoi les 16 bits de poids faible sont remplis par l'offset. Le résultat est bien un pointeur FAR ayant la structure désirée il suffit encore de le déclarer comme tel.

2.3.4. Accès aux ports d'entrée-sortie

Les compilateurs de Microsoft et de Borland proposent plusieurs fonctions pour accéder aux ports d'entrée-sortie. Ces fonctions portent des noms différents et ne sont pas déclarés dans les mêmes fichiers d'inclusion. Borland les place dans DOS.H, tandis que Microsoft les met dans CONIO.H où à mon avis ils n'ont rien à faire.

Le tableau suivant indique les différentes routines avec leurs déclarations :

Microsoft: Fichier d'inclusion <conio.h>

```
int      inp( unsigned port );
unsigned inpw( unsigned port );
int      outp( unsigned port, int databyte );
unsigned outpw( unsigned port, unsigned dataword );
```

Borland: Fichier d'inclusion <dos.h>

```
int           inport (int __portid);
unsigned char inportb(int __portid);
void          outport (int __portid, int __value);
void          outportb(int __portid, unsigned char __value);

#define inp(portid) inportb(portid)
#define outp(portid,v) outportb(portid,v)
```

Vous voyez qu'il existe à chaque fois deux fonctions de lecture et deux fonctions d'écriture, l'une associées aux ports de 16 bits et l'autre aux ports de 8 bits.

Le compilateur de Borland dispose également des macros inp() et outp() qui reprennent les noms des fonctions de Microsoft. Même si cette émulation n'existe que sur 8 bits, il est facile de l'étendre aux fonctions de 16 bits :

```
#ifdef __TURBOC__                          /* Compilation par Turbo C? */
  #define inpw(portid)    inport(portid)
  #define outpw(portid,v) outport(portid,v)
#endif
```

On peut ainsi utiliser les noms de Microsoft tout en travaillant avec un compilateur de Borland. Par exemple pour lire le contenu du port $3C4 (contrôleur graphique d'une carte EGA ou VGA), on écrira :

```
XByte = inp(0x3C4);
XWord := inpw(0x3C4);
```

Il est tout aussi simple d'émettre un octet ou un mot sur ce port :

```
outp( 0x3C4, XByte );
outpw( 0x3C4 XWord );
```

Vous trouverez des exemples d'application de ces instructions au chapitre 4, dans le cadre de la programmation des cartes EGA et VGA.

3. Le BIOS

Le terme "système d'exploitation" est généralement lié au "DOS" pour la grande majorité des utilisateurs de PC. Pourtant, bien avant que le DOS ne soit installé sur disque dur, le PC possède un autre système d'exploitation, correspondant plus étroitement au sens de cette expression : le BIOS. Gravé dans un circuit de ROM, le BIOS est le premier système activé au moment de la mise sous tension du PC.

Ce "Basic Input/Output System" possède les fonctions élémentaires qui permettront à un programme écrit pour PC de communiquer avec la machine et ses périphériques. Il s'agit des routines d'accès à l'écran, à l'imprimante ou encore de consultation de la date et de l'heure du système.

Les services du BIOS sont absolument indispensables aux applications ainsi qu'au DOS lui-même. En effet, le BIOS s'intercale entre les logiciels et les spécificités de la machine. On peut dire qu'il décharge dans des proportions non négligeables toutes les opérations sur les périphériques effectuées par les programmes, qu'il s'agisse du DOS ou de votre logiciel préféré.

Mais ne perdons pas de vue que l'utilisation des fonctions du BIOS n'est pas une pure question de facilité, c'est aussi et surtout une condition impérative pour atteindre une "compatibilité" satisfaisante entre les machines.

Ces différences infimes (un autre positionnement d'un bit, un registre matériel à une adresse différente) rendraient la vie très dure aux développeurs. Qui pourrait écrire des programmes capables de gérer les centaines de disques durs que l'on trouve aujourd'hui sur le marché, et surtout que l'on trouvera demain ? Sans parler des cartes vidéo et autres pilotes de claviers.

Le BIOS est une interface normalisée avec les composants de chaque système. Qu'un système soit équipé d'un disque dur de 20 Mo ou de 4 Go, qu'il soit commercialisé par IBM, Dell, Compaq ou autre, il sera piloté par les mêmes fonctions du BIOS. Inversement, la gestion adoptée par le BIOS est invisible et sans intérêt pour le développeur.

Il est intéressant de noter qu'une très grande partie du gigantesque succès du PC est très vraisemblablement due à ce concept indépendant du matériel. Il permet à tous les fabricants d'ordinateurs de développer des PC qui sont à la fois compatibles avec les logiciels standard et différents des machines fabriquées par IBM, l'auteur de cette interface pour son premier PC et qui l'a ensuite élargie sur l'AT pour permettre de tirer parti des avantages de cette machine.

Cette interface a jusqu'ici été respectée par les fabricants de PC car elle est le critère qui fait d'un ordinateur un PC.

Les BIOS sont écrits par une bonne douzaine de sociétés dont les plus connues sont sans doute AMI, Phoenix, Award et Quadtel. Elles sont plus ou moins performantes dans tel ou tel domaine, mais toutes fournissent des produits utilisant les fonctions de la norme IBM.

Nous traiterons dans ce chapitre de l'affichage, de l'interrogation du clavier et de l'accès à l'horloge du système à l'exception des interfaces mises à disposition par le BIOS. La gestion de ces opérations par le BIOS est sans intérêt pour nous.

Nous commençons par des questions plus fondamentales, par la localisation du BIOS, ses fonctions, ses variables et son comportement pendant le démarrage du système.

3.1. La norme BIOS

La norme du BIOS définie par IBM spécifie le mode d'appel des différentes fonctions (les routines) ainsi que le mode de passage des paramètres. Le concept d'interruption a été choisi pour distinguer l'appelant des autres fonctions BIOS. Chaque fonction ne pouvant avoir son interruption (dont le nombre est limité à 256), diverses interruptions ont été affectées selon la partie adressée du matériel. En outre, le BIOS utilise quelques interruptions comme variables qui stockent par exemple des pointeurs sur des tableaux de vidéo ou de disque dur. Nous reviendrons sur ces points.

Les interruptions du BIOS en ROM

Interruption	Prestation
10h	Accès à la carte vidéo
11h	Renvoie la configuration
12h	Renvoie la taille de la mémoire
13h	Fonctions de disquettes et disque dur
14h	Interface série
15h	Fonctions de cassettes et fonctions AT étendues
16h	Clavier
17h	Interface parallèle
1Ah	Date/heure et horloge sur piles

Pour permettre au BIOS d'identifier les fonctions, un numéro, qu'il faut passer au registre AH du processeur pendant l'appel, est affecté à chacune d'entre elles. Ce registre n'est pas le seul que l'on doit charger avant un appel car tout le passage des paramètres est effectué à l'aide des registres du processeur. L'utilisation des registres du processeur fait également partie de la norme du BIOS, et elle est définie fonction par fonction.

Conception et emplacement du BIOS

Outre ses fonctions, l'adresse du BIOS en mémoire est également normalisée. La ROM contenant le code du BIOS est toujours localisée dans le plus haut segment de mémoire F000h, mais aucune définition précise de l'adresse exacte du BIOS dans ce segment. Par exemple, le BIOS original du PC/XT commence à E000h mais le BIOS Phoenix, bien connu, commence dès C000h dans ce segment.

Le début du BIOS est avant tout fonction de sa longueur car il doit impérativement prendre fin à la dernière adresse du segment F : FFFFh. Ceci est la dernière adresse accessible en mode réel par les différents processeurs Intel.

Shadow RAM, paramètres de disque dur, etc

La concurrence effrénée amène les fabricants à tenter de se distinguer par les propriétés de leurs produits. Il en va ainsi des cartes VGA comme des BIOS. Les chevaux de bataille ont pour nom "Shadow RAM", paramètres de disque dur, mot de passe et Setup. Que signifie tout ceci ?

La "Shadow RAM" est une partie de la mémoire cachée derrière le BIOS, qui partage son adresse. La double affectation des adresses de mémoire étant impossible, cette zone est normalement réservée au profit du BIOS. Elle est "invisible", le système d'exploitation, ni les applications ne peuvent y accéder. Le BIOS étant indispensable pour la gestion des périphériques, aucun remplacement du BIOS en ROM n'est possible par le BIOS en RAM.

Pour tirer le meilleur parti possible de cette situation, de nombreux systèmes copient automatiquement le BIOS dans la fameuse "Shadow RAM", derrière le BIOS en ROM. Le code exécuté est celui copié dans la RAM et non plus celui de la ROM, après effacement des adresses de la ROM. Cela reste sans effet sur le fonctionnement du BIOS mais la vitesse d'exécution est améliorée en raison des meilleures performances des circuits de mémoire vive, et plus spécialement grâce à la largeur du bus de la RAM : 16 bits au lieu de 8 pour le bus de la ROM. On peut dès lors échanger davantage d'informations (donc d'instructions) par cycle de processeur. Tout ceci est possible à condition que les composants du PC permettent la "Shadow RAM" et qu'il y ait effectivement de la mémoire derrière le BIOS en ROM. C'est notamment le cas avec les puces NEAT de Chips&Technology, qui peuvent être configurées spécialement pour ce mode d'utilisation.

Paramètres de disque dur

Avec les paramètres de disque dur, nous touchons au délicat sujet de la collaboration du BIOS avec les innombrables disques durs offerts sur le marché. Le problème se situe

au niveau des différentes caractéristiques techniques des disques durs. Pour que le BIOS puisse exploiter un disque dur correctement, il doit connaître le nombre de têtes, de pistes, de secteurs par pistes, etc.

Dans le passé, ce problème a été traité à l'aide de longues tables stockées dans la ROM du BIOS. Leur liste était présentée à l'utilisateur par le Setup pour qu'il sélectionne le modèle monté dans sa machine.

Cette méthode était à peu près utilisable tant que l'on pouvait couvrir le marché des disques durs au prix d'un travail raisonnable, ce qui n'est plus le cas. Les programmes de Setup récents demandent la saisie manuelle des paramètres du disque dur. Le problème du stockage de ces données, naturellement impossible dans la ROM, a été résolu en faisant appel aux adresses utilisées par l'horloge du système.

Le programme Setup

Ici aussi, les concurrents se surpassent par le confort et le nombre des paramètres modifiables de leurs programmes Setup. C'est ainsi que de nombreux systèmes de BIOS permettent, outre les paramètres de base tels que la date, l'heure, les types de lecteurs, etc, de configurer une partie de la RAM comme mémoire EMS, ce qui exige toutefois la présence d'un matériel spécifique.

Sur les portables, il est possible de paramétrer le délai d'inactivité avant extinction de l'écran et l'arrêt du disque dur. Mais d'autres mesures d'économies d'énergie sont offertes par de nombreux fabricants, dans la limite des possibilités offertes par leurs circuits.

Certains programmes de Setup vont jusqu'à permettre une modification de la cadence du processeur à l'aide des touches <+> et <-> du clavier numérique, généralement accompagnées de <Alt>.

Mot de passe

Le BIOS en ROM est sans l'ombre d'un doute le meilleur emplacement possible pour stocker une demande de mot de passe sur PC. En effet, le BIOS est lancé longtemps avant le noyau du DOS, au tout début du l'initialisation de la machine. De nombreux fabricants de BIOS permettent de définir dans le Setup un mot de passe stocké ensuite dans les zones mémoire de l'horloge à temps réel ou sur une zone cachée du disque dur.

3.2. Comme une lettre à la POST

L'exécution du programme suivant la mise sous tension commence sur tous les ordinateurs de la famille 8088 et suivants à l'adresse F000H:FFF0H. Cette partie de la mémoire BIOS contient généralement une instruction de saut à une routine du BIOS qui teste le système et procède à l'initialisation des composants. Cette procédure est dénommée POST : "Power-On Self Test", ou "auto test à la mise sous tension" pour les purs francophones.

Les tests

Le POST procède à de nombreux tests des organes vitaux du PC (processeur, mémoire, contrôleur d'interruptions, DMA, etc). De plus, ils sont accompagnés de l'initialisation des extensions et cartes diverses pour permettre par exemple à la carte vidéo de prendre son service. Toute erreur détectée pendant ces tests est généralement accompagnée d'un bip et d'un message d'erreur et/ou code d'erreur.

Voici la liste des tests dans l'ordre de leur exécution. Sachez que cet ordre n'est pas strictement respecté par les fabricants de BIOS.

Tests POST effectués sur le matériel central :

- ✓ Test de fonctionnement de la CPU (arithmétique, mode réel, mode protégé, etc)
- ✓ Calcul d'une somme de contrôle de la ROM du BIOS (checkSum)
- ✓ Calcul d'une somme de contrôle de la RAM CMOS de l'horloge alimentée par accus et comparaison avec la somme de contrôle stockée dans la mémoire.
- ✓ Test et initialisation du contrôleur DMA
- ✓ Test et initialisation du contrôleur de clavier
- ✓ Test des premiers 64 Ko de la mémoire RAM
- ✓ Test et initialisation du contrôleur d'interruptions
- ✓ Test et initialisation du contrôleur d'antémémoire (AT seulement)

Ensuite arrivent les tests des fonctions du processeur, de ses registres et de quelques instructions. Toute erreur détectée à ce stade provoque l'arrêt du système sans message (en raison du processeur défectueux). Si le processeur sort blanchi de ce test, une somme de contrôle de la ROM du BIOS est calculée pour traquer toute erreur éventuelle. Vient le tour des circuits de la platine mère (contrôleurs d'interruption 8259 et 8237 de DMA, circuits de mémoire), qui sont soumis à des tests fonctionnels avant d'être initialisés.

Tests POST des extensions

- ✓ Contrôleur vidéo
- ✓ Mémoire centrale au-dessus des 64 Ko
- ✓ Interfaces série et parallèles

✓ Contrôleur de lecteurs de disquettes et de disque dur

Le fonctionnement correct de la platine une fois établi, les périphériques (clavier, lecteur(s) de disquettes, etc) sont également auscultés. Mais ces vérifications du matériel ne suffisent pas, il faut également initialiser les variables du BIOS et la table des vecteurs d'interruptions.

Recherche des extensions de la ROM

Ces tests sont suivis par la recherche d'extensions de la ROM pouvant se trouver sur la carte mère ou sur une carte d'extension. Leur rôle consiste à ajouter de nouvelles fonctions au BIOS ou à remplacer les fonctions existantes. C'est le cas des cartes EGA et VGA dont le BIOS en ROM remplace l'interruption 10h, destinée aux cartes CGA et MDA. Autre exemple, les contrôleurs SCSI de disques durs qui redirigent l'interruption BIOS 13h car elle ne permet pas l'utilisation de ce type de contrôleur.

Ces extensions du BIOS se reconnaissent aux deux premiers octets de la zone de mémoire qu'elles occupent : 55h et AAh. L'octet suivant est la taille du module ROM divisée par 512. Le troisième octet est le début de la routine d'initialisation appelée par le BIOS après qu'il ait identifié l'extension BIOS.

Identification d'un module ROM		
Offset	**Contenu**	**Type**
00h	1er octet identificateur (55h)	1 octet
01h	2ème octet identificateur (AAh)	1 octet
02h	Longueur du module en blocs de 512 octets	1 octet
03h	Routine d'initialisation	

Le module ROM peut de la sorte aiguiller des vecteurs d'interruption vers ses propres routines et s'insérer ainsi dans le système et remplacer les fonctions du système par les siennes.

Le BIOS ne recherche pas les extensions ROM dans toutes les adresses du PC, il se limite aux zones prévues pour ces extensions.

Zone	Extensions possibles
C000h:0000h - C000h:7FFFh	Les extensions BIOS des cartes EGA et VGA sont généralement stockées dans cette zone. Le BIOS lit cette zone par pas de deux Ko parce que plusieurs extensions peuvent la partager.

Zone	Extensions possibles
C000h:8000h -D000h:FFFFh	Aucune extension BIOS particulière n'est affectée à cette zone, mais elle est fréquemment utilisée par les contrôleurs de disques durs. Permettant également le stockage de plusieurs extensions, elle est parcourue par pas de deux Ko. Une partie, le segment D (D000h:0000h -D000h:FFFFh), est souvent utilisée par le "Page Frame" des cartes EMS. Elle est alors indisponible pour les extensions de ROM.
E000h:0000h - E000h:FFFFh	Les BIOS manquant de place dans le segment F réservent cette zone qui doit être entièrement utilisée par l'extension car le BIOS n'autorise qu'une extension dans cette zone.

Après le POST

L'initialisation des modules ROM marque la fin du lancement pour ce qui concerne le BIOS. Aucun système d'exploitation n'a encore été chargé dans la mémoire de l'ordinateur car il est stocké sur disquette ou sur le disque dur. L'interruption 19h, la "bootstrap loader" bien connue car elle est activée par <Ctrl><Alt><Del>, se charge de cette tâche.

Cette routine tente de trouver à un emplacement bien défini de la disquette une sorte de système d'exploitation primitif qui charge et lance le système d'exploitation proprement dit. Si cette tentative échoue, par exemple parce que la disquette ne contient pas de système d'exploitation ou parce que le lecteur est vide, une tentative est faite, selon le BIOS, sur l'autre lecteur actif ou sur le disque dur. En cas de nouvel échec, de nombreux systèmes chargent le BASIC en ROM, un petit interpréteur intégré dans la ROM directement après le BIOS, aux adresses F000h:6000h. Les modèles récents ont supprimé cette relique remontant à l'aube de l'ère du PC, ils affichent un message demandant à l'utilisateur d'insérer une disquette système dans le lecteur.

Notez bien que "disquette système" ne signifie pas disquette DOS ! Le BIOS ne saura jamais quel système d'exploitation vous chargez, ce qui permet de faire tourner UNIX, OS/2 ou Windows-NT.

3.3. Version de BIOS et type de PC

Outre le code du BIOS et quelques variables statiques (par exemple la table des paramètres des disques durs), le BIOS contient certaines informations sur le fabricant du PC, la version du BIOS et le type du PC. Rien de plus simple que de lire ces informations... si l'on sait où elles se trouvent.

Nous avons mentionné, dans la section précédente, la zone de mémoire F000h:FFF0h au sujet du lancement du système et du POST. Cette zone abrite la plupart du temps une instruction de saut longue de cinq octets, qui appelle la routine POST. Cette instruction est suivie de onze octets jusqu'à la fin du circuit de ROM, généralement utilisés pour stocker la version du BIOS ou la date de son agrément.

Si cela vous intéresse, vous pouvez consulter ces données très simplement. Appelez le programme DEBUG fourni avec le DOS : debug

Tapez la ligne suivante pour demander l'affichage de huit octets à la fin du BIOS en ROM :

```
d f000:fff0 l 8
```

DEBUG affiche sur la ligne suivante le contenu de cette zone, en hexadécimal et en ASCII. L'illustration suivante présente le codage sur deux chiffres du jour, du mois et de l'année. N'oubliez pas qu'aux Etats-Unis le mois est écrit avant le jour.

```
C>debug
-d f000:fff0 l 10
F000:FFF0  EA D4 04 A1 02 30 31 2F-31 35 2F 38 36 FF FC 00 ..... 01/15/88 ...
-q
C>
```

Connaître le type du PC

Les utilisateurs de certaines fonctions du BIOS apprécieront davantage de connaître l'identification du modèle que la version du BIOS. Elle indique le type du PC et elle est codée sur un octet à l'adresse F000h:FFFEh.

Codes de l'identification d'un modèle

Code	Signification
FCh	AT
FEh FBh	XT
FFh	simple PC

Il faut savoir que cette information n'est pas totalement fiable car les fabricants du BIOS préparent souvent une soupe à leur façon. Il est préférable de demander le type du processeur comme il est décrit dans la section 14.2.

3.4. Les variables du BIOS

Dans son principe, le BIOS est un programme figé dans un circuit ROM. Rien d'étonnant à ce qu'il aie besoin de variables, comme tout programme qui se respecte.

Le point intéressant est que ces variables ne peuvent être copiées dans la ROM, il faut donc leur réserver une zone de la RAM. Cette zone commence à 0040h:0000h et s'étend sur un peu plus de 256 octets. Elle est parfois nommée "segment des variables BIOS" ou "bloc des variables BIOS".

L'utilisation de cette zone est normalisée en détail parce que de nombreux programmes DOS lisent le contenu de ces variables, ce qui en pratique contraint les fabricants de BIOS à suivre les règles du jeu.

Cette zone de la mémoire est accessible comme n'importe quelle autre zone à condition de créer un pointeur FAR sur la variable désirée et de le déréférencer.

Mais... cette normalisation n'est réelle que pour les variables datant des PC et XT. Elles sont en tête du segment BIOS et couvrent les adresses jusqu'à 0071h. La suite a été ajoutée avec les AT, les cartes EGA et VGA et les systèmes PS/2. Cette partie est gérée différemment par les fabricants de BIOS, de PC et de cartes graphiques.

Quand les variables sont définies dans une seule configuration, celle-ci est stockée dans la description suivante des variables BIOS. Les pages suivantes vous apprendront les noms et significations des variables et naturellement leur adresse par rapport au segment 0040h:0000h et le format de données utilisé. Enfin, vous trouverez l'interruption du BIOS qui recourt à chaque variable.

00h	**4 MOTS INT 14h**
Adresses des interfaces série	

Pendant le POST le BIOS donne la configuration du PC et indique entre autres le nombre d'interfaces séries installées. Il note les adresses des ports de ces cartes dans un tableau de quatre mots dont chacun est l'adresse de base d'une des interfaces dont le nombre ne peut excéder quatre. L'adresse des ports introuvables est notée 0.

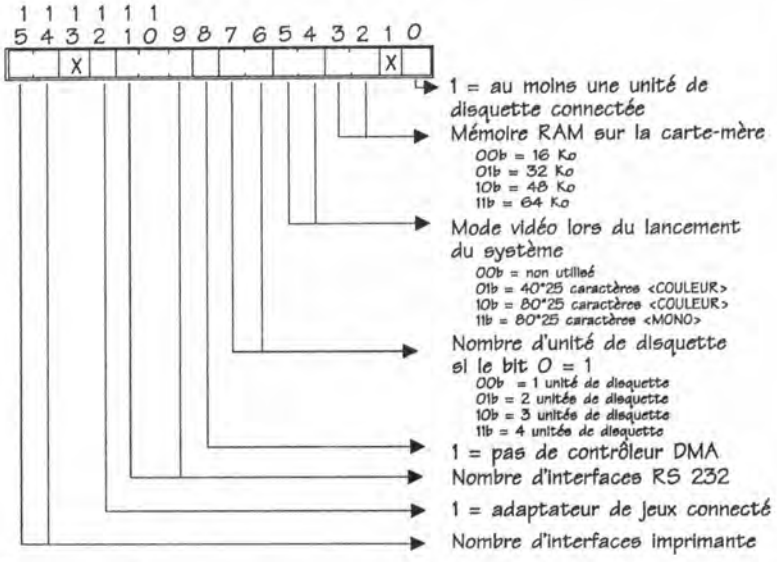
08h **4 MOTS INT 17h**
Adresses des interfaces parallèles

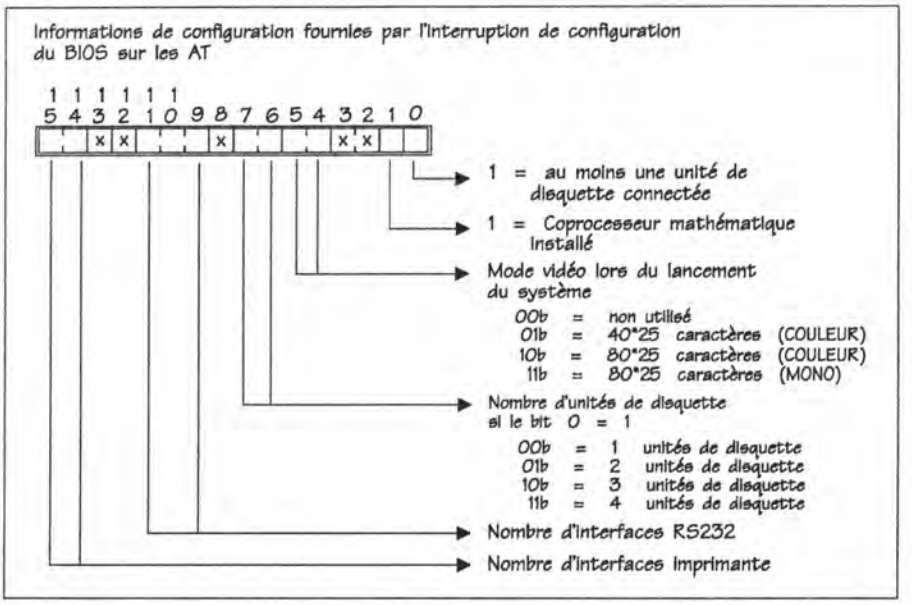
De manière analogue à la variable précédente, le BIOS copie ici les adresses de base des quatre interfaces parallèles. Les interfaces absentes reçoivent l'adresse 0.

10h **1 MOT INT 11h**
Configuration

Ce terme représente l'équipement matériel du PC, son mode d'interrogation par l'interruption 11h du BIOS et de passage par le BIOS pendant le POST. Ces variables sont gérées comme un champ de bits dont la signification varie entre les PC et XT d'une part et les AT de l'autre.

Informations de configuration fournies par l'interruption de configuration du BIOS sur les PC et XT





12h **1 OCTET POST**
Etat POST numéro 1

Cet octet sert à sauvegarder des informations pendant l'auto-test du système effectué pendant le POST et après un redémarrage à chaud. Il est utilisé ensuite par les routines du BIOS pour l'identification des touches frappées. Il n'offre pas d'intérêt pratique pour le développeur.

13h **1 MOT INT 12h**
Taille de la mémoire RAM

Le mot de cette adresse donne la taille de la mémoire totale exprimée en Ko. La mémoire EMS n'est pas prise en compte car elle ne fait pas partie de la mémoire centrale normale. Cette variable peut être interrogée par l'interruption BIOS 12h.

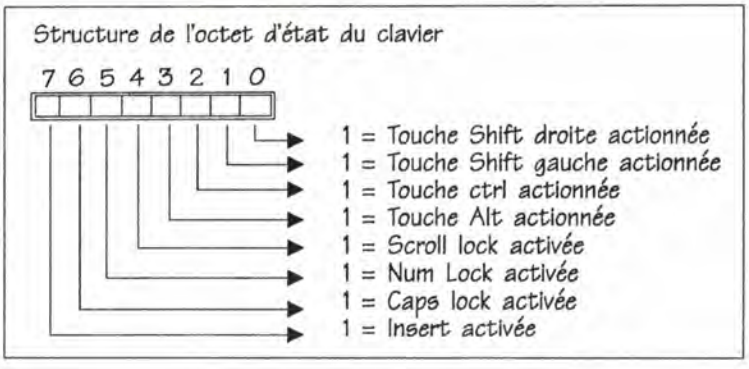
15h **1 MOT POST**
Etat POST numéro 2

Ce mot est lui aussi utilisé exclusivement pendant l'initialisation de l'ordinateur. Il est utilisé différemment selon le fabricant du BIOS.

17h **1 OCTET INT 16h**
Etat clavier numéro 1

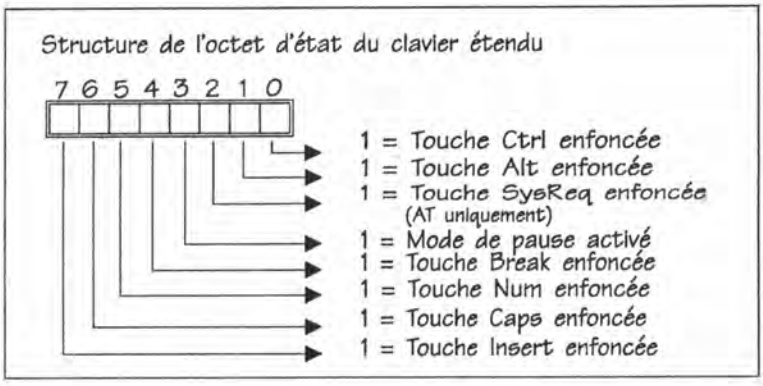
Cet octet est désigné sous le nom d'octet d'état du clavier parce qu'il contient l'état du clavier et de plusieurs touches. Vous pouvez l'interroger à l'aide de la fonction 02h de l'interruption de clavier 16h du BIOS.

En manipulant cet octet, vous pouvez activer ou inactiver les touches <Ins> et <Shift>. Seuls les quatre bits de poids fort sont intéressants car les quatre autres renseignent sur l'état des mêmes touches sans le modifier.



18h **1 OCTET INT 16h**
Etat de clavier étendu

La signification de cet octet est proche de celle du précédent, mais il est appelé "octet d'état de clavier étendu". En effet, cet octet n'est utilisé que depuis l'apparition des claviers étendus. Contrairement au premier octet d'état du clavier, celui-ci renseigne sur les touches commandant les changements de mode et non sur les changements eux-mêmes. Le bit 3 est l'exception, il indique le mode pause.



19h **1 OCTET INT 16h**
Saisie de code ASCII

Le code ASCII saisi sur le pavé numérique (avec la touche <Alt> enfoncée) est stocké dans cet octet.

1Ah **1 MOT INT 16h**
Caractère suivant dans le tampon du clavier

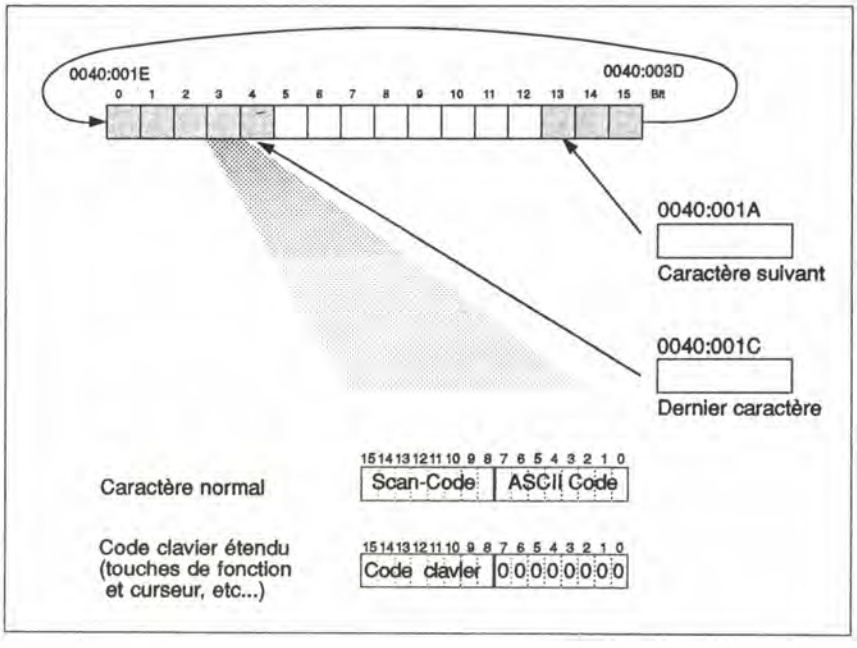
Ce mot contient l'adresse du caractère suivant dans le tampon du clavier (voir également 1Eh).

1Ch **1 MOT INT 16h**
Dernier caractère du tampon de clavier

Cette variable contient l'adresse du dernier caractère actuellement stocké dans le tampon du clavier (voir également 1Eh).

1Eh **16 MOTS INT 16h**
Tampon du clavier

Cette zone de mémoire contient le tampon BIOS du clavier. Chaque caractère occupant 2 octets, cette zone de 32 octets peut contenir 16 caractères. Les caractères ASCII sont stockés sous forme de leur code ASCII suivi de leur code clavier ("Scan code"). Les codes de clavier étendus ont une valeur ASCII égale à 0 car le code du caractère se trouve en fait dans l'octet suivant. Ce tampon est conçu de manière circulaire : les nouveaux caractères lus remplacent les anciens. La variable à l'adresse 1Ah contient l'adresse du caractère suivant . Quand un caractère est lu, ce pointeur est décalé de deux octets vers la fin du tampon. Quand le pointeur arrive au dernier octet du tampon, il revient au début, fermant le cercle.



Conception du tampon du clavier avec le pointeur de début et de fin

Il en va de même quand le pointeur est positionné sur l'adresse 1Ch, qui est la fin du tampon du clavier. Si l'utilisateur frappe une touche, son code clavier est stocké à l'adresse désignée par le pointeur. Il est ensuite décalé de deux octets vers la fin du tampon. Si un nouveau caractère a été stocké dans le dernier mot du tampon du clavier, ce pointeur est ramené au début du tampon.

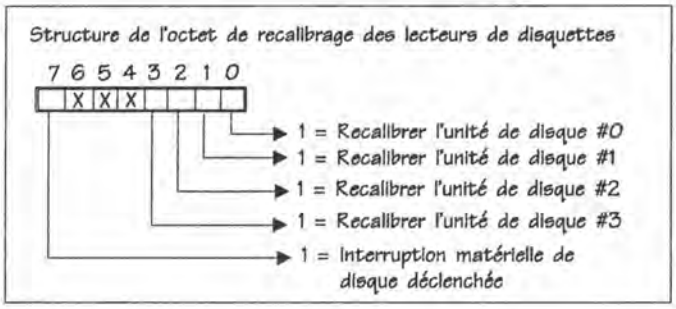
Les positions respectives du pointeur de début et de celui de fin définissent l'état du tampon, et deux états sont remarquables. Dans le premier, les deux pointeurs se trouvent

réunis en quittant les variables aux adresses 1Ah et 1Ch. Conclusion : le tampon est vide.

L'autre état remarquable se présente quand, au moment d'ajouter un caractère à celui qui était le dernier du tampon, l'incrémentation du pointeur de fin l'envoie sur le pointeur de début. Cela signifie que le tampon est plein et ne peut plus recevoir de caractère.

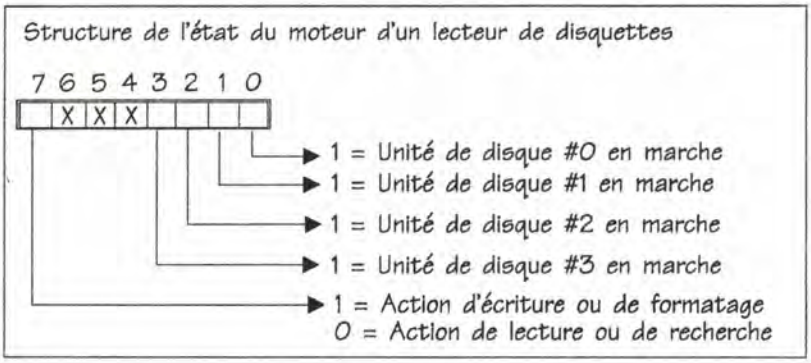
3Eh **1 OCTET INT 13h**
Recalibrage des lecteurs de disquettes

Les quatre bits de poids faible de cette variable correspondent aux quatre lecteurs de disquettes théoriquement autorisés par le BIOS. Ils indiquent que lecteur doit être recalibré en raison d'une erreur en lecture ou en écriture. De plus, le bit 7 est mis à 1 si un lecteur de disquettes a déclenché l'interruption matérielle de disquettes.



3Fh **INT 13h**
Etat du moteur d'un lecteur de disquettes

Ici aussi, les quatre bits de poids faible désignent les quatre lecteurs de disquettes. Ils indiquent si leur moteur est en marche. Le bit 7 est à un pendant les accès en écriture (et de formatage) et il est effacé pendant les accès en lecture et les recherches.



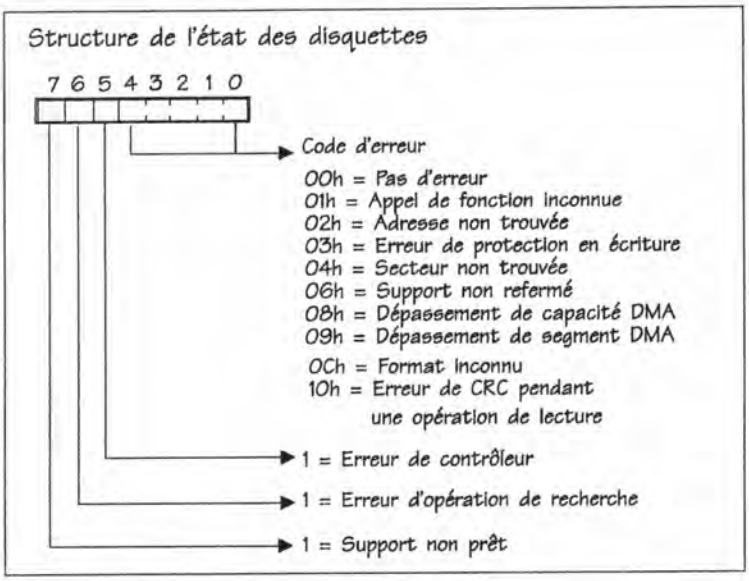
40h **1 OCTET INT 13h**
Chronomètre, lecteurs de disquettes

Cet octet contient un compteur indiquant le nombre d'appels de timer (interruption 08h) autorisés avant l'arrêt du moteur du lecteur. Le BIOS ne pouvant accéder qu'à un seul lecteur à la fois, ce compteur concerne le lecteur qui a fait l'objet du dernier accès.

Le BIOS écrit la valeur 37 immédiatement après la fin d'un accès à un lecteur. Le délai est d'environ 2 secondes.Si un nouvel accès a lieu dans ce délai, le compteur est remis à sa valeur initiale.

41h **1 OCTET INT 13h**
Etat des disquettes

Cet octet contient l'état du dernier accès à la disquette. Il vaut 0 si l'accès s'est déroulé normalement, une autre valeur est le code d'erreur envoyé par le contrôleur.



42h **7 OCTETS INT 13h**
Etat du contrôleur de disquettes

Ces sept octets donnent l'état du contrôleur de disquettes. Sur les systèmes avec disque dur, ils stockent également l'état du contrôleur de disque dur.

49h **1 OCTET INT 10h**
Mode graphique courant

Le BIOS copie le mode graphique utilisé dans cet octet. Cette valeur est identique à celle qui doit être donnée au moment de l'activation d'un mode d'écran à l'aide de la fonction 00h de l'interruption graphique 10h du BIOS.

4Ah **1 MOT INT 10h**
Nombre des colonnes de l'écran

Le mot de cette adresse contient le nombre de colonnes par ligne d'écran de ce mode graphique.

4Ch **1 MOT INT 10h**
Taille de la page écran

Ce mot indique au BIOS le nombre d'octets requis pour afficher une page écran dans le mode graphique utilisé. En mode texte 80 * 25 par exemple, cela donne 4000 octets.

4Eh **1 MOT INT 10h**
Adresse de la page écran affichée

Ce mot contient l'adresse de la page écran affichée, comptée à partir du début de la mémoire vidéo.

50h **8 MOT INT 10h**
Position du curseur dans les huit pages d'écran

Le BIOS peut gérer jusqu'à huit pages d'écran. Il stocke la position du curseur sur chacune d'entre elles dans ces huit mots. L'octet de poids faible de chaque page est la colonne et l'octet suivant est la ligne de l'écran.

60h **1 OCTET INT 10h**
Première ligne du curseur de l'écran

La première ligne du curseur (0 à 7 avec une carte couleur, 0 à 14 avec une carte monochrome) est stockée dans cet octet. Notez qu'une modification "manuelle" de cette valeur n'a aucun effet sur le curseur parce que cette valeur doit être passée par le BIOS au contrôleur graphique.

61h **1 OCTET INT 10h**
Dernière ligne du curseur de l'écran

La dernière ligne du curseur (0 à 7 avec une carte couleur, 0 à 14 avec une carte monochrome) est stockée dans cet octet.

62h **1 OCTET INT 10h**
Numéro de la page écran affichée

Le numéro de la page écran affichée est stockée dans cet octet.

63h **1 MOT INT 10h**
Adresse du port du contrôleur graphique

Cet octet contient l'adresse de base du port de la carte vidéo. Si un PC est équipé de plusieurs cartes graphiques, l'adresse contenue est celle de la carte active. Cette adresse est 3B4h pour les cartes monochromes, ou 3D4h pour les cartes CGA, EGA et VGA. La valeur de ce mot indique donc si la carte active est monochrome ou couleur.

65h **1 OCTET INT 10h**
Contenu du registre de sélection de mode

Nous trouvons ici le contenu du registre de sélection de mode d'une carte graphique. Il définit le mode graphique en cours d'utilisation.

66h **1 OCTET INT 10h**
Contenu du registre de palette

Les cartes graphiques en couleurs possèdent des registres de palette leur permettant de sélectionner une palette de couleurs dans les modes compatibles CGA. La dernière valeur écrite dans l'un de ces registres par le BIOS est stockée dans cette variable.

67h **5 OCTETS POST**
Divers

Les versions des BIOS des premiers PC, ceux qui permettaient encore de brancher un magnétophone à cassette pour stocker les données, utilisaient ces 5 octets pour gérer ce mode de stockage. Les XT et AT, qui ont perdu cette faculté, utilisent cette adresse à d'autres fins.

6Ch	1 DOUBLE-MOT INT 1Ah
Chronomètre	

Ces quatre octets sont utilisés par le BIOS comme un chronomètre à 32 bits, incrémenté à chaque appel du timer. La valeur de ce chronomètre peut être lue ou écrite à l'aide de l'interruption 1Ah du BIOS. Elle est remise à 0 toutes les vingt-quatre heures et reprend ensuite sa progression.

70h	1 OCTET INT 1Ah
Flag de 24 heures	

Cet octet est incrémenté par l'interruption Timer quand un jour s'est écoulé et le chronomètre est remis à zéro. Cet octet permet de savoir connaître le nombre de jours de fonctionnement continu ou depuis le redémarrage précédent. Il est remis à zéro si l'heure ou la valeur du chronomètre sont modifiées à l'aide de l'interruption 1Ah.

71h	1 OCTET INT 16h
Flag Control Break	

Le bit numéro sept de cet octet est mis à un quand le BIOS rencontre la combinaison de touches <Ctrl><Brk>. Cette valeur n'est pas modifiée par la suite.

Nous sommes ainsi parvenus à la fin des variables BIOS du PC normal. Les extensions matérielles de l'XT et de l'AT utilisent quelques variables supplémentaires nouvellement introduites.

72h	1 MOT POST
Test POST	

Pendant le POST, le BIOS détecte à l'aide de ce drapeau s'il s'agit d'un démarrage à froid ou à chaud (<Ctrl><Alt><Del>). La routine spécialisée du BIOS écrit 1234h à cette adresse avant un démarrage à chaud, ce qui provoque l'omission du test de la mémoire.

74h — **1 OCTET INT 13h**
Etat de la dernière opération sur disque dur — *AT seulement*

Cet octet contient l'état de la dernière opération sur disque dur effectuée par le BIOS.

01h:	Fonction ou lecteur demandé inconnu
02h:	Marquage d'adresse introuvable
04h:	Secteur introuvable
05h:	Erreur de remise à zéro du contrôleur
07h:	Erreur d'initialisation du contrôleur
09h:	Erreur de transmission DMA : débordement de limite d'un segment
0Ah:	Secteur défectueux
0Bh:	Piste défectueuse
0Dh:	Nombre de secteurs invalide dans une piste
0Eh:	Marquage d'adresse introuvable
0Fh:	Débordement DMA
10h:	Erreur en lecture
11h:	Erreur en lecture corrigée par ECC
20h:	Contrôleur défectueux
40h:	Opération de recherche a échoué
80h:	Le lecteur ne répond pas (time out)
AAh:	Le lecteur n'est pas prêt
CCh:	Erreur en écriture

75h — **1 OCTET INT 13h**
Nombre de disques durs — *AT seulement*

Stockage du nombre de disques durs.

76h	1 OCTET INT 13h
Octet de contrôle du disque dur	*AT seulement*

Cet octet est utilisé par l'interruption 13h pour la commande du disque dur. Sa signification exacte est inconnue.

77h	1 OCTET INT 17h
Port de disque dur	*AT seulement*

L'adresse de base du contrôleur de disque dur est stockée ici.

78h	4 OCTETS INT 14h
Compteur de time out, interface parallèle	

Cette variable est un tableau de quatre octets dont chacun correspond à une interface parallèle et stocke son nombre de time outs. Ce compteur définit le délai limite donné au correspondant pour répondre. Si aucune réponse n'a été reçue à l'expiration du délai, le BIOS interrompt la communication.

7Ch	4 OCTETS INT 16h
Compteur de time out, interface série	

Ce tableau est chargé de la même tâche que celui de l'adresse 78h, mais il traite l'interface série.

80h	1 MOT INT 16h
Adresse du tampon de clavier	*AT seulement*

Ce mot contient l'adresse du tampon de clavier exprimée à partir du segment des variables BIOS 0040h. Le but visé à l'origine était de permettre une extension du tampon du clavier en le décalant. L'idée n'a pas été concrétisée car de trop nombreux programmes lisent directement le tampon à l'adresse 001Eh sans demande préalable de l'adresse, ce qui interdit le décalage du tampon en mémoire

82h	1 MOT INT 10h
Fin du tampon de clavier	*AT seulement*

Cette variable est l'inverse de la précédente, elle donne la fin du tampon du clavier dans le segment des variables BIOS. Elle est victime du même sort que son homologue et reste inutilisable pour la même raison.

84h	1 MOT INT 10h
Nombre de lignes écran	*EGA/VGA seulement*

Le BIOS des cartes EGA et VGA, et lui seul, stocke le nombre de lignes en mode texte dans cette variable.

85h	1 MOT INT 10h
Hauteur des caractères en points	*EGA/VGA seulement*

La hauteur des caractères exprimée en lignes de points du mode graphique courant est stockée ici si l'on utilise une carte EGA ou VGA. On peut connaître le nombre de lignes de texte affichées en divisant le nombre de lignes de points de l'écran par cette valeur.

87h	4 OCTETS INT 10h
Zone d'état EGA/VGA	*EGA/VGA seulement*

Cette zone est mise à la libre disposition des cartes EGA et VGA. Elles en font des utilisations très variées.

8Bh	11 OCTETS INT 13h
Paramètres des disquettes et disques durs PS/2	*PS/2 seulement*

Les systèmes PS/2 stockent dans cette zone diverses informations relatives aux accès disquettes et disque dur.

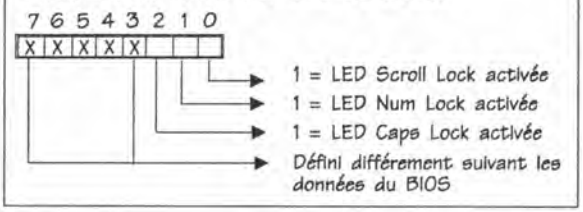
96h	**1 OCTET INT 16h**
Etat clavier étendu	*AT seulement*

Cet octet est utilisé sur les systèmes à clavier étendu comme un champ de bits dont les éléments stockent certains évènements relatifs à la gestion interne du clavier par le BIOS. Le bit intéressant pour le développeur est le numéro quatre qui est mis à un si un clavier étendu à 101 touches (version américaine) ou 102 touches est connecté. Un programme utilisant les touches étendues du clavier étendu (par exemple <F11> ou <F12>) peut savoir si elles sont disponibles en lisant la valeur de ce bit.

97h	**1 OCTET INT 16h**
Etat des LED	*AT seulement*

Avec un clavier étendu, cet octet donne l'état des LED correspondant à trois modes de connexion. On peut allumer ou éteindre les LED en modifiant "manuellement" ces bits. Toutefois, il faut après basculement, appeler l'une des fonctions de l'interruption clavier 16h pour que le BIOS en ROM interroge cet octet et allume ou éteigne la LED. La fonction la plus qualifiée pour cette tâche est la 02h qui renvoie l'état courant du clavier sans interroger un seul caractère du clavier.

Structure de l'état des LED du clavier



98h	**1 DOUBLE-MOT INT 15h**
Pointeur sur le flag "Wait"	*AT Seulement*

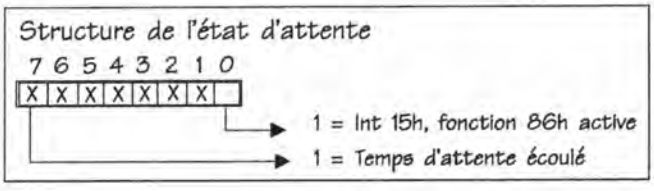
Ce pointeur FAR est lié à la fonction 83h de l'interruption BIOS 15h. La fonction 83h permet de définir une variable BYTE dont le bit numéro 7 sera mis à un après écoulement d'un délai défini. L'adresse de cette variable est stockée ici, dans le segment des variables BIOS.

9Ch	1 DOUNLE-MOT INT 15h
Chronomètre	*AT seulement*

Cette variable est également liée à la fonction 83h de l'interruption BIOS 15h. Elle sert également à la fonction 86h. Ces deux fonctions ont besoin d'une variable dans laquelle copier le délai donné par la fonction qui les a appelées.

A0h	1 OCTET INT 15h
Etat d'attente	*AT seulement*

Voici la dernière variable utilisée par la fonction 86h de l'interruption 15h. Elle indique si le délai est écoulé et si la fonction 86h est déjà active.



A1h	95 OCTETS
Réservé	

Cette zone est dite "réservée", ce qui en fait l'enfant chéri d'innombrables extensions du BIOS et de programmes.

100h	1 OCTET INT 05h
Flag de récursion de copie d'écran	

Tous les membres de la famille PC (PC, XT et AT) ont une variable en commun à l'adresse 0050:0000h. Elle est utilisée comme drapeau de récursion par la routine de copie d'écran (interruption 05h) pour interdire le lancement d'une deuxième copie d'écran pendant que la première est en cours d'exécution. Ce drapeau est mis à un au début de la routine. A la fin de la copie d'écran, le drapeau est mis à la valeur 255 si une erreur s'est produite pendant la sortie.

4. Les cartes vidéo

En tant que périphérique de sortie de premier ordre, l'écran assure la liaison entre l'utilisateur et les divers programmes utilisant ce qu'on appelle les cartes vidéo pour effectuer leurs sorties. La création de routines destinées à rendre les informations visibles sur l'écran constitue une activité essentielle dans le cadre du développement. Il s'agit généralement d'avoir recours à la carte vidéo vu que le BIOS en ROM contient toute une série de routines prédéfinies pour l'accès à l'écran. La plupart des programmeurs considèrent que ces routines sont trop lentes et préfèrent piloter l'électronique vidéo par leurs propres soins. Pourtant, ce n'est pas une tâche simple quand on pense qu'il existe actuellement six standards vidéo différents dans le monde PC et on ne sait jamais quel type de carte vidéo est implantée présentement dans un ordinateur.

Ce chapitre apporte de la lumière dans la confusion résultant des différents standards vidéo, décrit leurs caractéristiques, les routines de sortie écran BIOS et la programmation directe des cartes vidéo. La programmation des cartes EGA et VGA occupe une place primordiale car elles représentent actuellement le standard dans le domaine des cartes vidéo. En même temps que les cartes, nous serons amenés à évoquer tous les secrets de la programmation graphique ainsi que le thème de la programmation Sprite. Ce dernier intéresse particulièrement le monde des animations graphiques.

Les standards graphiques d'antan ne sont pas pour autant oubliés. Ainsi, nous rencontrerons les modes MDA, CGA et Hercules (HGC) ainsi que les cartes modernes Super VGA pour lesquelles il n'existe encore aucun standard, les cartes TIGA qui introduisent un nouveau chapitre et la programmation graphique. Pour commencer, il convient de raconter brièvement l'histoire et présenter les caractéristiques performantes des différents standards vidéo du monde PC.

4.1. Histoire et Performances

A côté de la vitesse de calcul toujours plus croissante des ordinateurs, les progrès dans la technologie électronique n'ont jamais été aussi marqués que dans le cadre des cartes vidéo. Ici, l'évolution s'effectue non seulement de plus en plus vite mais tend surtout à promouvoir des cartes vidéo extrêmement performantes. La résolution écran et le nombre de couleurs qu'elles reconnaissent sont tellement élevés qu'elles dépassent de loin leurs antécédents.

Jusqu'à présent, les fabricants utilisaient les technologies existantes pour y apporter des améliorations quantitatives fondamentales. La tendance actuelle se dirige au contraire dans une autre direction : les cartes graphiques se voient équipées de processeurs intelligents se chargeant de dessiner des lignes, cercles et surfaces à la place du processeur. Ainsi, il a davantage le temps de s'occuper de ses propres tâches, notamment la préparation des informations et l'interaction avec l'utilisateur. Cette technologie prend justement une signification particulière au seuil de l'ère des interfaces

graphiques parce que ces systèmes réclament au processeur de satisfaire des exigences considérables. Si les applications ne souhaitent pas donner une impression de lenteur à l'utilisateur, il faut alors que le processeur soit soulagé de certaines corvées. Les cartes graphiques intelligentes visent d'ailleurs ce but, comme vous le saurez dans ce chapitre avec la description du standard 8514/A et TIGA.

Grâce aux modifications intervenues au cours de l'histoire des cartes vidéo, les sections suivantes mettent l'accent sur l'évolution de ce secteur électronique et résument les caractéristiques performantes des différentes classes de cartes vidéo. Vous aurez également un aperçu sur les perspectives d'avenir puisque le développement dans le domaine des cartes vidéo n'est pas encore arrivé à son stade final.

MDA

L'appellation "IBM Monochrome Display Adapter", MDA en abrégé, représente avec la carte CGA le plus ancien des adapteurs graphiques disponible sur le PC. Il a été officiellement présenté en 1981 avec le premier ordinateur personnel IBM et tenait le titre de standard dans le domaine des cartes vidéo monochromes au cours de plusieurs années.

La carte MDA ne reconnaît qu'un seul mode d'exploitation où les caractères apparaissent sur 25 lignes et 80 colonnes sur l'écran. Contrairement aux nombreuses cartes graphiques, elle ne dispose que d'une quantité peu importante de RAM vidéo si bien que la mémoire ne peut stocker qu'une seule page écran.

Il va sans dire que cette carte ne permet pas de créer des graphiques. Malgré tout, bon nombre d'utilisateurs préféraient la carte MDA à la carte CGA qui représentait la seule et unique alternative à cette époque. En comparaison à une carte CGA, une carte MDA garantit une résolution écran nettement plus précise. Ces cartes fatiguent beaucoup moins les yeux de l'utilisateur par rapport aux cartes CGA.

Aujourd'hui, les cartes MDA ne sont presque plus utilisées d'autant plus qu'elles ne sont plus fabriquées depuis déjà longtemps. Dans le domaine des cartes monochromes, leur place a été prise par les cartes Hercules. Ces dernières reconnaissent non seulement tous les attributs d'une carte MDA, mais peuvent de surcroît afficher des graphiques monochromes. Elles se présentent ainsi comme une véritable alternative aux divers types de cartes graphiques couleur.

CGA

Le standard CGA ou Color Graphics Adapter est également apparu en 1981. Se substituant aux cartes MDA, cette carte permettait déjà à l'utilisateur de créer des graphiques durant les premiers jours du PC.

Si vous avez vidé votre tire-lire pour acquérir une carte CGA, soyez rassuré que vous n'avez pas besoin d'envisager l'achat d'un moniteur puisque les cartes CGA disposent d'un port de sortie spécial adaptable sur un téléviseur normal. Elles sont équipées en outre d'une sortie RVB, en fait une ligne, sur laquelle la couleur d'un point écran vient se placer à côté des divers signaux de synchronisation selon ses proportions de rouge, vert et bleu. En comparaison avec les cartes MDA, la qualité de l'image obtenue est nettement inférieure en raison de la résolution moindre et de l'écart trop important entre les points dans le moniteur CGA.

Tout comme la carte MDA, la carte CGA affiche 25 lignes et 80 colonnes sur l'écran en mode texte. Mais ici, les différents caractères se basent sur une matrice de points plus petite que celle des cartes MDA. Ainsi, on peut afficher des graphiques avec une résolution de 320*200 points où le choix des couleurs est plutôt maigre avec seulement quatre couleurs. En résolution maximale, la configuration de l'écran s'effectue avec 640*200 points mais avec deux couleurs seulement.

Bien que les points performants d'une carte CGA et d'une carte MDA soient nettement différents, les deux cartes reposent néanmoins sur un contrôleur vidéo identique qui est le MC6845 de Motorola. Eu égard au sectionnement du processeur dans le PC, ce fabricant de Chip a peut-être supporté les frais à cause d'INTEL, mais il n'en reste pas moins qu'il est resté le maître absolu pendant plusieurs années.

Hercules Graphics Card

Un an après l'introduction du PC sur le marché, la société Hercules, encore complètement inconnue, fait son apparition avec une nouvelle carte graphique pour le PC et remporte un énorme succès. Utilisant également le contrôleur Motorola déjà évoqué, cette carte est entièrement compatible à la carte MDA d'IBM. Hercules a mis toutes ses facultés à l'épreuve pour cette carte puisqu'en dehors du mode texte, la carte HGC peut gérer deux pages graphiques avec une résolution de 720*348 points. En fait, elle associe la haute lisibilité d'une carte MDA aux capacités graphiques d'une carte CGA et satisfait en plus en raison de sa haute résolution.

Aujourd'hui encore, cette carte est considérée comme le standard dans le domaine des cartes graphiques monochromes alors que les cartes graphiques monochromes continuent de perdre leur valeur par rapport aux cartes graphiques couleurs. A l'heure où les cartes CGA et MDA ne sont pratiquement plus fabriquées, la carte graphique Hercules se rencontre encore dans les carnets de commande de nombreux fabricants. Les ordinateurs provenant de l'Extrême-Orient sont généralement équipés avec une telle carte qui est de moins en moins fabriquée par la société Hercules. Elle est par contre le fruit d'un fournisseur tiers.

Le modèle original ainsi que tous les modèles dérivés sont touchés par la non reconnaissance du BIOS, en effet IBM a toujours refusé de soutenir les cartes vidéo étrangères via son BIOS. Mais la carte graphique Hercules n'en souffre pas trop car elle

est compatible avec la carte MDA d'IBM et n'est réclamée que rarement à travers le BIOS en ce qui concerne le mode texte.

Quant au mode graphique de cette carte, le BIOS n'apporte aucune aide au programmeur dans ce contexte. Cela concerne à la fois l'initialisation du mode graphique et l'accès aux différents points écran. Il n'y a pas à s'en inquiéter puisque les routines BIOS sont généralement contournées en raison de leur vitesse laissant à désirer. Ainsi, l'accès aux informations sur les points dans une HGC ne peut être que géré très simplement.

Etant donné que la carte graphique Hercules connaît pour ainsi dire un succès permanent sur le marché du PC, la miniaturisation progressive des divers composants du PC s'est effectuée convenablement sur cette dernière. Alors que les premières cartes Hercules occupaient toute la longueur de construction et contenaient plus de 40 IC, les cartes Hercules modernes reposent sur une petite carte cachée et comptent généralement moins de dix IC contenant une interface imprimante parallèle en plus d'une carte graphique.

La société Hercules a développé d'autres cartes vidéo en dehors de la carte graphique Hercules. Ces dernières ont toutefois obtenu un succès moindre et ne jouent désormais aucun rôle sur le marché du PC. A titre d'information, il s'agit de Hercules Graphics Card Plus et Hercules InColor Card.

EGA

Après qu'une autre société, avec sa carte graphique Hercules, ait pénétré pour la première fois le marché détenu entièrement par IBM, on s'est alors efforcé chez IBM de développer un successeur de la carte CGA capable d'éliminer la carte graphique Hercules du marché. Le résultat de ces efforts a donné naissance à la carte Enhanced Graphics Adapter ou EGA en 1985.

Grâce aux énormes progrès réalisés dans le domaine électronique entre 1981 et 1985, la carte EGA allait largement au-delà des capacités des cartes CGA et MDA et générait une petite révolution dans ce secteur. En raison de son prix élevé, cette carte a mis quelque temps avant d'être accessible à la majorité des utilisateurs et devenir ainsi le modèle standard.

Outre ses modes vidéo individuels, la carte EGA est entièrement compatible avec les cartes CGA et MDA. De ce fait, elle peut également être utilisée dans des programmes qui ne reconnaissent que ce type de cartes vidéo. De surcroît, la carte EGA est capable, comme une carte graphique Hercules, de produire des graphiques monochromes sur un moniteur monochrome. Par conséquent, elle représente la première carte graphique sur le PC utilisable avec des écrans monochromes et couleur.

Une carte EGA n'étale toutefois toute sa puissance qu'en collaboration avec un moniteur EGA spécifique dont les performances n'ont rien à envier à un moniteur CGA. En mode

graphique élevé, si la résolution n'atteint que 640*350 points (non pour autant supérieure à celle d'une carte EGA), 16 couleurs différentes affichées simultanément provenant d'une palette de 64 couleurs comblent cette lacune. L'affichage d'un nombre de couleurs plus important s'accompagne de l'agrandissement de la RAM vidéo pouvant atteindre 256 Ko dans les cartes EGA.

Pour obtenir ce résultat, il fallait renoncer au contrôleur vidéo MC6845 qui ne s'adaptait plus à de telles exigences. La carte EGA utilise donc des Chips VLSI intégrés qui se chargent de remplir toutes les tâches liées à la création graphique. Ce standard comporte une différence fondamentale par rapport à la méthode connue jusque-là en ce qui concerne surtout la manière dont les informations graphiques sont stockées sur le plan interne dans la RAM vidéo. Du point de vue de la programmation des cartes vidéo, il franchit un pas important.

Grâce à la sélection d'un écart infime entre les points dans un moniteur EGA, la carte EGA excelle par rapport à la carte CGA en produisant une image nettement plus précise. Elle laisse loin derrière elle les modèles précédents à d'autres points de vue également. Par exemple, elle est la première carte vidéo à utiliser les polices variables parce que des jeux de caractères quelconques peuvent être chargés par le logiciel.

La carte EGA comporte également pour la première fois des techniques nécessaires dans les animations. Les jeux d'action ont pu être introduits dans le monde PC grâce à l'apparition de la carte EGA.

Comme les techniques performantes évoluées des cartes EGA ne sont pas reconnues à travers le BIOS en ROM contrairement aux cartes CGA et MDA, les cartes EGA sont entrées dans la course avec un BIOS personnel contenu directement dans la carte. Dans le domaine de la sortie vidéo, il remplace le BIOS d'origine et ouvre ainsi la voie à toutes les techniques performantes des cartes EGA.

Le développement croissant des cartes EGA a provoqué une folie chez les fabricants de cartes compatibles. Ils se sont mis à créer des cartes vidéo de plus en plus performantes avec des modes vidéo toujours plus surprenants qui finalement n'étaient pratiquement jamais reconnus par les programmes. Alors qu'au début, IBM intentait une action contre chaque fournisseur de carte compatible EGA, il a fallu bientôt se défendre contre l'afflux de cartes compatibles EGA en provenance d'Extrême-Orient. A la longue, on s'est trouvé devant la situation absurde où les fabricants de cartes EGA proposaient des cartes dont les caractéristiques techniques dépassaient largement le standard VGA conçu en fait comme une extension et comme le successeur du standard EGA.

Comme autrefois, les cartes EGA sont encore aujourd'hui très demandées mais ne remplissent plus le rôle de modèle standard car elles sont de plus en plus anéanties par les cartes VGA. Elles sont très compatibles à ces dernières parce que le standard VGA représente un modèle évolué du standard EGA à divers points de vue.

VGA

La carte VGA, présentée en 1987 avec les premiers systèmes PS/2 d'IBM, se conforme strictement à la carte EGA. Autrement dit : compatibilité avec tous les modèles précédents, plus de couleurs, résolution plus élevée et meilleur affichage du texte.

Ces caractéristiques, se rajoutant au prix fort avantageux des cartes VGA et des moniteurs, ont permis rapidement à la carte VGA de se hisser au rang du modèle standard dans le monde PC. Si vous ne souhaitez plus travailler uniquement en mode monochrome, il ne vous reste plus qu'à sélectionner la méthode VGA lors de l'achat d'un ordinateur.

Au départ, le standard VGA était exclusivement destiné aux systèmes IBM PS/2 et par voie de conséquence au nouveau mode Micro Channel. Mais la plupart des fournisseurs se sont mis rapidement à fournir des cartes VGA pour le bus ISA, si bien que des systèmes fort ordinaires peuvent être équipés de cartes VGA.

La différence entre les cartes EGA et VGA repose essentiellement sur la densité d'intégration qui est plus élevée dans les cartes VGA. En outre, les signaux de couleur émis par ces dernières au moniteur ne sont plus digitaux mais analogiques. Ainsi, les cartes VGA peuvent produire plus de 260000 variations de couleurs accessibles respectivement en mode 2, 4, 16 ou 256.

Dans les cartes VGA, la résolution maximale atteint 640*480 points avec 2, 4 ou 16 couleurs. Dans un mode étendu 320*200 points, cela prend des proportions considérables puisqu'il est possible d'afficher simultanément 256 couleurs. Il est clair qu'en raison de la résolution particulièrement élevée et de la multitude des couleurs, la quantité de RAM vidéo nécessaire pour recevoir les données de l'image ne peut être qu'importante. Généralement, les cartes VGA sont fournies avec une RAM vidéo de 256 Ko minimum pouvant facilement être augmentés de 512 Ko.

Tout comme les cartes EGA, les cartes VGA disposent d'un BIOS individuel remplaçant le BIOS standard dans le cadre de la sortie vidéo. De même, il est compatible avec le BIOS EGA. Ainsi, tous les programmes conçus pour le BIOS EGA peuvent fonctionner sans aucun problème sous le BIOS d'une carte VGA.

A cause de la concurrence féroce régnant sur le marché PC, il s'est créé une situation autour des cartes VGA d'ailleurs plus marquée par rapport aux cartes EGA, où les divers fabricants cherchent à développer des modes vidéo de plus en plus performants et à élargir autant que possible l'éventail des couleurs. Jusqu'à présent, chaque fabricant a développé sa propre formule obligeant le programmeur a adapté ses applications aux différentes cartes vidéo dont les modes étendus sont susceptibles d'être soutenus. Cela explique pourquoi la plupart des programmes reconnaissent, encore aujourd'hui, exclusivement les modes standard VGA malgré la grande diversité des cartes VGA étendues.

Le souhait de définir un standard VGA étendu laisse cependant espérer que l'accès aux modes VGA étendus soit standardisé et que ces derniers soient rendus accessibles à tous les programmes.

Super VGA

Du point de vue de l'électronique fondamentale, les cartes Super VGA ressemblent aux cartes VGA ordinaires mais travaillent plus vite. Dans le même laps de temps, elles affichent en effet un nombre de points plus important sur l'écran tout en atteignant une haute résolution. Elles reconnaissent de surcroît tous les modes VGA habituels et sont en mesure de reproduire simultanément beaucoup plus de couleurs que les cartes VGA ordinaires.

Contrairement aux cartes VGA ordinaires qui ne peuvent afficher 256 couleurs qu'en mode 320*200 points, les cartes Super VGA assurent cette diversité de couleurs dans les autres modes également (640*200, 640*350 ou 640*480 points). Elles octroient en outre l'utilisateur d'un mode graphique avec une résolution de 800*600 ou 1024*768 points. Ces résolutions nécessitent évidemment des moniteurs VGA ou Multiscan tout aussi performants sans oublier que leur consommation en RAM vidéo atteint des mesures gigantesques.

Comme précédemment, les fabricants n'ont adopté aucune homogénéité quant à l'initialisation et l'accès des modes étendus au niveau de l'électronique à travers les registres d'une carte de ce type. Les principaux fabricants de Chips compatibles VGA (Tseng, Paradise et Video Seven) ont formé un consortium appelé "Video Electronic Standard Association" (VESA). Ils ont défini également un standard pour l'accès aux modes Super VGA étendus à travers le BIOS. Ce dernier est destiné à être inclus dans le BIOS des cartes vidéo conçues à partir des Chips de ces fabricants. Les anciennes cartes Super VGA se verront dotées de programmes TSR ajoutant de nouvelles fonctions dans le BIOS actuel.

Malheureusement, cette corporation ne s'est créée qu'au début de l'année 1990 si bien qu'il faudra attendre quelque temps avant qu'elle ne soit largement répandue. Jusque là, tout programme souhaitant utiliser les modes VGA étendus doit accéder directement à l'électronique individuelle des diverses cartes Super VGA.

MCGA

Pendant qu'IBM dotait ses modèles PS/2 d'une carte VGA, les petits modèles ont été équipés d'une carte MCGA. Il s'agit là d'un produit de remplacement de pratiquement tous les standards antérieurs. Le nom MCGA signifie "Memory Controller Gate Array", une notion qui se rapproche des cartes vidéo.

En ce qui concerne le mode de texte, cette carte se comporte exactement comme une carte CGA affichant 80*20 caractères où les couleurs de premier plan et de fond proviennent d'une palette de 16 couleurs. Contrairement à une carte CGA, ces couleurs ne sont pas prédéfinies. Elles peuvent donc être sélectionnées parmi une palette de plus de 262000 couleurs tout comme avec une carte VGA. Dans ce mode, la résolution verticale s'élève à 400 lignes de points ce qui permet d'obtenir une qualité d'affichage particulièrement appréciable.

Par rapport aux divers modes graphiques, la carte MCGA agit également comme un hybride. Outre les deux modes VGA compatibles, elle reconnaît notamment les deux modes CGA avec 320*200 et 640*200 points. Mais comme la carte doit toujours fonctionner sur la base d'une résolution verticale de 400 points, les points de lignes sont dédoublés dans ce mode. Le résultat est qu'on obtient seulement la moitié de la résolution escomptée.

Cela ressemble plutôt aux modes VGA atteignant la résolution VGA connue habituellement, mais fait preuve d'une limitation dans le domaine des couleurs. Cette situation s'explique par le fait qu'une carte MCGA est équipée seulement de 64 Ko de RAM vidéo offrant naturellement peu de couleurs en haute résolution. Une telle circonstance rappelle l'image du propriétaire d'une limousine qui ne voulait pas dépenser trop d'argent pour acheter de l'essence.

En raison des limitations citées plus haut, les cartes MCGA n'ont pas pu franchir l'étape entre le Micro Channel et le vaste monde du PC et sont restées quelque peu en désuétude. Nous n'insisterons donc pas outre mesure sur la programmation des cartes MCGA dans le cadre de ce livre.

8514/A

Pour garder l'emprise sur les cartes vidéo, IBM annonça, en 1987, l'héritier du standard VGA. 8514/A correspond en quelque sorte à un nom codé cachant derrière lui une révolution dans le domaine des cartes vidéo. Le contrôleur vidéo qui faisait jusque-là piètre figure à côté du processeur, la carte vidéo vient s'équiper personnellement d'un processeur adressable par les commandes externes. L'avantage apparaît nettement : désormais, le processeur n'est plus obligé de calculer les différents points d'une ligne ou courbe puisqu'il peut les déléguer au processeur graphique. Parallèlement à une autre exécution de programme, ce dernier traite la ligne souhaitée ou un autre objet graphique et soulage ainsi le processeur d'une lourde tâche. Il en va ainsi de la théorie.

En pratique, la nouvelle trouvaille d'IBM est à peine capable d'assurer la relève puisque l'utilisateur a déjà investi dans le standard VGA. L'échec de ce système graphique incombe également à IBM qui a fait preuve d'un mauvais aiguillage.

D'une part, il était dit expressément que ce standard graphique s'adressait uniquement au Micro Channel qui n'a d'ailleurs pas pu s'outrepasser et se tailler une place dans le

futur. D'autre part, IBM a tenu strictement en secret les caractéristiques techniques de cette carte pour qu'un tiers ne puisse pas la copier. Une telle stratégie est justement à bannir dans le marché du PC où l'idée maîtresse est de divulguer toutes les informations et les rendre accessibles à quiconque, ce qui n'est pas le cas dans le domaine Apple Macintosh par exemple.

Mais une troisième raison parle à l'encontre de cet adaptateur vidéo. L'interface logicielle conçue par IBM pour accéder à cette carte ne pouvait être comprise que par les spécialistes. Bien que cette interface soit dotée d'une orientation extraordinaire, il n'en reste pas moins que la faculté de l'électronique en termes de vitesse laissait à désirer. Si on prend en considération le prix élevé de cette carte, il ne faut pas s'étonner que le standard 8514/A n'ait trouvé que quelques rares clients.

Même si à l'avenir ces cartes sont devenues bon marché et rendues accessibles au bus ISA grâce à un Chip compatible 8514/A de Chips & Technologies, le niveau d'acceptation de ce standard ne s'est pas pour autant amélioré. Avec son standard TIGA, la société Texas Instruments mettait entre-temps les cartes graphiques déjà intelligentes à la disposition du bus ISA à un prix relativement faible et utilisable sur un moniteur quelconque.

TIGA

Depuis plusieurs années déjà, il existe des processeurs graphiques performants dont on se sert pour réaliser des cartes graphiques intelligentes et programmables. A titre d'exemple, on peut citer la 34010 de Texas Instruments et la 82786 d'INTEL. Le règne de la dernière touche déjà à sa fin puisqu'elle ne fera plus partie du circuit courant 1991. Il ne reste plus que la TI 34010 et le modèle ultérieur 34020 dont la demande reste toujours croissante et tient une place prépondérante dans le domaine PC.

Tout comme le standard 8514/A, ces processeurs permettent de développer des cartes graphiques. Parallèlement à une autre routine, celles-ci sont capables de créer des graphiques à travers le processeur et prendre en charge des tâches relativement complexes telles que le calcul des effets miroir dans le Raytracing. La mise en exploitation du processeur n'est pas liée à une résolution écran déterminée. Il peut au contraire être utilisé par les modes compatibles VGA jusqu'aux résolutions écran que l'on rencontre dans le domaine CAD et répondant aux souhaits d'un utilisateur PC.

Pour que les différentes cartes puissent être programmées selon la même méthode malgré leurs diverses résolutions, Texas Instruments a créé une interface logicielle uniforme avec le standard TIGA. TIGA signifie par conséquent "Texas Instruments Graphics Architecture" et décrit une série de fonctions pouvant être appelées facilement par un programme en langage évolué également pour communiquer des commandes au processeur graphique.

Il existe actuellement de nombreuses cartes TIGA sur le marché. Parmi ces dernières, une provient de la société Hercules qui s'est servie du succès remporté par la Hercules Graphics Card. Le rapport qualité/prix de cette carte est intéressant d'autant plus qu'elle peut être utilisée en mode TIGA sur un moniteur VGA tout à fait ordinaire et dispose de surcroît d'une carte VGA intégrée.

Utilisée en mode TIGA sous Windows, cette carte offre déjà ce que l'on pouvait espérer de mieux des cartes vidéo programmables dans les années 1990. En effet, par rapport à une carte VGA ordinaire, la construction de l'écran s'effectue ici cinq fois plus vite sinon plus, sans compter que l'évolution ne s'arrêtera certainement pas à ce point. A l'heure actuelle, le standard TIGA apparaît comme le postulant idéal pour succéder aux cartes VGA et Super VGA.

4.2. Le BIOS vidéo

Le BIOS en ROM du PC offre des services de toutes sortes concernant la sortie sur l'écran. Ces services interviennent dans le cadre de l'interruption 10h désignée également par interruption vidéo. Pour se limiter aux services rendus par les autres fonctions du BIOS, on parle ici de BIOS vidéo qui fait appel à toutes les fonctions de l'interruption 10h.

Cette section traite les sujets suivants :

- ✓ Description des fonctions du BIOS vidéo et leur mode d'utilisation
- ✓ Pourquoi vaut-il parfois mieux avoir recours à ces fonctions plutôt qu'accéder directement à l'électronique vidéo ?

Le BIOS vidéo et ses extensions

Dans sa forme originale, l'interruption 10h ne propose que des fonctions accessibles avec les cartes MDA et CGA. Il n'est pas utile de spécifier que les cartes Hercules sont également reconnues, du moins en mode texte où elles sont entièrement compatibles avec le standard MDA.

Les cartes EGA et VGA ainsi que leurs techniques étendues en modes texte et graphique ne sont nullement soutenues par le BIOS original. Cela explique pourquoi les cartes EGA et VGA contiennent une extension BIOS qui vient s'intégrer automatiquement dans l'interruption 10h lors du lancement du système pour l'adapter à d'autres fonctions nécessaires aux cartes EGA/VGA. Ainsi, le BIOS VGA contient les mêmes extensions que le BIOS EGA sinon davantage. Bien que le nombre de constructeurs de cartes EGA et VGA soit indéfinissable, ils se conforment pourtant tous aux extensions BIOS

imposées par IBM avec ses cartes EGA et VGA. Il ne faut donc pas s'étonner si elles bénéficient de l'appellation "compatibles EGA/VGA".

Mais il s'est avéré que les gros systèmes PC se verront de plus en plus équipés de cartes VGA dont l'électronique est implantée directement sur la carte-mère afin de garantir une rapidité de communication entre la carte vidéo et l'unité centrale. Ces systèmes étant conçus également pour travailler en collaboration avec une carte VGA, les fonctions étendues du BIOS EGA et VGA seront transférées dans le BIOS en ROM de la carte-mère si bien qu'elles ne jouent plus le rôle d'extensions. Leur fonctionnement est identique à celui de ces fonctions.

Rapidité des fonctions BIOS

Les fonctions du BIOS vidéo ne représentent pas la seule et unique solution permettant d'afficher des caractères sur l'écran ou de positionner le curseur. L'interface des fonctions du DOS contient également des fonctions de sortie écran capables d'assurer naturellement tous les services rendus par le BIOS par la programmation directe de l'électronique vidéo. A bien des égards, les services du BIOS se rapprochent des fonctions du DOS et de la programmation directe de l'électronique. Il faut citer en particulier les aspects tels que la vitesse d'exécution, la souplesse et l'indépendance par rapport au périphérique qui ne sont pas toujours faciles à mettre en harmonie.

Par exemple, DOS garantit la meilleure indépendance par rapport au périphérique parce que les sorties peuvent être dirigées sans encombre vers une imprimante ou un fichier. Mais les fonctions nécessaires à cet effet sont très lentes et peu souples. En revanche, la programmation directe de l'électronique offre une plus grande vitesse d'exécution et plus de souplesse en déléguant le contrôle absolu des opérations au programmeur. Il faut toutefois noter qu'elle est extrêmement tributaire de l'électronique et les routines conçues par exemple pour la sortie de caractères sur les cartes CGA ne fonctionnement pratiquement pas avec les cartes MDA.

Les fonctions BIOS ne sont certainement pas aussi rapides que les routines d'accès direct à l'électronique, mais fonctionnent sans contexte avec n'importe que type de carte vidéo. Ici, le programmeur n'a pas besoin de prêter attention aux différences entre les cartes CGA et MDA puisque le BIOS se charge de ces opérations.

Deux raisons principales sont à l'origine de la lenteur des fonctions BIOS par rapport à l'accès direct à l'électronique. Tout d'abord, il s'agit du mécanisme utilisé pour appeler les fonctions BIOS. Ensuite, l'appel d'une interruption nécessite un temps plus long que l'appel d'une routine à l'intérieur d'un programme. Là est le problème majeur des routines BIOS qui tendent à ralentir la vitesse d'exécution. La plupart des cartes vidéo sont surtout des cartes 8 bits si bien que les accès au BIOS s'effectuent beaucoup plus lentement que les accès à la RAM. Dans les PC modernes, cette dernière peut par exemple être réclamée avec des unités 16 ou 32 bits. Si on considère que l'unité centrale

doit faire passer chaque instruction de langage machine dans les routines BIOS à travers 8 bits, on comprend alors pourquoi les routines sont si lentes.

Ce n'est pas sans raison qu'une technique est apparue dans les PC pour charger le BIOS en ROM dans une zone RAM - appelée Shadow ROM - située entre la RAM vidéo et la limite des 1 Mo. Les différentes routines BIOS s'exécutent ainsi relativement vite grâce aux accès 16 ou 32 bits, mais les principaux inconvénients par rapport à la programmation directe de l'électronique ne peuvent pas pour autant être écartés.

Les fonctions BIOS offrant de nombreux services utiles, la plupart des applications PC fonctionnent également avec les fonctions BIOS. Mais elles comportent également des routines accédant directement à l'électronique de la carte vidéo, en particulier pour accélérer la sortie écran en modes texte et graphique. Les routines graphiques du BIOS sont encore moins utilisées que les fonctions de sortie écran. Mais on fait appel au BIOS lorsqu'il s'agit notamment des tâches de contrôle telles que l'initialisation d'un mode vidéo, la sélection d'une page écran ou le positionnement du curseur de texte. C'est ce qu'on appelle le partage obligé.

Les services rendus par le BIOS vidéo

Ce chapitre ne décrit pas tous les services rendus par le BIOS vidéo. Il se limite aux principales fonctions de contrôle et les tâches liées à la sortie écran en mode texte. Les autres fonctions sont présentées dans les sections suivantes comme par exemple 4.8.2 consacrée à la programmation des jeux de caractères avec les cartes EGA et VGA. Vous trouverez la liste de toutes les fonctions du BIOS vidéo en annexe.

Etablissons d'abord la liste des diverses tâches assurées par les différentes fonctions de l'interruption vidéo du BIOS sans oublier les sous-fonctions.

Les fonctions du BIOS vidéo et leur reconnaissance avec les cartes EGA, VGA et le BIOS standard

Nº	Tâche	BIOS*
00h	Fixer le mode vidéo	SEV
01h	Définir la taille du curseur	SEV
02h	Fixer la position du curseur	SEV
03h	Consulter la position du curseur	SEV
04h	Tester le crayon optique	SEV
05h	Définir la page écran actuelle	SEV
06h	Faire défiler l'écran vers le haut	SEV
07h	Faire défiler l'écran vers le bas	SEV

Les fonctions du BIOS vidéo et leur reconnaissance avec les cartes EGA, VGA et le BIOS standard

Nº	Tâche	BIOS*
08h	Lire le caractère et l'attribut	SEV
09h	Ecrire le caractère et l'attribut	SEV
0Ah	Ecrire le caractère à la position du curseur	SEV
0Bh	Définir la palette de couleurs pour le mode graphique	SEV
0Ch	Fixer un point de l'écran en mode graphique	SEV
0Dh	Tester un point écran en mode graphique	SEV
0Eh	Sortie de caractère	SEV
0Fh	Déterminer le mode vidéo	SEV
10h	Définitions de couleurs pour EGA/VGA	EV
11h	Accès au générateur de caractères	EV
12h	Fixer/Consulter la configuration vidéo	EV
13h	Ecrire une chaîne de caractères (sur l'AT uniquement)	SEV
14h	Réservé	---
15h	Réservé	---
16h	Réservé	---
17h	Réservé	---
18h	Réservé	---
19h	Réservé	---
1Bh	Commutation entre deux cartes vidéo	V
1Ch	Sauvegarder/Restaurer l'état de la carte vidéo	V

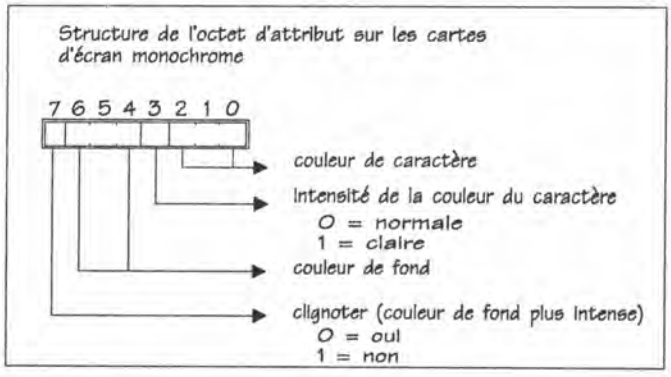
* **S = BIOS standard**
E = BIOS EGA
V = BIOS VGA

Une règle générale s'applique à l'appel de toutes les fonctions : Le registre AH doit être chargé avec le numéro de fonction avant l'appel de l'interruption vidéo du BIOS. S'il faut appeler une sous-fonction, le numéro de la sous-fonction doit être généralement placé dans le registre AL. Il existe évidemment des exceptions qui ne seront pas citées dans ce livre. L'exception à la règle est représentée par les fonctions retournant des valeurs de toutes sortes dans les registres du processeur. Normalement, le contenu des divers registres reste inchangé par les différentes fonctions.

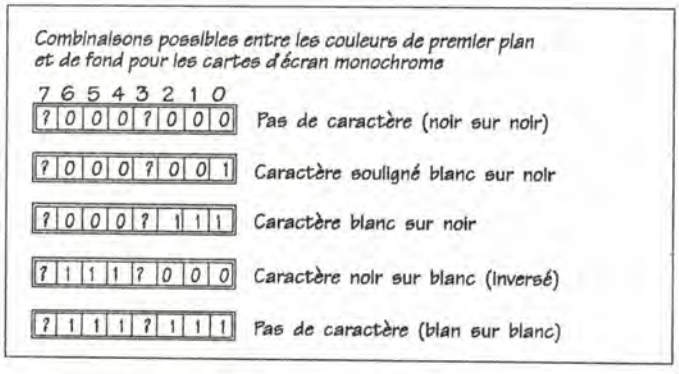
Sélection de la couleur des caractères et du fond

Certaines fonctions chargées de sortir des caractères sur l'écran attendent une valeur spécifiant la couleur du premier plan et du fond. Dans les diverses cartes vidéo du PC, les caractères peuvent en effet être dotés d'une couleur de premier plan différente de celle du fond. Les cartes vidéo monochromes et couleur codifient ces couleurs selon des méthodes complètement différentes, la diversité de leurs techniques ne se reflétant fort heureusement que sur le mode d'attribution des couleurs. Dans les deux cas, un octet de couleur ou d'attribut réparti entre deux quartets est réservé pour chaque caractère. Le quartet de poids faible c'est-à-dire les bits 0 à 3 définissent la couleur du premier plan, les bits 4 à 7 - représentant le quartet de poids fort - la couleur du fond.

Dans les cartes monochromes, un autre bit vient s'ajouter dans les deux quartets pour l'intensité de la couleur de premier plan et le clignotement du caractère, comme le montre la figure suivante.



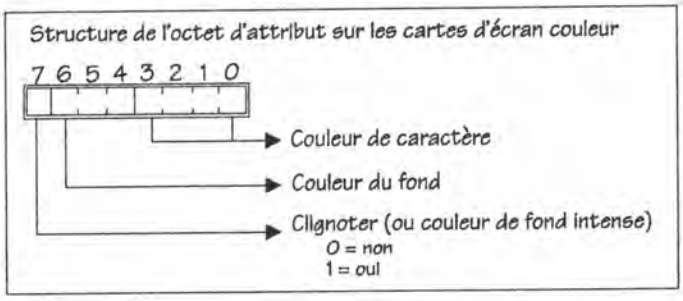
Si on fait abstraction du bit d'intensité pour la couleur de premier plan et de fond, on peut alors affecter des combinaisons de bits précises aux couleurs de premier plan et de fond, comme le montre la figure suivante.



Les couleurs fréquemment sélectionnées sur les cartes écran monochromes sont les couleurs 07h et 70h. Les premières installent une police claire sur un fond foncé et les secondes placent une police foncée sur un fond clair, ce qui donne une police en inversion vidéo. Ces codes intéressent également les cartes couleur dans la mesure où ils complètent les facultés des cartes puisqu'ils servent notamment à créer une police gris clair sur un fond noir ou une police noire sur un fond gris clair.

La sélection des valeurs 0 à 7 comme couleur de police ou de fond empêche de spécifier en toute conscience le bit 7 dans l'octet d'attribut. Le résultat est d'interdire le clignotement des caractères au cas où la carte vidéo est configurée de telle manière qu'elle provoque le clignotement des caractères au lieu de leur affecter une couleur de fond intensive.

Les octets de couleur sur les cartes couleur sont organisés pratiquement de la même manière. Mais ici, vous pouvez sélectionner une véritable couleur pour le premier plan et le fond à partir d'une palette prédéfinie composée de 16 couleurs. Les deux figures suivantes montrent la structure de l'octet de couleur sur des cartes couleur et la palette disponible.



La palette de couleurs de la carte couleur

Décimal	Hexa	Binaire	Couleur
0	00h	0000b	Noir
1	01h	0001b	Bleu
2	02h	0010b	Vert
3	03h	0011b	Turquoise
4	04h	0100b	Rouge
5	05h	0101b	Violet
6	06h	0110b	Marron
7	07h	0111b	Gris clair
8	08h	1000b	Gris foncé
9	09h	1001b	Bleu clair

La palette de couleurs de la carte couleur			
Décimal	**Hexa**	**Binaire**	**Couleur**
10	0Ah	1010b	Vert clair
11	0Bh	1011b	Turquoise clair
12	0Ch	1100b	Rouge clair
13	0Dh	1101b	Violet clair
14	0Eh	1110b	Jaune
15	0Fh	1111b	Blanc

Le jeu de caractères ASCII du PC

Le PC utilise un jeu composé de 256 caractères. Mis à part les nombres, lettres et caractères génériques, il contient également un grand nombre d'autres caractères. Il s'agit entre autres des caractères de cadre et des symboles mathématiques sans oublier les caractères en langue étrangère. La représentation des caractères du jeu en leur code ASCII correspondant s'effectue à l'aide des diverses fonctions du BIOS vidéo comme le montre la table suivante.

Table ASCII du jeu de caractères PC

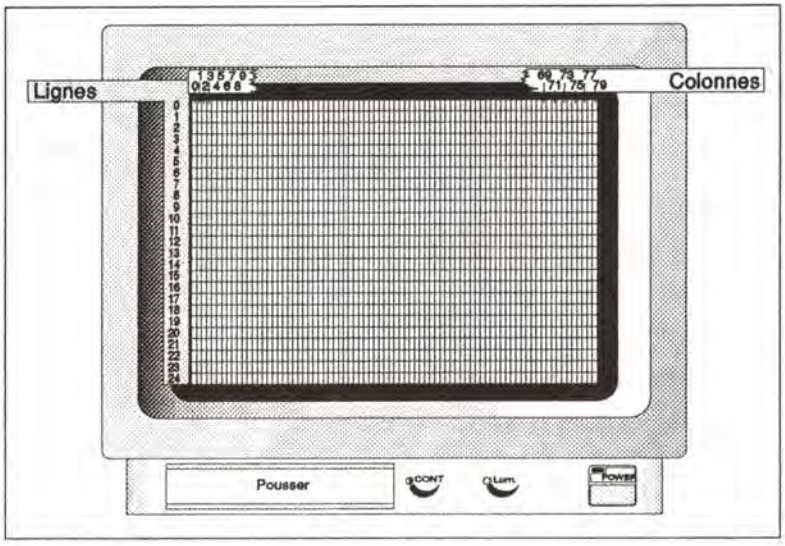
Déc.	Hex.	Caract.	Déc.	Hex.	Caract.	Déc.	Hex.	Caract.	Déc.	Hex.	Caract.
0	00		32	20		64	40	@	96	60	`
1	01	☺	33	21	!	65	41	A	97	61	a
2	02	☻	34	22	"	66	42	B	98	62	b
3	03	♥	35	23	#	67	43	C	99	63	c
4	04	♦	36	24	$	68	44	D	100	64	d
5	05	♣	37	25	%	69	45	E	101	65	e
6	06	♠	38	26	&	70	46	F	102	66	f
7	07	•	39	27	'	71	47	G	103	67	g
8	08	◘	40	28	(	72	48	H	104	68	h
9	09	○	41	29	)	73	49	I	105	69	i
10	0A	◙	42	2A	*	74	4A	J	106	6A	j
11	0B	♂	43	2B	+	75	4B	K	107	6B	k
12	0C	♀	44	2C	,	76	4C	L	108	6C	l
13	0D	♪	45	2D	-	77	4D	M	109	6D	m
14	0E	♫	46	2E	.	78	4E	N	110	6E	n
15	0F	☼	47	2F	/	79	4F	O	111	6F	o
16	10	►	48	30	0	80	50	P	112	70	p
17	11	◄	49	31	1	81	51	Q	113	71	q
18	12	↕	50	32	2	82	52	R	114	72	r
19	13	‼	51	33	3	83	53	S	115	73	s
20	14	¶	52	34	4	84	54	T	116	74	t
21	15	§	53	35	5	85	55	U	117	75	u
22	16	▬	54	36	6	86	56	V	118	76	v
23	17	↨	55	37	7	87	57	W	119	77	w
24	18	↑	56	38	8	88	58	X	120	78	x
25	19	↓	57	39	9	89	59	Y	121	79	y
26	1A	→	58	3A	:	90	5A	Z	122	7A	z
27	1B	←	59	3B	;	91	5B	[	123	7B	{
28	1C	∟	60	3C	<	92	5C	\	124	7C	¦
29	1D	↔	61	3D	=	93	5D	]	125	7D	}
30	1E	▲	62	3E	>	94	5E	^	126	7E	~
31	1F	▼	63	3F	?	95	5F	_	127	7F	⌂

Déc.	Hex.	Caract.	Déc.	Hex.	Caract.	Déc.	Hex.	Caract.	Déc.	Hex.	Caract.
128	80	Ç	160	A0	á	192	C0	└	224	E0	α
129	81	ü	161	A1	í	193	C1	┴	225	E1	ß
130	82	é	162	A2	ó	194	C2	┬	226	E2	Γ
131	83	â	163	A3	ú	195	C3	├	227	E3	π
132	84	ä	164	A4	ñ	196	C4	─	228	E4	Σ
133	85	à	165	A5	Ñ	197	C5	┼	229	E5	σ
134	86	å	166	A6	ª	198	C6	╞	230	E6	µ
135	87	ç	167	A7	º	199	C7	╟	231	E7	τ
136	88	ê	168	A8	¿	200	C8	╚	232	E8	Φ
137	89	ë	169	A9	⌐	201	C9	╔	233	E9	Θ
138	8A	è	170	AA	¬	202	CA	╩	234	EA	Ω
139	8B	ï	171	AB	½	203	CB	╦	235	EB	δ
140	8C	î	172	AC	¼	204	CC	╠	236	EC	∞
141	8D	ì	173	AD	¡	205	CD	═	237	ED	φ
142	8E	Ä	174	AE	«	206	CE	╬	238	EE	ε
143	8F	Å	175	AF	»	207	CF	╧	239	EF	∩
144	90	É	176	B0	▒	208	D0	╨	240	F0	≡
145	91	æ	177	B1	▒	209	D1	╤	241	F1	±
146	92	Æ	178	B2	▓	210	D2	╥	242	F2	≥
147	93	ô	179	B3	│	211	D3	╙	243	F3	≤
148	94	ö	180	B4	┤	212	D4	╘	244	F4	⌠
149	95	ò	181	B5	╡	213	D5	╒	245	F5	⌡
150	96	û	182	B6	╢	214	D6	╓	246	F6	÷
151	97	ù	183	B7	╖	215	D7	╫	247	F7	≈
152	98	ÿ	184	B8	╕	216	D8	╪	248	F8	°
153	99	Ö	185	B9	╣	217	D9	┘	249	F9	∙
154	9A	Ü	186	BA	║	218	DA	┌	250	FA	·
155	9B	¢	187	BB	╗	219	DB	▮	251	FB	√
156	9C	£	188	BC	╝	220	DC	▄	252	FC	ⁿ
157	9D	¥	189	BD	╜	221	DD	▌	253	FD	²
158	9E	₧	190	BE	╛	222	DE	▐	254	FE	■
159	9F	ƒ	191	BF	┐	223	DF	▀	255	FF	

Le système des coordonnées de l'écran

En guise de paramètres d'entrée, la plupart des fonctions BIOS attendent des coordonnées écran permettant d'adresser une position précise sur l'écran. Il est donc indispensable de connaître le système de coordonnées défini pour l'appel de nombreuses fonctions.

Le point d'origine du repère se situe dans le coin supérieur gauche de l'écran quel que soit le mode activé (mode texte ou mode graphique). Il porte les coordonnées (0/0). La numérotation des ordonnées X ou Y commence à partir de ce point en allant respectivement vers la droite ou la gauche. En mode 80*25 caractères, les coordonnées de la portion inférieure droite de l'écran correspondent à (79/24). En mode graphique 640*200 points de la carte CGA, elle se situe aux coordonnées (639*199).



Numérotation des lignes et colonnes de l'écran

Initialisation d'un mode vidéo

La fonction 00h joue un rôle important. Elle est d'ailleurs rarement remplacée par la programmation directe de l'électronique. Grâce à un code de mode indiqué dans le registre AL, elle permet d'activer tous les modes vidéo standard des cartes MDA jusqu'aux cartes VGA. Cela vaut à la fois pour le mode texte et le mode graphique. La seule exception rencontrée ici est le mode graphique de la carte Hercules.

La condition préalable pour l'initialisation d'un mode vidéo est naturellement l'existence d'une carte vidéo. Cette fonction ne permet toutefois pas de le vérifier car l'échec de l'opération n'est pas signalé à l'utilisateur.

Codage des modes vidéo par la fonction 00h du BIOS vidéo

Code	Mode	Carte
00h	40*25 caractères de texte, 16 couleurs, pas d'affichage couleur	CEV
01h	40*25 caractères de texte, 16 couleurs	CEV
02h	80*25 caractères de texte, 16 couleurs, pas d'affichage couleur	CEV
03h	80*25 caractères de texte, 16 couleurs	CEV
04h	320*200 points graphiques, 4 couleurs	CEV
05h	320*200 points graphiques, 4 couleurs	CEV
06h	640*200 points graphiques, 2 couleurs	CEV
07h	80*25 caractères, monochrome	MH E*
08h	réservé	---
09h	réservé	---
0Ah	réservé	---
0Bh	réservé	---
0Ch	réservé	---
0Dh	320*200 points graphiques, 16 couleurs	EV
0Eh	640*200 points graphiques, 16 couleurs	EV
0Fh	640*350 points graphiques, monochrome	E*
10h	640*350 points graphiques, 16 couleurs	EV
11h	640*480 points graphiques, 2 couleurs	V
12h	640*480 points graphiques, 16 couleurs	V
13h	320*200 points graphiques, 256 couleurs	V

*** Carte EGA sur moniteur MDA**
M = MDA
H = Hercules
C = CGA
E = EGA
V = VGA

L'appel de la fonction 00h ne provoque pas seulement l'initialisation du mode vidéo souhaité, mais áussi la suppression du contenu de la RAM vidéo. Cette action peut être

interdite lors de l'initialisation des modes des cartes EGA et VGA. Dans ce cas, la valeur 128 est rajoutée au numéro de mode ce qui revient à fixer le bit 7 dans le numéro de mode. Le contenu actuel de la RAM vidéo reste alors conservé et réapparaît une fois le mode souhaité initialisé.

Lors du lancement d'un programme, on peut toujours supposer qu'un mode texte 80*25 caractères est activé. Il correspond au mode 7 avec une carte monochrome et au mode 3 avec une carte couleur. Il n'est donc pas indispensable d'appeler la fonction 00h dans la mesure où votre programme doit fonctionner en mode texte 80*25 caractères.

La fonction 0Fh représente l'inverse de la fonction 00h. Elle sert à déterminer le mode vidéo en cours. Cette fonction doit être appelée avec la valeur 0Fh dans le registre AH. En guise de résultat, elle retourne la valeur du mode vidéo dans le registre AL conformément au tableau précédent. En dehors du mode vidéo, cette fonction retourne également le nombre de colonnes par ligne écran dans le mode en cours à condition qu'il s'agisse d'un mode texte. Après appel, cette information se trouve dans le registre AH. Elle renvoie également le numéro de la page écran actuelle dans le registre BH.

Programmation du curseur de texte

En mode texte, toutes les cartes graphiques depuis MDA jusqu'à VGA montrent un curseur de texte clignotant. Il indique la position d'entrée ou de sortie actuelle. Tant l'apparence de ce curseur que son emplacement sur l'écran peuvent être spécifiés avec le BIOS vidéo.

La définition de la position du curseur incombe à la fonction 02h. Pour son appel, elle attend le numéro de fonction 02h dans le registre AH, la ligne dans le registre DH et la colonne en DL où doit apparaître le curseur. Par ailleurs, le registre BH doit contenir le numéro de la page écran devant comporter le curseur. Notez dans ce contexte que chaque page écran dispose d'un curseur individuel. Le déplacement du curseur clignotant à l'aide de cette fonction ne devient visible que lorsque la valeur du registre BH correspond à la page écran actuelle.

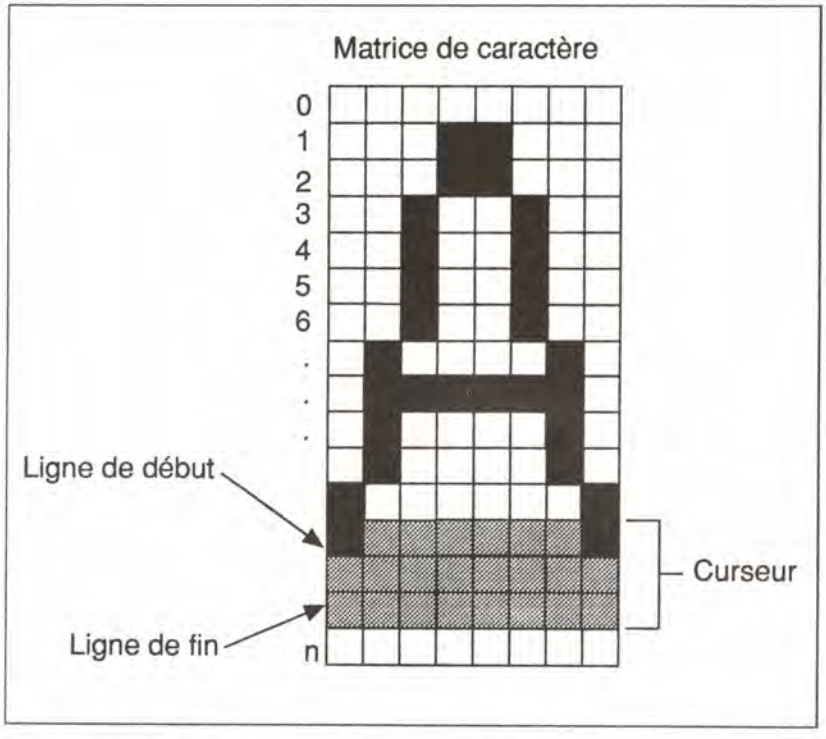
L'appel de cette fonction ne permet pas seulement de déplacer le curseur clignotant sur l'écran mais de fixer simultanément la position où doit commencer la sortie des caractères au moyen des fonctions appropriées. Tout cela est mieux expliqué dans le cadre de la description de ces fonctions.

La fonction 03h représente l'inverse de la fonction 02h. Elle lit la position du curseur dans une page écran déterminée et retourne la valeur au programme. Lors de l'appel de cette fonction, le registre AL doit contenir le numéro de fonction 03h et le registre BH le numéro de la page écran dont il faut lire la position du curseur. Cette fonction retourne également deux autres valeurs dans les registres CH et CL : elles indiquent la ligne de départ et de fin du curseur définissant l'aspect du curseur et non son emplacement.

Pour comprendre la signification de ces paramètres, il faut savoir que chaque caractère (de texte) est représenté par 8 lignes de points avec une carte couleur et par 14 lignes de points avec une carte monochrome (ne pas confondre lignes de points et lignes d'écran). Ainsi, le programmeur peut décider de la ligne où le curseur clignotant doit commencer et de la ligne où il doit se terminer. Il peut définir de ce fait des petits ou grands curseurs qui sont généralement utilisés pour l'affichage de divers modes d'exploitation lors de l'entrée des informations.

La hauteur de la matrice de caractères joue un rôle primordial dans la détermination de ces valeurs. Les différents caractères sont en effet définis dans la matrice. Avec une carte CGA, cela correspond à 8 lignes de points permettant ainsi de définir des valeurs comprises entre 0 et 7 pour les lignes de départ et de fin. Avec une carte Hercules ou MDA, la hauteur de la matrice est de 14 lignes, les lignes de départ et de fin du curseur clignotant devant se trouver alors dans l'intervalle 0 et 13. Cette matrice de caractères est plus importante avec les cartes EGA et VGA. Ici, le BIOS utilise l'unité de mesure CGA allant de 0 à 7 où les indications sont converties automatiquement par rapport à la hauteur réelle de la matrice de caractères.

L'indication de valeurs élevées pour les lignes de départ et de fin provoque la disparition du curseur de l'écran, ce qui est parfois utile.



Lignes de départ et de fin du curseur texte

Contrairement à la fonction 03h qui permet de lire les lignes de départ et de fin du curseur, la fonction 01h sert exclusivement à les définir. Dans ce cas, le registre AH doit être chargé avec 1, le registre CH avec la ligne de départ et le registre CL avec la ligne de fin du curseur clignotant avant l'appel de l'interruption 10h. Notez que la ligne de départ doit être inférieure ou égale à la ligne de fin sinon le curseur risque de devenir invisible.

Sélection de la page écran

Jusqu'à présent, nous avons maintes fois parlé de la page écran actuelle sans pour autant expliquer comment l'activer. La fonction 05h du BIOS vidéo répond à cette interrogation. Elle doit être appelée avec la valeur 05h dans le registre AH et le numéro de la page écran à activer dans le registre AL. Le numéro de la page écran à activer dépend naturellement du nombre de pages écran disponibles dans le mode vidéo en cours par la carte vidéo concernée. Par exemple, l'appel de cette fonction ne présente aucun intérêt avec une carte MDA puisqu'elle ne peut offrir qu'une page écran. Les valeurs suivantes sont autorisées pour les autres cartes vidéo, compte tenu également du mode vidéo :

Nombre de pages de texte disponibles en fonction du mode vidéo et de la carte vidéo

Mode	Résolution	Carte	Pages
7	80*25	MDA/Hercules	1
0/1	40*25	CGA	8
2/3	80*25	CGA	4
0/1	40*25	EGA/VGA	16
2/3	80*25	EGA/VGA	8

La numérotation des pages écran commence toujours à partir de 0. Par exemple avec les cartes EGA et VGA, les pages écran 0 à 7 peuvent être réclamées en mode 2.

Sortie de caractères avec le BIOS

Le BIOS vidéo réserve une grande partie de ses fonctions pour la sortie des caractères. Il existe des différences entre le mode de travail de ces fonctions résultant par exemple de la manipulation des codes de contrôle. Ces noms portent les codes ASCII 7, 8, 10 et 13. En fait, il s'agit de caractères tout à fait ordinaires issus du jeu IBM (voir plus haut). Tirant leur origine de l'ère des terminaux d'imprimante, ils ont acquis une signification importante dans l'histoire de la PAO.

Codes de contrôle du jeu de caractères ASCII du PC		
ASCII	**Nom**	**Fonction**
7	Bell	Emet un signal sonore
8	Backspace	Efface le caractère immédiatement à gauche du curseur et avance le curseur d'une position vers la droite
10	Linefeed	Place le curseur à la ligne suivante
13	Carriage Return	Place le curseur au début la ligne en cours

Certaines fonctions considèrent ces codes comme des caractères ASCII normaux et les affichent par conséquent sur l'écran. D'autres au contraire les évaluent comme des codes de contrôle et émettent par exemple un signal sonore lorsqu'il s'agit du code 7. Vous devez utiliser la fonction qui convient selon que vous souhaitez que les codes de contrôle soient traités ou non.

Il convient de noter ici que toutes les fonctions de sortie de texte peuvent être utilisées en mode texte et en mode graphique. En mode graphique, la sortie de caractères ne coule pas de source puisqu'aucun jeu de caractères n'est disponible ici. Mais le BIOS remédie automatiquement à ce handicap en remplaçant les motifs de caractères des codes ASCII par des points graphiques. Les motifs de caractères des codes ASCII 0 à 127 sont déjà stockés dans la ROM. Quant aux motifs de caractères des codes 128 à 255, ils seront lus à partir d'une table dans la RAM devant être installée préalablement au moyen de la commande DOS GRAFTABL.

Le BIOS tire l'adresse de cette table du pointeur FAR situé à la cellule de mémoire 0000:007C. Ces adresses de mémoire se trouvent certes à l'intérieur de la table des vecteurs d'interruption, mais elle peuvent être utilisées à cet effet. En effet, l'interruption 1Fh dont l'adresse y est normalement ignorée est inutilisée.

Le fait que cette table soit stockée dans la RAM permet de définir une table personnelle. Le résultat est que les caractères spéciaux non répertoriés dans le jeu standard peuvent être affichés sur l'écran. Chaque caractère étant défini par 8 octets, les huit premiers octets de la table concernent le code ASCII 128, les huit suivants le code 129, etc. Chaque octet met le motif du bit à la disposition d'une des 8 lignes à partir de laquelle se crée un caractère. Dans chaque octet, le bit 0 représente le point de droite de la matrice de caractères, le bit 7 le point de gauche. Si un bit est réglé sur 1, cela signifie que le point correspondant est allumé sur l'écran.

Les fonctions 09h et 0Ah servent à sortir des caractères avec une légère différence. La fonction 0Ah représente le caractère dans la couleur déjà définie pour la position correspondante sur l'écran alors que la fonction 09h permet de spécifier également la couleur (l'attribut) du caractère à sortir. Après la sortie du caractère, les deux fonctions ne placent pas le curseur à la position d'écran suivante pour que la sortie des caractères s'effectue au même endroit lors d'un appel renouvelé de ces fonctions.

Pour sortir un texte homogène, il convient donc de placer le curseur à la position d'écran suivante au moyen de la fonction 02h après l'appel.

Sortie de chaîne de caractères

Avec l'introduction de l'AT, une fonction est venue s'ajouter dans le BIOS vidéo disponible également dans les versions étendues du BIOS des cartes EGA et VGA. Elle permet enfin de sortir une chaîne de caractères sur l'écran avec un seul appel de la fonction sans qu'il soit nécessaire d'appeler une fonction de sortie différente pour chaque caractère. Cette fonction porte le numéro 13h.

En dehors du numéro de fonction dans le registre AH, il faut transmettre toute une série d'autres arguments pour son appel. Le registre BH contient le numéro de la page écran dans laquelle la chaîne doit sortir. Dans ce cas, il doit s'agir obligatoirement de la page écran actuelle. La position de départ de la chaîne sur l'écran se trouve dans le registre DH (ligne) et dans le registre DL (colonne). La sortie n'est donc pas tributaire de la position actuelle du curseur. Le registre CX contient le nombre de caractères à sortir dans la chaîne.

Le contenu du registre AL définit l'un des quatre modes autorisés pour sortir la chaîne. Le format de la chaîne de caractères à sortir dans les modes 0 et 1 est différent de celui des modes 2 et 3. Dans les modes 2 et 3, chaque code ASCII d'un caractère doit être suivi de l'octet d'attribut correspondant. Dans les modes 0 et 1, les différents caractères de la chaîne sont consécutifs. Dans ces modes, l'octet d'attribut est spécifié par le contenu du registre BL pour tous les caractères. Mais dans tous les cas, un pointeur FAR doit être transmis sur le buffer dans la paire de registres ES:BP indépendamment de la structure du buffer.

Bien que dans les modes 2 et 3, deux octets soient réservés à chaque caractère de la chaîne et de ce fait une chaîne de 4 caractères de long nécessite par exemple 8 octets, il faut tout de même indiquer uniquement le nombre de caractères à sortir dans le registre CX. Dans l'exemple cité, cela revient à 4. Il existe une autre différence entre les modes 0 et 2 et les modes 1 et 3. Après la sortie de la chaîne en modes 1 et 3, le curseur vient se placer à la position écran qui fait suite au dernier caractère de la chaîne. La sortie de caractères suivante à travers une fonction BIOS s'effectue alors à cette position d'écran. Cette réactualisation de la position du curseur reste inhibée dans les modes 0 et 2.

Lire les caractères de l'écran

Contrairement fonctions 09h, 0Ah et 0Eh qui servent à sortir des caractères sur l'écran, la fonction 08h permet de les lire, c'est-à-dire déterminer la position et l'attribut de chaque caractère sur l'écran. Pour l'appel de cette fonction, le registre AH doit contenir la valeur 8 et le registre BH le numéro de la page écran nécessaire. La position écran à partir de laquelle le caractère sera lu représente la position actuelle du curseur dans la page spécifiée.

Les deux fonctions interprètent les codes de contrôle décrits plus haut comme des caractères normaux et n'hésitent pas à les sortir. Elles doivent être appelées avec le numéro de fonction chargé dans le registre AH et le code ASCII du caractère à sortir dans le registre AL. La page écran devant contenir le caractère se trouve comme d'habitude dans le registre BH. Le registre CX contient un nombre indiquant la fréquence de la sortie. Autrement dit, un appel de fonction permet de sortir plusieurs fois le même caractère, méthode qui économise du temps. Si le caractère ne doit sortir qu'une seule fois dans le registre AL, il suffit alors de charger la valeur 1 dans le registre CX avant l'appel.

En cas d'erreur dans le BIOS, le facteur de répétition sélectionné en mode graphique lors de l'appel de cette fonction doit être suffisamment élevé pour que tous les caractères à afficher puissent être contenus dans une ligne.

En voilà assez à propos des registres de la fonction 0Ah. La fonction 09h permettant de spécifier également la couleur du caractère à sortir, le registre BL intervient ici aussi. Il est destiné à transmettre la couleur des caractères.

L'inconvénient majeur de ces deux fonctions est que l'écriture de la position du curseur ne se poursuit pas après appel des fonctions. La fonction 0Eh arrange la situation. Elle simule un terminal ou un périphérique à distance ce qui lui vaut d'ailleurs l'appellation anglaise de routine TTY (TTY = Teletype). Son appel provoque le déplacement du curseur vers la page écran suivante après la sortie d'un caractère. S'il se trouve déjà à la fin d'une ligne écran, un saut se produit vers le début de la ligne qui suit.

Si aucune ligne ne fait suite parce que le curseur se trouve déjà dans le coin inférieur droit de l'écran (ligne 24, colonne 79), le contenu de l'écran décale alors d'une ligne vers le haut. Cette action fait disparaître la première ligne écran du champ de vision. Ensuite, la ligne 24 est effacée et le curseur vient se placer au début de cette ligne en vue d'une autre sortie de caractères.

Contrairement aux fonctions 09h et 0Ah, la fonction TTY n'interprète pas les codes de contrôle comme des codes ASCII ordinaires mais les traite conformément à leur fonction. Tout comme la fonction 0Ah, la fonction TTY reprend la couleur de la page écran concernée lors de la sortie d'un caractère. Cela ne s'applique cependant qu'aux modes texte. En mode graphique, la couleur de premier plan du caractère doit être transmise à la fonction TTY dans le registre BL.

Lors de son appel, cette fonction attend également le numéro de fonction 0Eh dans le registre AH, le code du caractère à sortir dans le registre AL et la page écran où doit apparaître le caractère dans le registre BH.

En mode texte, le code du caractère peut être lu directement dans la RAM. Pour l'obtenir en mode graphique, le motif du caractère situé à la position actuelle du curseur est comparé avec les motifs de tous les caractères. Cela ne fonctionne pas dans tous les cas, c'est pourquoi il ne faut pas attendre beaucoup de cette fonction en mode graphique.

En guise de résultat, vous obtenez l'attribut (la couleur) dans le registre AH et le code ASCII du caractère à lire dans le registre AL après appel de la fonction.

Défilement de l'écran

Dans le contexte de la fonction 0Eh (sortie TTY), il convient de parler de la technique permettant, si cela est nécessaire, de décaler (ou scrolling en anglais) l'écran d'une ligne vers le haut ou le bas.

Cette faculté est octroyée à la fonction 0Eh à travers un appel interne de la fonction 06h disponible également par le programmeur. Elle permet de décaler une portion précise de l'écran d'une ou plusieurs lignes vers le haut ou de compléter par des espaces. Cette opération ne peut se réaliser que sur la page écran actuelle. Pour appeler cette fonction, le registre AH doit être chargé avec le numéro de fonction 06h et le registre AL avec le nombre de lignes à prendre en compte pour décaler la portion spécifiée de l'écran vers le haut. La valeur 0 signale au BIOS que la portion de l'écran ne peut pas être décalée. Il faut au contraire la remplir d'espaces. Exceptionnellement, le registre BH n'attend pas la page écran mais la couleur à affecter aux lignes blanches (ou à la ligne blanche). La portion d'écran à traiter par la fonction est définie à l'aide des registres CH, CL, DH et DL.

CH	Ligne du coin supérieur gauche de la fenêtre
CL	Colonne du coin supérieur gauche de la fenêtre
DH	Ligne du coin inférieur droit de la fenêtre
DL	Colonne du coin inférieur droit de la fenêtre

Outre la fonction 06h, le BIOS reconnaît la fonction 07h qui exécute la même tâche. Cette fois, la portion d'écran spécifiée subit un défilement vers le bas. L'affectation des registres reste identique à celle de la fonction 06h si on fait abstraction des divers numéros de fonction.

Exemples de programmes

Ce chapitre se termine par trois programmes rendant accessibles des fonctions de l'interruption vidéo du BIOS à partir de langages évolués. Il s'agit notamment des fonctions de contrôle et des fonctions d'accès à l'écran en mode texte.

L'avantage de ces programmes, du moins en Pascal et C, est que la sortie écran s'effectue plus vite avec les fonctions BIOS qu'avec les fonctions et procédures prédéfinies sous DOS. L'utilisation des fonctions BIOS en BASIC n'est pas importante car la sortie écran à l'aide de ces dernières s'effectue encore plus lentement que l'instruction PRINT elle-même déjà lente.

L'accès aux fonctions BIOS dans ce langage offre toutefois un avantage. Elles peuvent être appelées là où elles ne peuvent pas être adressées à l'aide d'instructions prédéfinies. C'est par exemple le cas des deux fonctions BIOS servant à faire défiler une fenêtre quelconque vers le haut ou le bas et ce, dans les trois langages.

Il ne faut pas non plus oublier de souligner un inconvénient important lié à la sortie écran au moyen des fonctions BIOS : les instructions de sortie écran en langages évolués acceptent également des variables numériques que vous devez d'abord convertir - en tenant compte d'un nombre d'emplacements précis ou d'une précision exacte - en codes ASCII avant de pouvoir les afficher. Cela n'est pas possible avec les fonctions BIOS. Si vous souhaitez malgré tout sortir des variables numériques avec les fonctions BIOS, vous devez d'abord les convertir en une chaîne à l'aide de la fonction adéquate et transmettre ensuite cette chaîne à la fonction BIOS. Une procédure qui est naturellement gourmande en temps...

Le programme d'exemple est la réunion de ces trois programmes. Il remplit d'abord l'écran avec des caractères issus du jeu PC et ouvre ensuite deux fenêtres où deux flèches sont à déplacer de haut en bas. Nous vous montrerons comment réaliser cela en pratique après avoir examiné de plus près les exemples. Il existe une homogénéité entre les trois programmes en vue de garantir la compatibilité entre les cartes monochromes et couleur. Ils n'utilisent qu'une seule page écran et ne contiennent ni des sous-programmes ou fonctions ni des procédures pour appeler les fonctions graphiques du BIOS.

Si vous avez compris ce chapitre, il ne vous sera pas difficile de compléter le programme concerné par les fonctions qui y font défaut et d'écrire un petit programme de démonstration. Vous allez certainement animer ces programmes en sachant d'une part que l'appel de l'interruption vidéo du BIOS garantit l'ordinateur contre un plantage ou quelque chose de grave et d'autre part qu'il ne vous reste qu'à donner libre cours à votre imagination pour organiser l'écran. Mais avant de vous lancer à l'aventure, examinons chaque programme en détail.

Listing : VIDEOP.PAS

```
{*****************************************************************}
{*                          V I D E O P                          *}
{*---------------------------------------------------------------*}
{*   Fonction     : fournit  quelques fonctions qui exploitent   *}
{*                  l'interruption vidéo du BIOS et qui ne sont pas *}
{*                  encore intégrées dans Turbo Pascal           *}
{*---------------------------------------------------------------*}
{*   Auteur       : MICHAEL TISCHER                              *}
{*   Développé le : 10.07.1987                                   *}
{*   Dernière MAJ : 14.01.1991                                   *}
{*****************************************************************}

program VIDEOP;

Uses Crt, Dos;                          { Inclut les unités CTR et DOS }

const NORMAL      = $07;                { Définit les attributs }
      CLAIR       = $0f;                {   pour les cartes }
      INVERSE     = $70;                {   écran monochromes }
      SOULIGNE = $01;
      CLIGNOTANT  = $80;

type TextTyp = string[80];

var  i,                                 { Variables pour les boucles }
     j,
     k,
     l       : integer;
     IString : string[2];               { Nombre de flèches tracées }

{*****************************************************************}
{* GetVideoMode : Détermine le mode vidéo courant et divers autres *}
{*                paramètres                                     *}
{* Entrée       : néant                                         *}
{* Sorties      : Les variables citées plus bas sont mises à jour *}
{*****************************************************************}

procedure GetVideoMode(var ModeVideo,   { Numéro du mode vidéo courant }
                       Nombre,          { Nombre de colonnes par ligne }
                   Page : integer);     { page d'écran courante }

var Regs : Registers;                   { Registres pour l'interruption }

begin
  Regs.ah := $0F;                       { Numéro de la fonction }
  Intr($10, Regs);                  { Déclenche l'interruption vidéo du BIOS }
  ModeVideo := Regs.al;                 { Numéro du mode vidéo }
  Nombre := Regs.ah;                    { Nombre de caractères par ligne }
  Page := Regs.bh;                      { Numéro de la page d'écran courante }
end;

{*****************************************************************}
{* SetCursorType : Définit l'aspect du curseur clignotant        *}
{* Entrée        : Voir plus bas                                 *}
{* Sortie        : néant                                         *}
{* Info          : Les paramètres peuvent être compris entre 0 et 13  *}
{*                 pour une carte monochrome, 0 et 7 pour une carte  *}
{*                 couleur                                       *}
{*****************************************************************}

procedure SetCursorType(Debut,          { Ligne de début du curseur }
                        Fin   : integer);   { Ligne de fin du curseur }

var Regs : Registers;                   { Registres pour l'interruption }

begin
  Regs.ah := 1;                         { Numéro de la fonction }
  Regs.ch := Debut;                 { Fixe l'extrémité supérieure du curseur }
  Regs.cl := Fin;                   { Fixe l'extrémité inférieure du curseur }
  Intr($10, Regs);                  { Déclenche l'interruption vidéo du BIOS }
end;

{*****************************************************************}
{* SetCursorPos : Fixe la position du curseur dans la page       *}
{*                d'écran indiquée                               *}
{* Entrée       : Voir plus bas                                 *}
{* Sortie       : néant                                         *}
{* Info         : La position du curseur clignotant ne change à *}
{*                l'issue de cet appel que si la page d'écran indiquée*}
{*                est la page d'écran courante                  *}
{*****************************************************************}

procedure SetCursorPos(Page,                { Page d'écran concernée }
                       Colonne,             { nouvelle colonne du curseur }
                       Ligne   : integer);  { nouvelle ligne du curseur }

var Regs : Registers;                   { Registres pour l'interruption }

begin
  Regs.ah := 2;                         { Numéro de la fonction }
  Regs.bh := Page;                      { Page d'écran }
  Regs.dh := Ligne;                     { Fixe les coordonnées }
  Regs.dl := Colonne;                   { de l'écran }
  Intr($10, Regs);                  { Déclenche l'interruption vidéo du BIOS }
end;

{*****************************************************************}
{* GetCursorPos : Détermine la position du curseur dans la page  *}
{*                d'écran ainsi que les limites de son aspect    *}
{* Entrée       : Voir plus bas                                 *}
{* Sortie       : Les variables citées plus bas sont mises à jour *}
{* Info         : Les limites du curseur sont indépendantes de la *}
{*                page écran indiquée                           *}
{*****************************************************************}

procedure GetCursorPos(Page : integer;          { Page d'écran }
                   var Colonne,                 { Colonne du curseur }
                       Ligne,                   { Ligne du curseur }
                       Debut,                   { Extrémité supérieure du curseur }
                       Fin     : integer);      { Extrémité inférieure }

var Regs : Registers;                   { Registres pour l'interruption }

begin
  Regs.ah := 3;                         { Numéro de la fonction }
  Regs.bh := Page;                      { Page d'écran }
  Intr($10, Regs);                  {Déclenche l'interruption vidéo du BIOS }
  Colonne := Regs.dl;                   { Prélève les résultats }
  Ligne := Regs.dh;                     {   dans les registres }
  Debut := Regs.ch;                     {   et les affecte }
  Fin := Regs.cl;                       {   aux variables }
end;

{*****************************************************************}
{* SetDisplayPage : Fixe la page d'écran à afficher              *}
{* Entrée         : Voir plus bas                                *}
{* Sortie         : néant                                        *}
{*****************************************************************}

procedure SetDisplayPage(Page : integer);   { nouvelle page d'écran }

var Regs : Registers;                   { Registres pour l'interruption }

begin
  Regs.ah := 5;                         { Numéro de la fonction "Set Page" }
  Regs.al := Page;                      { Page d'écran }
  Intr($10, Regs);                  { Déclenche l'interruption vidéo du BIOS }
end;

{*****************************************************************}
{* ScrollUp : Fait défiler une zone d'écran d'une ou plusieurs   *}
{*            lignes vers le haut ou provoque son effacement     *}
{* Entrée   : Voir plus bas                                      *}
{* Sortie   : néant                                              *}
{* Info     : SI on prend 0 comme paramètre, la zone d'écran est *}
{*            remplie avec des lignes vierges                    *}
{*****************************************************************}

procedure ScrollUp(Nombre,              { Nombre de lignes à faire défiler }
                   Couleur,             { Attribut pour les lignes vierges }
                   ColonneHG,           { Colonne coin sup gauche }
                   LigneHG,             { Ligne coin sup gauche }
                   ColonneBD,           { Colonne coin inférieur droit }
                   LigneBD   : integer);    { Ligne coin inférieur droit }

var Regs : Registers;                   { Registres pour l'interruption }

begin
  Regs.ah := 6;                         { Numéro de la fonction }
  Regs.al := Nombre;                { Nombre de lignes à faire défiler }
  Regs.bh := Couleur;                   { Couleur des lignes vierges }
  Regs.ch := LigneHG;           { Fixe les coordonnées du coin sup gauche }
  Regs.cl := ColonneHG;
  Regs.dh := LigneBD;   { Fixe les coordonnées du coins inférieur droit }
  Regs.dl := ColonneBD;
  Intr($10, Regs);                  { Déclenche l'interruption vidéo du BIOS }
end;

{*****************************************************************}
{* ScrollDown : Fait défiler une zone d'écran d'une ou de plusieurs *}
{*              lignes vers le bas ou provoque son effacement    *}
{* Entrée     : Voir plus bas                                    *}
{* Sortie     : néant                                            *}
{* Info       : SI on prend 0 comme paramètre, la zone d'écran est *}
```

```
!(*              remplie avec des lignes vierges                    *)
!(*******************************************************************)

!procedure ScrollDown(Nombre,          { Nombre de lignes à faire défiler }
!          Couleur,                    { Attribut pour les lignes vierges }
!          ColonneHG,                    { Colonne coin supérieur gauche }
!          LigneHG,                      { Ligne coin supérieur gauche }
!          ColonneBD,                  { Colonne coin inférieur droit }
!          LigneBD : integer);           { Ligne coin inférieur droit }

!var Regs : Registers;                  { Registres pour l'interruption }

!begin
! Regs.ah := 7;                                { Numéro de la fonction }
! Regs.al := Nombre;                  { Nombre de lignes à faire défiler }
! Regs.bh := Couleur;                     { Couleur des lignes vierges }
! Regs.ch := LigneHG;                        { Fixe les coordonnées }
! Regs.cl := ColonneHG;                   { du coin supérieur gauche }
! Regs.dh := LigneBD;                        { Fixe les coordonnées }
! Regs.dl := ColonneBD;                   { du coin inférieur droit }
! Intr($10, Regs);                { Déclenche l'interruption vidéo du BIOS }
!end;

!(*******************************************************************)
!(* GetChar : Lit un caractère et son attribut à une position donnée *)
!(*           de la page d'écran                                    *)
!(* Entrée  : Voir plus bas                                         *)
!(* Sortie  : Voir plus bas                                         *)
!(*******************************************************************)

!procedure GetChar(Page,                          { Page d'écran concernée }
!          Colonne,                                 { Colonne d'écran }
!          Ligne      : integer;                     { Ligne d'écran }
!          var Caractere : char;                      { Caractère et }
!          var Couleur   : integer);                  { son attribut }

!var Regs     : Registers;              { Registres pour l'interruption }
!    ColonneCour,                                  { Colonne courante }
!    LigneCour,                                    { Ligne courante }
!    PageCour,                                { Page d'écran courante }
!    Dummy    : integer;       { pour variables accessoires ou inutiles }

!begin
! GetVideoMode(Dummy, Dummy, PageCour);{Détermine la page écran courante}
! GetCursorPos(PageCour, ColonneCour, LigneCour, Dummy, Dummy);
!   { lit la position courante du curseur dans la page d'écran courante }
! SetCursorPos(Page, Colonne, Ligne);            { Positionne le curseur }

! Regs.ah := 8;    { Numéro de fonction pour lire caractère et attribut }
! Regs.bh := Page;                                    { Page d'écran }
! Intr($10, Regs);              { Déclenche l'interruption vidéo du BIOS }
! Caractere := chr(Regs.al);                  { Code ASCII du caractère }
! Couleur := Regs.ah;                          { Attribut du caractère }
! SetCursorPos(PageCour, ColonneCour, LigneCour);
!end;                          { Ramène le curseur à son ancienne position }

!(*******************************************************************)
!(* WRITECHAR : Affiche un caractère dans une couleur donnée à la   *)
!(*             pos. courante du curseur dans la page écran indiquée *)
!(* Entrée    : Voir plus bas                                       *)
!(* Sortie    : néant                                               *)
!(* Info      : Dans le traitement de l'affichage les caractères de *)
!(*             contrôle comme <Entrée> sont considérés comme des   *)
!(*             caractères ordinaires                               *)
!(*******************************************************************)

!procedure WriteChar(Page    : integer;          { Page d'écran concernée }
!                    Caractere : char;         { Code ASCII du caractère }
!                    Couleur   : integer);              { son attribut }

!var Regs : Registers;                  { Registres pour l'interruption }

!begin
! Regs.ah:=9;        { Numéro de la fonction pour afficher un caractère }
! Regs.al := ord(Caractere);              { Code du caractère à afficher }
! Regs.bh := Page;                              { Fixe la page d'écran }
! Regs.bl := Couleur;                    { Fixe la couleur du caractère }
! Regs.cx := 1;                         { Un seul exemplaire à afficher }
! Intr($10, Regs);                     { Déclenche l'interruption vidéo }
!end;

!(*******************************************************************)
!(* WriteText: Affiche une chaîne de caractères à la position indiquée *)
!(*            de la page d'écran                                   *)
!(* Entrée   : Voir plus bas                                        *)
!(* Sortie   : néant                                                *)
!(* Info     : Les caractères de contrôle (par ex <Entrée>) sont    *)
!(*            traités comme tels                                   *)
!(*******************************************************************)

!procedure WriteText(Page,                        { Page d'écran concernée }
!                    Colonne,                { Colonne où débute l'affichage }
!                    Ligne,                  { Ligne, où débute l'affichage }
!                    Couleur : integer;      { Couleur pour tous caractères }
!                    Text    : TextTyp);                { Texte à afficher }

!var Regs : Registers;                  { Registres pour l'interruption }
!    Compteur  : integer;                        { Compteur d'itérations }

!begin
! SetCursorPos(Page, Colonne, Ligne);          { Positionne le curseur }
! for Compteur:=1 to length(Text) do           { Traite successivement }
!  begin                                      { les différents caractères }
!   WriteChar(Page, ' ', Couleur);          { Couleur en position courante }
!   Regs.ah := 14;                { Numéro de la fonction "sortie télétype" }
!   Regs.al := ord(Text[compteur]);       { Code du caractère à afficher }
!   Regs.bh := Page;                                   { Page d'écran }
!   Intr($10, Regs);              { Déclenche l'interruption vidéo du BIOS }
!  end;
!end;

!(*******************************************************************)
!(**                        Programme principal                    *)
!(*******************************************************************)

!begin
! ClrScr;                                              { Efface l'écran }
! for i := 1 to 24 do                          { Parcourt les lignes 1 à 24 }
!  for j := 0 to 79 do                             { et toutes les colonnes }
!   begin
!    SetCursorPos(0, j, i);                       { Positionne le curseur }
!    WriteChar(0, chr(i*80+j and 255), NORMAL);  { affiche un caractère }
!   end;
! ScrollDown(0, NORMAL, 5, 8, 19, 22);              { Efface la fenêtre 1 }
! WriteText(0, 5, 8, INVERSE, '   Fenêtre 1   ');
! ScrollDown(0, NORMAL, 60, 2, 74, 16);             { Efface la fenêtre 2 }
! WriteText(0, 60, 2, INVERSE, '   Fenêtre 2   ');
! WriteText(0, 30, 12, INVERSE or CLIGNOTANT, ' >>> BIBLE PC <<< ');
! WriteText(0, 0, 0, INVERSE, '                                    Il reste '+
!                             '   flèches à tracer                        ');
! for i := 49 downto 0 do                          { 50 flèches au total }
!  begin
!   Str(i:2, IString);                          { Transforme i en chaîne }
!   WriteText(0, 37, 0, INVERSE, IString);
!   j := 1;                          { Chaque flèche se compose de 16 lignes }
!   while j <= 15 do
!    begin
!     k := 0;
!     while k < j do                  { Fabrique une ligne de la flèche }
!      begin
!       SetCursorPos(0, 12-(j shr 1)+k, 9);          { Flèche fenêtre 1 }
!       WriteChar(0, '*', CLAIR);
!       SetCursorPos(0, 67-(j shr 1)+k, 16);         { Flèche fenêtre 2 }
!       WriteChar(0, '*', CLAIR);
!       k := succ(k);
!      end;
!     ScrollDown(1, NORMAL, 5, 9, 19, 22); { Fait défiler la fenêtre 1 }
!     ScrollUp(1, NORMAL, 60, 3, 74, 16);  { Fait défiler la fenêtre 2 }
!     for l := 0 to 8000 do                          { Boucle d'attente }
!      ;
!     j := j+2;
!    end;
!  end;
! ClrScr;                                              { Efface l'écran }
!end.
```

Listing : VIDEOC.C

```
/*****************************************************************/
/*                          V I D E O C                          */
/*---------------------------------------------------------------*/
/*    Fonction        : fournit quelques fonctions qui exploitent */
/*          l'interruption vidéo du BIOS et qui ne sont pas encore */
/* intégrées dans les bibliothèques du compilateur C de Microsoft ou */
/* de Borland                                                    */
/*---------------------------------------------------------------*/
/*    Auteur          : MICHAEL TISCHER                          */
/*    Développé le    : 13.08.1987                               */
/*    Dernière MAJ    : 13.02.1992                               */
/*---------------------------------------------------------------*/
/*    (MICROSOFT C)                                              */
/*    Compilation     : CL /AS VIDEOC.C                          */
/*    Appel           : VIDEOC                                   */
/*---------------------------------------------------------------*/
/*    (BORLAND TURBO C)                                          */
/*    Compilation     : par la commande RUN de l'EDI             */
/*                      (sans fichier de projet                  */
/*****************************************************************/

/*-- Fichiers d'inclusion ---------------------------------------*/

#include <dos.h>                            /* Fichiers d'en-tête */
#include <io.h>
#include <stdio.h>

/*-- Constantes -------------------------------------------------*/

#define NORMAL          0x07                /* Définit les attributs*/
#define CLAIR           0x0F            /* pour une carte monochrome*/
#define INVERSE         0x70
#define SOULIGNE        0x01
#define CLIGNOTANT      0x80

/*****************************************************************/
/* GETVIDEOMODE : Détermine le mode vidéo et divers autres paramètres */
/* Entrée :  Modevideo  = Pointe sur le mode vidéo courant       */
/*           NOMBRE     = Pointe sur le nombre de colonnes       */
/*           PAGE       = Pointe sur la page d'écran courante    */
/* Sortie  : néant                                               */
/*****************************************************************/

void GetVideoMode( int *ModeVideo, int *Nombre, int *Page)
{
 union REGS Register;                   /* Registres pour l'interruption */

 Register.h.ah = 15;                            /* Numéro de la fonction */
 int86(0x10, &Register, &Register);     /* Déclenche l'interruption 10h*/
 *ModeVideo = Register.h.al;                    /* Numéro du mode vidéo*/
 *Nombre = Register.h.ah;               /* Nombre de caractères par ligne */
 *Page = Register.h.bh;          /* Numéro de la page d'écran courante */
}

/*****************************************************************/
/* SETCURSORTYPE : Définit l'aspect du cursur clignotant         */
/* Entrée : Debut  = Extrémité supérieure du curseur             */
/*          Fin    = Extrémité inférieure du curseur             */
/* Sortie : néant                                                */
/* Info   : Les paramètres peuvent être compris entre 0 et 13 pour */
/*          une carte monochrome, 0 et 7 pour une carte couleur   */
/*****************************************************************/

void SetCursorType(int Debut, int Fin)
{
 union REGS Regfster;                   /* Registres pour l'interruption*/

 Register.h.ah = 1;                             /* Numéro de la fonction */
 Register.h.ch = Debut;          /* Ligne de l'extrémité sup du curseur */
 Register.h.cl = Fin;            /* Ligne de l'extrémité inf du curseur */
 int86(0x10, &Register, &Register);             /* Interruption 10(h) */
}

/*****************************************************************/
/* SETCURSORPOS : Position du curseur dans la page d'écran indiquée */
/*                                                               */
/* Entrée : PAGE    = Page d'écran                               */
/*          COLONNE = Colonne                                    */
/*          LIGNE   = Ligne                                      */
/* Sortie : néant                                                */
/* Info   : La position du curseur clignotant ne change à l'issue de */
/*          cet appel que si la page d'écran indiquée est la page */
/*          d'écran courante                                     */
/*****************************************************************/

void SetCursorPos( int Page, int Colonne, int Ligne)
{
 union REGS Register;                   /* Registres pour l'interruption*/

 Register.h.ah = 2;                             /* Numéro de la fonction*/
 Register.h.bh = Page;                                /* Page d'écran */
 Register.h.dh = Ligne;                                   /* Ligne */
 Register.h.dl = Colonne;                               /* Colonne */
 int86(0x10, &Register, &Register); /* Déclenche l'interruption 10(h) */
}

/*****************************************************************/
/* GETCURSORPOS : Détermine la position courante du curseur dans la */
/*                page d'écran ainsi que les limites de son aspect */
/* Entrée :  PAGE    = Page d'écran                              */
/*           COLONNE = Pointe sur la colonne courante            */
/*           LIGNE   = Pointe sur la ligne courante              */
/*           DEBUT   = Pointe sur la ligne de trame représentant */
/*                     l'extrémité supérieure du curseur.        */
/*           FIN     = Pointe sur la ligne de trame représentant */
/*                     l'extrémité inférieure du curseur         */
/* Sortie : néant                                                */
/*****************************************************************/

void GetCursorPos( int Page, int *Colonne,
                   int *Ligne, int *Debut, int *Fin)
{
 union REGS Register;                   /* Registres pour l'interruption*/

 Register.h.ah = 3;                             /* Numéro de la fonction */
 Register.h.bh = Page;                                /* Page d'écran */
 int86(0x10, &Register, &Register); /* Déclenche l'interruption 10(h) */
 *Colonne = Register.h.dl;                      /* Prélève les résultats */
 *Ligne  = Register.h.dh;                         /* dans les registres */
 *Debut = Register.h.ch;                            /* et les affecte */
 *Fin   = Register.h.cl;                            /* aux variables*/
}

/*****************************************************************/
/* SETDISPLAYPAGE: Affiche la page d'écran demandée              */
/* Entrée : PAGE = Page d'écran                                  */
/* Sortie : néant                                                */
/*****************************************************************/

void SetDisplayPage( int Page )
{
 union REGS Register;                   /* Registres pour l'interruption */

 Register.h.ah = 5;                             /* Numéro de la fonction */
 Register.h.al = Page;                                /* Page d'écran */
 int86(0x10, &Register, &Register); /* Déclenche l'interruption 10(h) */
}

/*****************************************************************/
/* SCROLLUP : Fait défiler une zone d'écran d'une ou plusieurs   */
/*            lignes vers le haut ou provoque son effacement     */
/* Entrée  : NOMBRE    = Nombre de lignes à faire défiler        */
/*           COULEUR   = Couleur ou attribut des lignes vierges  */
/*           COLONNEHG = Colonne du coin sup gauche de la zone   */
/*           LIGNEHG   = Ligne du coin sup gauche de la zone     */
/*           COLONNEBD = Colonne du coin inférieur droit de la zone */
/*           LIGNEBD   = Ligne du coin inférieur droit de la zone */
/* Sortie  : néant                                               */
/* Info    : Si on prend 0 comme paramètre, la zone d'écran      */
/*           est remplie avec des lignes vierges                 */
/*****************************************************************/

void ScrollUp( int Nombre, int Couleur, int ColonneHG,
               int LigneHG, int ColonneBD, int LigneBD )
{
 union REGS Register;                   /* Registres pour l'interruption */

 Register.h.ah = 6;                             /* Numéro de la fonction */
 Register.h.al = Nombre;                          /* Nombre de lignes */
 Register.h.bh = Couleur;                /* Couleur des lignes vierges*/
 Register.h.ch = LigneHG;                  /* Fixe les coordonnées de */
 Register.h.cl = ColonneHG;                                /* la fenêtre */
 Register.h.dh = LigneBD;                                /* de défilement */
 Register.h.dl = ColonneBD;
 int86(0x10, &Register, &Register); /* Déclenche l'interruption 10(h) */
}

/*****************************************************************/
/* SCROLLDOWN : Fait défiler une zone d'écran d'une ou de plusieurs */
/*              lignes vers le bas ou provoque son effacement    */
/* Entrée  : NOMBRE    = Nombre de lignes à faire défiler        */
/*           COULEUR   = Couleur ou attribut des lignes vierges  */
/*           COLONNEHG = Colonne du coin sup gauche de la zone   */
/*           LIGNEHG   = Ligne du coin sup gauche de la zone     */
/*           COLONNEBD = Colonne du coin inf droit de la zone    */
```

```
/*          LIGNEBD  = Ligne du coin inf droit de la zone          */
/* Sortie : néant                                                  */
/* Info   : Si le paramètre est 0, remplie de lignes vierges      */
/*****************************************************************/

void ScrollDown( int Nombre, int Couleur, int ColonneHG,
                 int LigneHG, int ColonneBD, int LigneBD )
{
 union REGS Register;                    /* Registres pour l'interruption */

 Register.h.ah = 7;                          /* Numéro de la fonction */
 Register.h.al = Nombre;                      /* Nombre de lignes */
 Register.h.bh = Couleur;                /* Couleur des lignes vierges */
 Register.h.ch = LigneHG;                /* Fixe les coordonnées de la */
 Register.h.cl = ColonneHG;                  /* fenêtre de défilement */
 Register.h.dh = LigneBD;
 Register.h.dl = ColonneBD;
 int86(0x10, &Register, &Register); /* Déclenche l'interruption 10(h) */
}

/*****************************************************************/
/* GETCHAR : lit un caractère et son attribut à une position donnée  */
/*           de la page d'écran                                      */
/* Entrée :  PAGE      = Page d'écran concernée                      */
/*           COLONNE   = Colonne du caractère                        */
/*           LIGNE     = Ligne du caractère                          */
/*           Caractere = Pointe sur le caractère traité              */
/*           COULEUR   = Pointe sur l'attribut (ou la couleur)       */
/* Sortie : néant                                                    */
/*****************************************************************/

void GetChar( int Page, int Colonne, int Ligne,
              char *Caractere, int *Couleur)
{
 union REGS Register;                    /* Registres pour l'interruption*/
 int Dummy;                      /* Pour variables accessoires ou inutiles*/
 int PageCour;                           /* page d'écran courante */
 int LigneCour;                                /* Ligne courante */
 int ColonneCour;                             /* Colonne courante */

 GetVideoMode(&Dummy, &Dummy, &PageCour);    /* Page d'écran courante */
 GetCursorPos( PageCour, &ColonneCour, &LigneCour,   /* Pos courante  */
               &Dummy, &Dummy);                          /* du curseur */
 SetCursorPos(Page, Colonne, Ligne);          /* Positionne le curseur */
 Register.h.ah = 8;                            /* Numéro de la fonction*/
 Register.h.bh = Page;                                /* Page d'écran */
 Int86(0x10, &Register, &Register); /* déclenche l'interruption 10(h) */
 *Caractere = Register.h.al;  /* Lit les résultats dans les registres */
 *Couleur = Register.h.ah;           /* et les affecte aux variables */
 SetCursorPos(PageCour, ColonneCour, LigneCour); /* Curseur anc. pos. */
}

/*****************************************************************/
/* WRITECHAR : Affiche un caractère avec son attribut              */
/*             à une position donnée de la page d'écran indiquée   */
/* Entrée :    PAGE      = Page d'écran concernée                  */
/*             CARACTERE = Caractère à afficher                    */
/*             COULEUR   = Attribut ou couleur du caractère        */
/* Sortie : néant                                                  */
/*****************************************************************/

void WriteChar( int Page, char Caractere, int Couleur)
{
 union REGS Register;                    /* Registres pour l'interruption */

 Register.h.ah = 9;                          /* Numéro de la fonction */
 Register.h.al = Caractere;                   /* Caractère à afficher */
 Register.h.bh = Page;                               /* Page d'écran */
 Register.h.bl = Couleur;            /* Couleur du caractère à afficher */
 Register.x.cx = 1;                             /* Un seul exemplaire */
 int86(0x10, &Register, &Register); /* déclenche l'interruption 10(h) */
}

/*****************************************************************/
/* WRITETEXT : Affiche une chaîne de caractère d' une certaine couleur*/
/*             à une position donnée de la page d'écran indiquée   */
/* Entrée  :   PAGE    = page d'écran concernée                    */
/*             COLONNE = Colonne où débute la chaîne               */
/*             LIGNE   = Ligne où débute la chaîne                 */
/*             COULEUR = Attribut ou couleur des caractères        */
/*             TEXT    = Pointeur sur la chaîne                    */
/* Sortie  : néant                                                 */
/* Info    : Text pointe sur un tableau de caractères qui contient le */
/*           texte à afficher et se termine par le caractère nul '\0' */
/*****************************************************************/

void WriteText( int Page, int Colonne, int Ligne, int Couleur,
                char *Text)
{
 union REGS InRegister,              /* Registres pour l'interruption */
            OutRegister;

 SetCursorPos(Page, Colonne, Ligne);          /* Positionne le curseur */

 InRegister.h.ah = 14;                        /* Numéro de la fonction */
 InRegister.h.bh = Page;                                /* Page d'écran */
 while (*Text)                       /* Affiche le texte jusqu'à '\0' */
  {
   WriteChar(Page, ' ', Couleur);              /* Couleur du caractère */
   InRegister.h.al = *Text++;                             /* Caractère */
   int86(0x10, &InRegister, &OutRegister);            /* Interruption */
  }
}

/*****************************************************************/
/* CLEARSCREEN : Efface les 80*25 caractères de l'écran et         */
/*               positionne le curseur en haut à gauche            */
/* Entrée : néant                                                  */
/* Sortie : néant                                                  */
/*****************************************************************/

void ClearScreen( void )
{
 int PageCour;                                 /* Page d'écran courante */
 int Dummy;                                      /* Variable fantôme */

 ScrollUp(0, NORMAL, 0, 0, 79, 24);           /* Effacer tout l'écran */
 GetVideoMode(&Dummy, &Dummy, &PageCour);    /* Page d'écran courante */
 SetCursorPos(PageCour, 0, 0);               /* Positionne le curseur */
}

/*****************************************************************/
/**                    PROGRAMME PRINCIPAL                        **/
/*****************************************************************/

void main()
{
 int i, j, k, l;                          /* Variables d'itération */
 char Fleche[3];                  /* Nombre de flèches en format ASCII*/

 ClearScreen();                                    /* Efface l'écran */
 for (i = 1; i < 25; i++)                /* Parcourt toutes les lignes */
  for (j = 0; j < 80; j++)                    /* et toutes les colonnes */
   {
    SetCursorPos(0, j, 1);                  /* Positionne le curseur */
    WriteChar(0, i*80+j&255, NORMAL);        /* Affiche un caractère */
   }
 ScrollDown(0, NORMAL, 5, 8, 19, 22);          /* Efface la fenêtre 1 */
 WriteText(0, 5, 8, INVERSE, "  Fenêtre 1   ");
 ScrollDown(0, NORMAL, 60, 2, 74, 16);         /* Efface la fenêtre 2 */
 WriteText(0, 60, 2, INVERSE, "  Fenêtre 2  ");
 WriteText(0, 30, 12, INVERSE | CLIGNOTANT, " >>>LA BIBLE PC<<< ");
 WriteText(0, 0, 0, INVERSE, "                    Il reste ");
 WriteText(0, 40, 0, INVERSE,"flèches à tracer");
 for (i = 49; i >= 0 ; i--)                          /* Trace 50 flèches */
  {
   sprintf(Fleche, "%2d", i); /* Conversion du nombre en chaîne ASCII */
   WriteText(0, 37, 0, INVERSE, Fleche);          /* affiche le nombre */
   for (j = 1; j < 16; j+= 2)          /* Flèche composée de 16 lignes */
    {
     for (k = 0; k < j; k++)          /* Fabrique une ligne de la flèche */
      {
       SetCursorPos(0, 12-(j>>1)+k, 9);          /* Flèche fenêtre 1 */
       WriteChar(0, '*', CLAIR);
       SetCursorPos(0, 67-(j>>1)+k, 16);         /* Flèche fenêtre 2 */
       WriteChar(0, '*', CLAIR);
      }
     ScrollDown(1, NORMAL, 5, 9, 19, 22); /* Fait défiler la fenêtre 1*/
     ScrollUp(1, NORMAL, 60, 3, 74, 16);  /* Fait défiler la fenêtre 2*/
     for (l = 0; l < 4000 ; l++)                  /* Boucle d'attente */
      ;
    }
  }
 ClearScreen();                                    /* Efface l'écran */
}
```

4.3. Déterminer quelle carte vidéo est installée

Chaque fois que vous voulez accéder directement à l'électronique d'une carte vidéo bien précise ou à des fonctions BIOS particulières, qui ne figurent que dans des versions spéciales du BIOS, comme le BIOS EGA par exemple, vous devez vous assurer auparavant que la carte appelée soit effectivement installée. A défaut de ce test, il peut arriver que vous écriviez des caractères et des caractères dans la RAM vidéo mais que l'écran reste noir parce que le PC dispose d'une carte CGA, par exemple, et non d'une carte MDA comme vous le supposiez.

C'est bien sûr surtout si votre programme est censé travailler indifféremment avec les différentes sortes de carte vidéo et accéder malgré cela directement à l'électronique de ces cartes, que la connaissance du type de carte vidéo installée sera d'une importance cruciale. Sans ces informations en effet, les routines de sortie ne pourraient s'adapter aux particularités et aux paramètres spécifiques de chaque carte. Dans ce chapitre, vous apprendrez à déterminer :

- ✓ le type de la carte vidéo installée pendant l'exécution d'un programme,
- ✓ les différentes cartes vidéo utilisables simultanément dans un PC.

Utilisation simultanée de cartes vidéo multiples

Il convient, à cet égard, de tenir également compte du fait que le PC peut disposer en même temps d'une carte monochrome (MDA, HGC, mais aussi EGA sur un moniteur monochrome) et d'une carte couleur (EGA, VGA ou CGA). Dans ce cas, une seule des deux cartes peut être activée à un moment donné.

Combinaisons possibles des cartes vidéo PC

	VGA	EGA	HGC	CGA	MDA
VGA			■		■
EGA			■	■	■
HGC	■	■		■	
CGA		■	■		
MDA	■	■		■	■

Il s'agit donc de déterminer le type de la carte vidéo installée. On ne peut malheureusement avoir recours pour cela ni à un appel de fonction BIOS ou DOS, ni au test d'une variable. Il faut donc développer à cet effet une routine assembleur qui examinera chaque fois, à l'aide de tests appropriés, quel type de carte vidéo est disponible. Cette routine pourra s'appuyer sur les indications des fabricants de matériel électronique car presque

chaque fabricant de carte vidéo indique, dans la documentation technique fournie avec sa carte, un procédé permettant de détecter la présence de cette carte sur un système. Il est naturellement très important que le test soit parfaitement fiable, c'est-à-dire qu'il ne fournisse un résultat positif que pour un type bien précis de carte vidéo. Les cartes EGA et VGA posent cependant des problèmes de ce point de vue car suivant le moniteur connecté elles peuvent émuler une carte CGA ou MDA et il est difficile de les distinguer de ces cartes.

Tests pour déterminer la carte vidéo active

Tous les tests décrits dans les pages suivantes sont repris à la fin de cette section dans deux petits programmes assembleur qui ont été conçus pour être intégrés dans des programmes en C et en Pascal. Ils inscrivent le type de carte vidéo connectée et le type du moniteur connecté dans un tableau (Array) dont l'adresse leur a été transmise par la fonction d'appel. Si deux cartes vidéo sont installées, l'ordre dans lequel elles sont citées dans le vecteur indique laquelle des deux est activée pour le moment.

La routine assembleur identifie les cartes suivantes :

- ✓ cartes MDA,
- ✓ cartes CGA,
- ✓ cartes HGC,
- ✓ cartes EGA et
- ✓ cartes VGA (y compris Super VGA)

Comme la routine assembleur teste successivement l'existence de chaque carte vidéo, le listing assembleur comporte un sous-programme particulier pour chaque type de carte vidéo. Ce sous-programme porte chaque fois le nom de la carte vidéo qu'il est chargé de détecter. Ces routines s'appellent donc TEST_EGA, TEST_VGA etc. Les différents tests sont appelés successivement mais certains tests peuvent être exclus lorsqu'il est certain qu'ils fourniraient un résultat erroné. C'est le cas par exemple pour le test CGA si une carte EGA ou VGA reliée à un moniteur hauteur résolution ou couleur a été détectée auparavant. Une carte CGA ne peut en effet être installée à côté d'une carte de ce type mais le test CGA annoncerait la présence d'une carte CGA alors qu'il ne pourrait en fait s'agir que de la carte EGA ou VGA déjà détectée.

Pour éviter les inexactitudes, il existe pour chaque test un flag décidant si le test doit être ou non exécuté. Avant le premier test, tous les flags sont fixés sur 1 pour que chaque test soit exécuté lorsque ce sera son tour. Au cours des tests, certains flags pourront être fixés sur 0, pour les raisons que nous venons d'indiquer, afin que la routine de test correspondante ne soit pas appelée.

C'est le test VGA qui ouvre la marche. Il est très simple à réaliser puisqu'une fonction spéciale du BIOS VGA, la sous-fonction 00h de la fonction 1Ah, permet d'obtenir exactement les informations devant être renvoyées par la routine assembleur. Ces

informations ne sont toutefois disponibles, après appel de la fonction, que si une carte VGA et donc aussi le BIOS VGA sont installés. C'est le cas si la valeur 1Ah figure dans le registre AL après appel de la fonction. Si la routine de test trouve une autre valeur, le test est interrompu avec un résultat négatif et les autres tests se poursuivent. Dans ce cas, c'est qu'il n'y a pas de carte VGA installée.

Après un appel positif de cette fonction, un code de périphérique spécial désigne dans le registre BL la carte vidéo activée et dans le registre BH celle qui n'est pas activée. Les codes suivants peuvent se présenter :

Codes renvoyés après appel de la sous-fonction 00h de la fonction 1Ah du BIOS VGA

Code	Signification
00h	Pas de carte vidéo
01h	Carte MDA sur un moniteur monochrome
02h	Carte CGA sur un moniteur couleur
03h	Réservé
04h	Carte EGA sur un moniteur haute résol.
05h	Carte EGA sur un moniteur monochrome
06h	Réservé
07h	Carte VGA sur un moniteur monochrome analogue
08h	Carte VGA sur un moniteur couleur analogue

Ces codes sont séparés par la routine de test VGA en valeurs pour la carte vidéo et en valeurs pour le moniteur connecté. Elles sont alors chargées dans le vecteur dont l'adresse a été transmise à la routine assembleur par celui qui l'a appelée. Comme cette routine renvoie déjà des informations sur les deux cartes vidéo, il n'est plus nécessaire d'effectuer les tests suivants. Seul le test mono doit encore être effectué si la fonction a détecté la présence d'une carte monochrome car elle ne sait pas distinguer une carte MDA d'une carte HGC.

Dans la routine principale du programme assembleur, le test VGA est suivi du test EGA, qui n'est effectué que si le résultat du test VGA a été négatif et si le flag EGA n'a pas été fixé de ce fait sur 0. Ce test a également recours à une fonction qui ne figure que dans le BIOS EGA : la sous-fonction 10h de la fonction 12h. S'il n'y a pas de carte EGA installée et si, par conséquent, cette fonction n'est pas disponible, la valeur 10h figurera toujours après appel de la fonction dans le registre BL, où elle avait été placée avant appel de la fonction. Dans ce cas, le test EGA est terminé.

En cas de succès, le registre CL contient après appel de la fonction la position des commutateurs DIP de la carte EGA. Ces commutateurs indiquent le type de moniteur connecté. Ils sont convertis en codes moniteur de la routine assembleur, puis inscrits

dans le vecteur avec le code de la carte EGA. Suivant le type de moniteur connecté, le flag CGA ou mono est fixé sur 0, le test correspondant n'ayant plus à être effectué après l'identification d'une carte EGA. La routine EGA est alors terminée.

Si le flag CGA n'a pas été fixé sur 0 à la suite de l'un des tests précédents, le test CGA fait suite au test EGA. De même que le test mono, il ne peut avoir recours à aucune fonction du BIOS et doit tester directement si l'électronique correspondante est présente. Dans les deux routines, cela s'effectue en appelant la routine TEST_6845, qui examine si le contrôleur vidéo 6845 de la carte vidéo considérée se trouve à l'adresse de port spécifiée. Pour la carte CGA, c'est l'adresse 3D4h qui est communiquée à la routine TEST_6845.

Le seul moyen de détecter la présence du CRTC à une adresse de port déterminée consiste à écrire une valeur quelconque (différente de 0) dans l'un des registres du CRTC et de la relire immédiatement. Si la valeur relue correspond à la valeur écrite, c'est que le CRTC, et donc aussi la carte vidéo, figurent bien à l'adresse de port spécifiée. Avant d'écrire toutefois n'importe quelle valeur dans l'un des registres du CRTC, il convient de ne pas oublier que ces registres jouent un rôle fondamental dans la mise en place des signaux vidéo. Un accès inconsidéré pourrait donc non seulement perturber gravement l'activité du CRTC, mais même, dans le pire des cas, endommager le moniteur. Les registres 0 à 9 sont donc exclus pour ce type de tests. Restent les registres 10 à 15 dont la modification peut malgré tout également affecter le contenu de l'écran. Les modifications les plus bénignes sont cependant celles occasionnées par les registres 10 et 11, qui fixent les lignes de départ et de fin du curseur clignotant de l'écran.

La routine assembleur lit donc tout d'abord le contenu du registre 10 avant d'écrire dans ce registre une valeur choisie arbitrairement. Après une courte pause, pendant laquelle le CRTC peut réagir à la sortie, le contenu de ce registre est à nouveau lu. Avant qu'il ne soit cependant comparé à la valeur écrite, l'ancienne valeur est réécrite dans le registre pour que ce test ne se traduise pas par des changements durables sur l'écran. S'il résulte de la comparaison qui s'ensuit que la valeur écrite est identique à la valeur qui a été relue plus tard, c'est qu'il y a un CRTC et donc aussi une carte vidéo (dans ce cas, une CGA). La routine CGA réagit alors en chargeant le code correspondant dans le vecteur et en signalant comme moniteur le moniteur couleur, le seul moniteur pouvant être exploité avec une carte CGA.

Le dernier test appelé est le test mono, qui recherche également dans un premier temps s'il y a un CRTC 6845, cette fois à l'adresse 3B4h. S'il trouve effectivement un CRTC à cet endroit, c'est nécessairement qu'une carte monochrome est installée. Il reste cependant encore à déterminer s'il s'agit d'une MDA ou d'une HGC. C'est à l'aide du registre d'état des deux cartes vidéo, auquel on accède à travers le port 3BAh, qu'on peut répondre à cette question. Le bit 7 de ce registre ne joue en effet aucun rôle sur la carte MDA, et sa valeur est donc indéterminée, alors que sur la carte HGC il vaut 1 chaque fois que le rayon électronique du tube cathodique se trouve en phase de retour vertical à travers l'écran. Comme il ne s'agit bien sûr pas d'un état permanent mais d'une situation qui se reproduit environ toutes les 2 millisecondes, le contenu de ce bit passe sans cesse de 0 à 1 et inversement.

La routine de test tire parti de ce changement, qui n'a pas lieu sur la carte MDA. Elle lit tout d'abord le contenu de ce registre et masque les bits 0 à 6. Elle enregistre le résultat obtenu comme valeur de comparaison pour les 32768 parcours de boucle, au maximum, au cours desquels la valeur du registre d'état est sans cesse relue. Si une modification est constatée par rapport à la valeur de référence, c'est-à-dire si l'état du bit 7 a changé, c'est qu'il s'agit d'une carte HGC. Si par contre l'état de ce bit ne change pas pendant les 32768 parcours de la boucle, c'est qu'on a affaire à une carte MDA.

Ici aussi, le code correspondant à la carte vidéo identifiée est chargé dans le vecteur. C'est systématiquement le code du moniteur monochrome qui est indiqué comme code de moniteur car c'est le seul type de moniteur pouvant être connecté sur les cartes MDA et HGC.

Les tests sont alors terminés pour l'essentiel, il ne reste plus qu'à déterminer quelle est la carte vidéo activée (primaire) et la carte non activée (secondaire). Si l'appel du test VGA a été positif, ce test n'a toutefois pas lieu d'être puisque la routine de test correspondante se charge automatiquement de distinguer entre la carte vidéo activée et la carte non activée.

Dans tous les autres cas, la carte vidéo activée doit être déterminée à l'aide du mode vidéo actuel, qui peut être obtenu à l'aide de la fonction 0Fh de l'interruption vidéo du BIOS. Si on obtient la valeur 7 comme résultat, c'est le mode de texte 80*25 caractères de la carte monochrome qui est activé. Tout autre mode implique que c'est une carte CGA, EGA ou VGA qui est activée. La dernière tâche de la routine assembleur consiste donc, sur la base de ces informations, à changer l'ordre des deux entrées, si ce n'est pas l'entrée de la carte activée qui figure déjà en premier.

La routine assembleur a alors terminé son travail et elle rend la main à la fonction qui l'avait appelée.

Après ces explications détaillées, vous ne devriez pas avoir trop de difficulté à imaginer le fonctionnement des deux programmes assembleur que voici. Chacun est précédé d'un programme en langage évolué qui appelle la fonction GetVIOS à partir du module assembleur, pour en montrer le mode d'utilisation.

Listing : VIOSC.C

```
/*****************************************************************/
/*                          V I O S C                            */
/*---------------------------------------------------------------*/
/*   Fonction       : Détermine le type de cartes vidéo          */
/*                    installées et les moniteurs connectés      */
/*---------------------------------------------------------------*/
/*   Auteur         : MICHAEL TISCHER                            */
/*   Développé le   :  2/10/1988                                 */
/*   Dernière modif.:  14/01/1991                                */
/*---------------------------------------------------------------*/
/*   (MICROSOFT C)                                               */
/*   Création       : CL /AS /c VIOSC.C                          */
/*                    LINK VIOSCA VIOSC                          */
/*   Appel          : VIOSC                                      */
/*---------------------------------------------------------------*/
/*   (BORLAND TURBO C)                                           */
/*   Création       : Avec un fichier Project de teneur suivante :*/
/*                    VIOSC                                      */
/*                    VIOSCA.OBJ                                 */
/*****************************************************************/

/*-- Déclaration de fonctions externes ------------------------*/

extern void get_vios( struct vios * );

/*-- Typedefs -------------------------------------------------*/

typedef unsigned char BYTE;          /* Nous nous bricolons un OCTET */

/*-- Structures -----------------------------------------------*/

struct vios {       /* Décrit une carte vidéo et le moniteur connecté */
            BYTE carte,
                 moniteur;
            };

/*-- Constantes -----------------------------------------------*/

/*-- Constantes pour la carte vidéo ---------------------------*/

#define NO_VIOS   0                              /* Pas de carte vidéo */
#define VGA       1                                    /* Carte VGA */
#define EGA       2                                    /* Carte EGA */
#define MDA       3                  /* Monochrome Display Adapter */
#define HGC       4                    /* Hercules Graphics Card */
#define CGA       5                          /* Color Graphics Adapter */

/*-- Constantes pour le type de moniteur ----------------------*/

#define NO_MON    0                                /* Pas de moniteur */
#define MONO      1                              /* Moniteur monochrome */
#define COLOR     2                                /* Moniteur couleur */
#define EGA_HIRES 3                          /* Moniteur haute résolution */
#define ANAL_MONO 4                        /* Moniteur analogue monochrome */
#define ANAL_COLOR 5                        /* Moniteur couleur analogique */

/*****************************************************************/
/**                     PROGRAMME PRINCIPAL                     **/
/*****************************************************************/

void main()
{
 static char *namscvi[] = {/* Pointeur sur les noms des cartes vidéo */
                "VGA",
                "EGA",
                "MDA",
                "HGC",
                "CGA"
               };

 static char *namsmoni[] = {
                /* Pointeur sur les noms des types de moniteur */
                 "Moniteur monochrome",
                 "Moniteur couleur",
                 "Moniteur haute résolution",
                 "Moniteur monochrome analogique",
                 "Moniteur couleur analogique"
                };

 struct vios vsys[2];                        /* Vecteur pour GET_VIOS */

 get_vios( vsys );                        /* Déterminer systèmes vidéo */
 printf("\nVIOS (c) 1988 by Michael Tischer\n\n");
 printf("Système vidéo primaire :   carte %s sur un %s\n",
         namscvi[vsys[0].carte-1], namsmoni[vsys[0].moniteur-1]);
 if ( vsys[1].carte != NO_VIOS )     /* système vidéo secondaire ? */
   printf("Système vidéo secondaire : carte %s sur un %s\n",
           namscvi[vsys[1].carte-1], namsmoni[vsys[1].moniteur-1]);
}
```

Listing : VIOSCA.ASM

```
;*****************************************************************;
;*                          V I O S C A                          *;
;*---------------------------------------------------------------*;
;*   Fonction       : Fournit une fonction à intégrer dans un    *;
;*                    programme C et servant à déterminer le type*;
;*                    des deux adaptateurs vidéo installés.      *;
;*---------------------------------------------------------------*;
;*   Auteur         : MICHAEL TISCHER                            *;
;*   Développé le   : 02.10.1988                                 *;
;*   Dernière MAJ   : 14.02.1992                                 *;
;*---------------------------------------------------------------*;
;*   Assemblage     : MASM VIOSCA;                               *;
;*                    ... puis édition de liens avec VIOSC.C     *;
;*****************************************************************;

;-- Constantes pour la structure VIOS ------------------------

;                                   ;-- Constantes pour la carte vidéo
NO_VIOS   = 0                                  ; Pas de carte vidéo
VGA       = 1                                  ; Carte VGA
EGA       = 2                                  ; Carte EGA
MDA       = 3                      ; Monochrome Display Adapter
HGC       = 4                        ; Hercules Graphics Card
CGA       = 5                        ; Color Graphics Adapter
                                  ; Constantes pour le moniteur
NO_MON    = 0                             ; Pas de moniteur
MONO      = 1                           ; Moniteur monochrome
COLOR     = 2                             ; Moniteur couleur
EGA_HIRES = 3                ; Moniteur haute résolution ou Multisync
ANAL_MONO = 4                   ; Moniteur analogique monochrome
ANAL_COLOR = 5                   ; Moniteur couleur analogique
; -- Déclarations de segment pour le programme C ---------------

IGROUP group _text                  ; Regroupement segments de programme
DGROUP group const,_bss, _data      ; Regroupement segments de données
       assume CS:IGROUP, DS:DGROUP, ES:DGROUP, SS:DGROUP

CONST  segment word public 'CONST'              ; Segment des constantes
CONST  ends                                        ; pour lecture seule

_BSS   segment word public 'BSS'                ; Segment des variables
_BSS   ends                                   ; statiques non initialisées

_DATA  segment word public 'DATA'                   ; Segment de données

vios_tab   equ this byte

      ;-- table de conversion traitant le résultat retourné par l'option
                                        ;-- 00h de la fonction 1Ah du BIOS VGA----
         db NO_VIOS, NO_MON                          ; Pas de carte vidéo
         db MDA    , MONO          ; Carte MDA sur moniteur monochrome
         db CGA    , COLOR           ; Carte CGA sur moniteur couleur
         db ?      , ?                          ; Code 3 non utilisé
         db EGA    , EGA_HIRES                ; Carte EGA/moniteur HR
         db EGA    , MONO          ; Carte EGA sur moniteur monochrome
         db ?      , ?                          ; Code 6 non utilisé
         db VGA    , ANAL_MONO    ; Carte VGA/moniteur mono analogique
         db VGA    , ANAL_COLOR    : Carte VGA/moniteur couleur analog

ega_dips   equ this byte

           ;-- Table de conversion pour traiter les micro-commutateurs
                                                  ;-- DIP de la carte EGA
```

```
;
;       db COLOR, EGA_HIRES, MONO
;       db COLOR, EGA_HIRES, MONO
;
_DATA   ends
;
; == Programme ================================================
;
_TEXT   segment byte public 'CODE'              ; Segment de programme
;
public  _get_vios
;
;;-----------------------------------------------------------------
;;-- GET_VIOS: Détermine le type des cartes vidéo installées ------
;;-- Appel en C  : void get_vios( struct vios *vp );
;;-- Déclaration : struct vios { BYTE carte, monitor; };
;;-- Valeur retournée : Aucune
;;-- Infos       : Dans cet exemple, la fonction est conçue pour être
;;--               intégrée dans le modèle SMALL
;
;
_get_vios proc near
;
sframe  struc                           ; Structure pour accéder à la pile
cga_possi db ?                                  ; Variable locale
ega_possi db ?                                  ;       "
mono_possi db ?                                 ;       "
bptr    dw ?                                    ; Reçoit BP
ret_adr dw ?                            ; Adresse de retour à l'appelant
vp      dw ?                            ; Pointeur sur la 1ère structure VIOS
sframe  ends                                    ; Fin de la structure
;
frame   equ [ bp - cga_possi ]                  ; Eléments de la structure
;
        push bp                                 ; Sauvegarde BP sur la pile
        sub  sp,3                       ; Place pour les variables locales
        mov  bp,sp                              ; Transfère SP dans BP
        push di                                 ; Sauve DI sur la pile
;
        mov  frame.cga_possi,1                          ; Peut être CGA
        mov  frame.ega_possi,1                          ; Peut être EGA
        mov  frame.mono_possi,1                 ; Peut être MDA ou HGC
;
        mov  di,frame.vp        ; Recherche l'offset de la structure
        mov  word ptr [di],NO_VIOS              ; Aucun système vidéo
        mov  word ptr [di+2],NO_VIOS            ; trouvé pour le moment
;
        call test_vga                           ; Teste si carte VGA
        cmp  frame.ega_possi,0          ; Carte EGA encore possible ?
        je   gv1                                ; NON --> passer au test CGA
;
        call test_ega                           ; Teste si carte EGA
gv1:    cmp  frame.cga_possi,0          ; Carte CGA encore possible ?
        je   gv2                        ; NON --> passe au test MDA et HGC
        call test_cga                           ; Teste si carte CGA
gv2:    cmp  frame.mono_possi,0  ; Carte MDA ou HGC encore possible ?
        je   gv3                                ; NON --> Tests terminés
;
        call test_mono                  ; Teste si carte MDA et HGC
;
;
        ;-- Détermine la carte vidéo active ----------------------
;
gv3:    cmp  byte ptr [di].VGA                  ; Carte VGA identifiée ?
        je   gvi_end                    ; OUI, carte active déjà identifiée
        cmp  byte ptr [di+2].VGA        ; Carte VGA= système secondaire
        je   gvi_end                    ; OUI, carte active déjà identifiée
;
        mov  ah,0Fh             ; Détermine le mode vidéo courant par
        int  10h                                ; l'interruption vidéo
;
        and  al,7               ; Seuls les modes 0 à 7 sont intéressants
        cmp  al,7                       ; Carte monochrome active ?
        jne  gv4                        ; Non, en mode CGA ou EGA
        ;-- La carte MDA, HGC ou EGA (mono) est active ------------
;
        cmp  byte ptr [di+1].MONO               ; Moniteur mono
        je   gvi_end                            ; OUI, ordre correct
        jmp  short echange                      ; NON, intervertir ordre
;
        ;-- Une carte CGA ou EGA est active ----------------------
;
gv4:    cmp  byte ptr [di+1],MONO   ; Moniteur mono ds 1ère structure
        jne  gvi_end                            ; NON, ordre correct
;
echange: mov  ax,[di]                   ; contenu de la première structure
        xchg ax,[di+2]                  ; Echange avec seconde structure
        mov  [di],ax
;
gvi_end: pop  di                                ; Retire DI de la pile
        add  sp,3               ; Elimine les variables locales de la pile
        pop  bp                                 ; Retire BP de la pile
        ret                                     ; Retour au programme C
;
_get_vios endp
;
;;-----------------------------------------------------------------
;;-- TEST_VGA: Détermine si une carte VGA est installée
;
test_vga proc near
;
        mov  ax,1a00h       ; Appelle l'option 00h de la fonction 1Ah
        int  10h                                        ; du BIOS
        cmp  al,1ah                             ; Fonction supportée ?
        jne  tvga_end                   ; NON --> Termine la routine
;
        ;-- La fonction est supportée , BL contient maintenant le code
        ;-- du système vidéo actif, BH celui du système non actif-----
;
        mov  cx,bx                              ; Range le résultat dans CX
        xor  bh,bh                                      ; Met BH à 0
        or   ch,ch                              ; Un seul système vidéo ?
        je   tvga_1                     ; OUI --> Code du premier système
;
        ;-- Convertit le code du second système ----------------------
;
        mov  bl,ch                              ; Code du second système en BL
        add  bl,bl                      ; Calcule l'offset dans la table
        mov  ax,offset DGROUP:vios_tab[bx]      ; Code dans la table
        mov  [di+2],ax          ; stocke dans la structure appelante
        mov  bl,cl      ; Remet dans BL les codes du premier système
;
        ;-- Convertit le code du premier système ----------------------
;
tvga_1: add  bl,bl                      ; Calcule l'offset dans la table
        mov  ax,offset DGROUP:vios_tab[bx]      ; Code dans la table
        mov  [di],ax                    ; stocke dans la structure appelante
;
        mov  frame.cga_possi,0  ; Le test CGA n'a plus lieu d'être
        mov  frame.ega_possi,0                  ; Le test EGA non plus
        mov  frame.mono_possi,0                 ; Reste à tester MONO
;
        mov  bx,di                      ; Adresse de la structure active
        cmp  byte ptr [bx],MDA  ; Système monochrome identifié ?
        je   do_tmono                   ; OUI --> Effectue test MDA/HGC
;
        add  bx,2                       ; Adresse de la structure inactive
        cmp  byte ptr [bx],MDA  ; Système monochrome identifié ?
        jne  tvga_end                           ; NON termine la routine
;
do_tmono: mov  word ptr [bx],0  ; Simule la non-identification du
                                                        ; système
        mov  frame.mono_possi,1         ; Exécute le test monochrome
;
tvga_end: ret                           ; Retour au programme appelant
;
test_vga endp
;
;;-----------------------------------------------------------------
;;-- TEST_EGA: Détermine si une carte EGA est installée
;
test_ega proc near
;
        mov  ah,12h                             ; Fonction 12h
        mov  bl,10h                             ; Option 10h
        int  10h                        ; Appelle le BIOS EGA
        cmp  bl,10h                     ; Fonction supportée ?
        je   tega_end                   ; NON --> Termine la routine
;
       ;-- La fonction est supportée , CL contient maintenant --------
       ;-- la position des micro-commutateurs DIP de la carte EGA ----
;
        mov  al,cl                              ; Commutateurs DIP en AL
        shr  al,1               ; Décale d'une position vers la droite
        mov  bx,offset DGROUP:ega_dips          ; Offset de la table
        xlat                            ; Elément AL de la table en AL
        mov  ah,al                              ; Type de moniteur en AH
        mov  al,EGA                             ; C'est une carte EGA
        call trouve             ; Inscrit les données dans le vecteur
;
        cmp  ah,MONO            ; Connexion sur  écran monochrome ?
        je   is_mono                            ; OUI --> ni MDA ou HGC
;
        mov  frame.cga_possi,0                  ; Carte CGA impossible
        jmp  short tega_end                     ; Terminer la routine
;
is_mono: mov  frame.mono_possi,0        ; carte EGA est sur un moniteur
                                ; monochrome, ni MDA ni HGC ne peuvent
                                                ; être installées
;
tega_end: ret                           ; Retour au programme appelant
;
test_ega endp
;
;;-----------------------------------------------------------------
```

```
;-- TEST_CGA: Détermine si une carte CGA est installée
;
test_cga   proc near
;
           mov  dx,3D4h      ; Adresse de port du registre CRTC pour CGA
           call test_6845          ; Teste si 6845 CRTC est installé
           jc   tega_end                 ; NON --> Termine le test
           mov  al,CGA                   ; OUI, CGA est installée
           mov  ah,COLOR     ; Toujours un moniteur couleur avec CGA
           jmp  trouve       ; Inscrit les données dans le vecteur
;
test_cga   endp
;
;-------------------------------------------------------------------
;-- TEST_MONO: Teste la présence d'une carte MDA ou HGC
;
test_mono  proc near
;
           mov  dx,3B4h      ; Adr. de port du registre CRTC pour MONO
           call test_6845          ; Teste si 6845 CRTC est installé
           jc   tega_end                 ; NON --> Termine le test
;
           ;-- Une carte d'écran monochrome est installée ------------
           ;-- Le test suivant fait la distinction entre MDA et HGC ----
;
           mov  dl,08Ah                  ; Port d'état MONO en 3BAh
           in   al,dx                      ; Teste le port d'état
           and  al,80h                     ; Ne garde que le bit 7
           mov  ah,al                      ; et le range dans AH
;
           ;-- Si le contenu du bit 7 du port d'état est modifié au ----
           ;-- cours de l'un des tests suivants, c'est qu'il s'agit ----
           ;-- d'une carte HGC                                      ----
;
           mov  cx,8000h               ; 32768 itérations au maximum
test_hgc:  in   al,dx                      ; Teste le port d'état
           and  al,80h                     ; Ne garde que le bit 7
           cmp  al,ah                ; Son contenu a-t-il été modifié ?
           jne  is_hgc               ; Bit 7 = 1 --> c'est une carte HGC
           loop test_hgc                   ; Poursuit la boucle
;
           mov  al,MDA               ; Le bit 7 != 1 --> c'est une carte MDA
           jmp  set_mono                   ; Fixe les paramètres
;
is_hgc:    mov  al,HGC    ; Le bit 7 est devenu 1 --> c'est une carte HGC
set_mono:  mov  ah,MONO             ; MDA et HGC seulement sur écran MONO
           jmp  trouve                     ; Fixe les paramètres
;
test_mono  endp
;
;
;-- TEST_6845: retourne un indicateur de retenue à 1 si aucun 6845
;--            ne figure à l'adresse de port en DX
;
test_6845  proc near
;
           mov  al,0Eh                          ; Appelle le registre 14
           out  dx,al        ; N° registre dans le registre d'adresse CRTC
           inc  dx           ; DX maintenant sur registre de données CRTC
;
           in   al,dx              ; Recherche le contenu du registre 14
           mov  ah,al                          ; et le range en AH
;
           mov  al,4Fh                         ; Ecrit dans le registre
           out  dx,al                          ; une valeur quelconque
;
           mov  cx,100             ; Petite boucle d'attente pour que le
wait:      loop wait                           ; 6845 puisse réagir
;
           in   al,dx                ; Relit le contenu du registre 14
           xchg al,ah                          ; Intervertit AH et AL
           out  dx,al                      ; Restaure l'ancienne valeur
;
           cmp  ah,4Fh             ; La valeur écrite a-t-elle été lue ?
           je   t6845_end                  ; OUI --> Termine le test
;
           stc                   ; NON --> Met à 1 l'indicateur de retenue
;
t6845_end: ret                                 ; Retour à l'appelant
;
test_6845  endp
;
;-------------------------------------------------------------------
;-- TROUVE : enregistre dans le vecteur vidéo le type de carte vidéo
;--          (en AL) et le type de moniteur (en AH)
;
trouve     proc near
;
           mov bx,dI                    ; Adresse de la structure active
           cmp word ptr [bx],0                    ; système vidéo?
           je  set_data          ; NON --> Données dans la structure active
;
           add bx,2              ; OUI, adresse de la structure désactivée
;
set_data:  mov [bx],ax                 ; Place les données dans la structure
           ret                             ; Retour au programme appelant
;
trouve     endp
;
;-------------------------------------------------------------------
;
_text      ends                                ; Fin du segment de code
           end                                     ; Fin du programme
;
```

Listing : VIOSP.PAS

```
{*****************************************************************}
{*                          V I O S P                            *}
{*---------------------------------------------------------------*}
{*  Fonction      : Détermine le type des cartes vidéo installées. *}
{*---------------------------------------------------------------*}
{*  Auteur        : MICHAEL TISCHER                              *}
{*  Développé le  : 2/10/1988                                    *}
{*  Dernière MAJ  : 14/01/1991                                   *}
{*****************************************************************}

program VIOSP;

{$L viospa}                     { Intégration du module en assembleur }

    const NO_VIOS    = 0;                       { Pas de carte vidéo }
    VGA        = 1;                             { Carte VGA }
    EGA        = 2;                             { Carte EGA }
    MDA        = 3;                  { Monochrome Display Adapter }
    HGC        = 4;                    { Hercules Graphics Card }
    CGA        = 5;                    { Color Graphics Adapter }

    NO_MON     = 0;                          { Pas de moniteur }
    MONO       = 1;                      { Moniteur monochrome }
    COLOR      = 2;                       { Moniteur couleur }
    EGA_HIRES  = 3;                  { Moniteur haute résolution }
    ANAL_MONO  = 4;              { Moniteur analogique monochrome }
    ANAL_COLOR = 5;                { Moniteur couleur analogique }

type Vios = record     { Décrit la carte vidéo et le moniteur connecté }
            Carte,
            Moniteur : byte;
          end;
    ViosPtr = ^Vios;                { Pointeur sur une structure VIOS }

procedure GetVios( vp : ViosPtr ) ; external ;

var VidSys : array[1..2] of Vios; { Tableau avec deux structures vidéo }

{*****************************************************************}
{* PrintSys : Affiche diverses informations sur le système vidéo *}
{* Entrée   : - CARTE : Numéro de code de la carte vidéo         *}
{*            - MON   : Numéro de code du moniteur connecté      *}
{* Sortie   : Aucune                                             *}
{*****************************************************************}

procedure PrintSys( Carte, Mon : byte );

begin
  write(' Carte ');
  case Carte of
    VGA : write('VGA');
    EGA : write('EGA');
    MDA : write('MDA');
    CGA : write('CGA');
    HGC : write('HGC');
  end;
  write(' avec ');
  case Mon of
    MONO      : writeln('moniteur monochrome');
    COLOR     : writeln('moniteur couleur');
```

```
    EGA_HIRES  : writeln('moniteur haute résolution');
    ANAL_MONO  : writeln('moniteur analogique monochrome');
    ANAL_COLOR : writeln('moniteur analogique couleur');
  end;
end;

{*********************************************************}
{**                  PROGRAMME PRINCIPAL                **}
{*********************************************************}

begin
  GetVios( @VidSys );          { Détermine les cartes vidéo installées }
  writeln('VIOS  -  (c) 1988 by MICHAEL TISCHER');
  write('Système vidéo primaire : ');
  PrintSys( VidSys[1].Carte, VidSys[1].Moniteur );
  if VidSys[2].Carte <> NO_VIOS then       { Une autre carte  vidéo ? }
    begin                                                  { OUI }
      write('Système vidéo secondaire :');
      PrintSys( VidSys[2].Carte, VidSys[2].Moniteur );
    end;
end.
```

Listing : VIOSPA.ASM

```
;*****************************************************************
;*                          V I O S P A                          *
;*---------------------------------------------------------------*
;*  Fonction        : Fournit une fonction à intégrer dans un    *
;*                    programme en Turbo Pascal                  *
;*                    et qui permet de déterminer le type des cartes *
;*                    vidéo installées.                          *
;*---------------------------------------------------------------*
;*  Auteur          : MICHAEL TISCHER                            *
;*  Développé le    :  2/10/1988                                 *
;*  Dernière MAJ.:  14/01/1991                                   *
;*---------------------------------------------------------------*
;*  Assemblage      : MASM VIOSPA;                               *
;*                    ... intégrer ensuite par (SL VIOSPA)       *
;*                    dans un programme en Turbo Pascal          *
;*****************************************************************

;== Constantes pour la structure VIOS ==========================

                                ;Constantes pour la carte vidéo
NO_VIOS     = 0                 ;Pas de carte vidéo
VGA         = 1                 ;Carte VGA
EGA         = 2                 ;Carte EGA
MDA         = 3                 ;Monochrome Display Adapter
HGC         = 4                 ;Hercules Graphics Card
CGA         = 5                 ;Color Graphics Adapter

                                ;Constantes pour le moniteur
NO_MON      = 0                 ;Pas de moniteur
MONO        = 1                 ;Moniteur monochrome
COLOR       = 2                 ;Moniteur couleur
EGA_HIRES   = 3                 ;Moniteur haute résolution ou Multisyn
ANAL_MONO   = 4                 ;Moniteur analogique monochrome
ANAL_COLOR  = 5                 ;Moniteur couleur analogique

;== Segment de données =========================================

DATA    segment word public     ;Segment de données Turbo

DATA    ends

;== Segment de code ============================================

CODE        segment byte public   ;Segment de code Turbo

            assume cs:CODE, ds:DATA

public      getvios

;-- Les variables globales initialisées doivent être placées dans le ---
;-- segment de code.                                                  ---

vios_tab    equ this word

        ;-- Table de conversion traitant le résultat retourné par
        ;-- l'option 00h de la fonction 1Ah du BIOS VGA

            db NO_VIOS, NO_MON      ;Pas de carte vidéo
            db MDA    , MONO        ;Carte MDA sur moniteur monochrome
            db CGA    , COLOR       ;Carte CGA sur moniteur couleur
            db ?      , ?           ;Code 3 non utilisé
            db EGA    , EGA_HIRES   ;Carte EGA sur moniteur haute rés.
            db EGA    , MONO        ;Carte EGA sur moniteur monochrome
            db ?      , ?           ;Code 6 non utilisé
            db VGA    , ANAL_MONO   ;Carte VGA sur moniteur mono. analog.
            db VGA    , ANAL_COLOR  ;Carte VGA sur moniteur coul. analog.

ega_dips    equ this byte

          ;-- Table de conversion pour traiter les micro-commutateurs
          ;-- DIP de la carte EGA

            db COLOR, EGA_HIRES, MONO
            db COLOR, EGA_HIRES, MONO

;---------------------------------------------------------------------
;-- GETVIOS             : Détermine le type des cartes vidéo installées
;-- Appel depuis Pascal : GetVios(@Vidsys)
;-- Déclaration         : Procédure GetVios (vp::ViosPtr); external;
;-- Valeur retournée    : Aucune

getvios  proc near

sframe      struc               ;Structure pour accéder à la pile
cga_possi   db ?                ;Variable locale
ega_possi   db ?                ;       "
mono_possi  db ?                ;       "
bptr        dw ?                ;Reçoit BP
ret_adr     dw ?                ;Adresse de retour à l'appelant
vp          dd ?                ;Pointeur sur la première structure VIOS
sframe      ends                ;Fin de la structure

frame       equ [ bp - cga_possi ] ;Adresse les éléments de la structure

            push bp             ;Sauvegarde BP sur la pile
            sub  sp,3           ;Place pour les variables locales
            mov  bp,sp          ;Transfère SP en BP

            mov  frame.cga_possi,1 ;Peut être CGA
            mov  frame.ega_possi,1 ;Peut être EGA
            mov  frame.mono_possi,1;Peut être MDA ou HGC

            mov  di,word ptr frame.vp     ;Recherche offset de structure
            mov  word ptr [di],NO_VIOS    ;Aucun système vidéo
            mov  word ptr [di+2],NO_VIOS  ;trouvé pour le moment

            call test_vga       ;Teste si carte VGA
            cmp  frame.ega_possi,0 ;Carte EGA encore possible ?
            je   gv1            ;NON --> passe au test CGA

            call test_ega       ;Teste si carte EGA
gv1:        cmp  frame.cga_possi,0 ;Carte CGA encore possible ?
            je   gv2            ;NON --> passe au test MDA et HGC

            call test_cga       ;Teste si carte CGA
gv2:        cmp  frame.mono_possi,0;Carte MDA ou HGC encore possible ?
            je   gv3            ;NON --> Tests terminés

            call test_mono      ;Teste si carte MDA et HGC

            ;-- Détermine la carte vidéo active ----------------------

gv3:        cmp  byte ptr [di],VGA ;Carte VGA identifiée ?
            je   gv1_end        ;OUI, carte active déjà identifiée
            cmp  byte ptr [di+2],VGA ;Carte VGA comme système secondaire?
            je   gv1_end        ;OUI, carte active déjà identifiée

            mov  ah,0Fh         ;Détermine le mode vidéo courant par
            int  10h            ;l'interruption vidéo

            and  al,7           ;Seuls modes 0 à 7 sont intéressants
            cmp  al,7           ;Carte monochrome active ?
            jne  gv4            ;Non, en mode CGA ou EGA

            ;-- La carte MDA, HGC ou EGA (mono) est active ------------

            cmp  byte ptr [di+1],MONO ;Mono dans la première structure ?
            je   gv1_end        ;OUI, ordre correct
            jmp  short echange  ;NON, intervertir l'ordre

            ;-- Une carte CGA ou EGA est active ----------------------
```

```
gv4:      cmp  byte ptr [di+1],MONO ;Mono dans la première structure ?
          jne  gvi_end              ;NON, ordre correct

echange:  mov  ax,[di]        ;Recherche le contenu première structure
          xchg ax,[di+2]      ;Echange avec seconde structure
          mov  [di],ax

gvi_end:  add  sp,3           ;Elimine les variables locales de la pile
          pop  bp             ;Retire BP de la pile
          ret  4              ;Retourne à TURBO en supprimant les variables
                              ;sur la pile
getvios   endp

;---------------------------------------------------------------------
;-- TEST_VGA: détermine si une carte VGA est installée

test_vga  proc near

          mov  ax,1a00h       ;Appelle l'option 00h de la fonction 1Ah
          int  10h            ;du BIOS
          cmp  al,1ah         ;Fonction supportée?
          jne  tvga_end       ;NON --> Termine la routine

          ;-- La fonction est supportée, BL contient maintenant le code
          ;-- du système vidéo actif, BH celui du système non actif

          mov  cx,bx          ;Range le résultat dans CX
          xor  bh,bh          ;Annule BH
          or   ch,ch          ;Un seul système vidéo ?
          je   tvga_1     ;OUI --> Sélectionne code du premier système

          ;-- Convertit le code du second système ----------------------

          mov  bl,ch          ;Code du second système en BL
          add  bl,bl          ;Calculer l'offset dans la table
          mov  ax,vios_tab[bx] ;Rechercher le code dans la table
          mov  [di+2],ax    ;et stocke dans structure du prog appelant
          mov  bl,cl          ;Remet dans BL codes du premier système

          ;-- Convertit le code du premier système ---------------------

tvga_1:   add  bl,bl          ;Calculer l'offset dans la table
          mov  ax,vios_tab[bx] ;Recherche le code dans la table
          mov  [di],ax        ;stocke dans la struct du prog appelant

          mov  frame.cga_possi,0 ;Le test CGA n'a plus lieu d'être
          mov  frame.ega_possi,0 ;Le test EGA non plus
          mov  frame.mono_possi,0 ;Reste à examiner MONO
          mov  bx,di          ;Adresse de la structure active
          cmp  byte ptr [bx],MDA ;Système monochrome identifié ?
          je   do_tmono       ;OUI --> Effectue le test MDA/HGC

          add  bx,2           ;Adresse de la structure inactive
          cmp  byte ptr [bx],MDA ;Système monochrome identifié ?
          jne  tvga_end       ;NON termine la routine

do_tmono: mov  word ptr [bx],0   ;Simule la non-identification du sys.

          mov  frame.mono_possi,1;Exécute le test monochrome

tvga_end: ret                 ;Retour au programme appelant

test_vga  endp

;---------------------------------------------------------------------
;-- TEST_EGA: Détermine si une carte EGA est installée

test_ega  proc near

          mov  ah,12h         ;Fonction 12h
          mov  bl,10h         ;Option 10h
          int  10h            ;Appelle le BIOS EGA
          cmp  bl,10h         ;Fonction supportée ?
          je   tega_end       ;NON --> Termine la routine

          ;-- La fonction est supportée , CL contient maintenant  ----
          ;-- la position des micro-commutateurs DIP de la carte EGA --

          mov  bl,cl          ;Commutateurs DIP en BL
          shr  bl,1           ;Décale d'une position vers la droite
          xor  bh,bh          ;Octet fort pour index à 0
          mov  ah,ega_dips[bx] ;Recherche l'élément de la table
          mov  al,EGA         ;C'est une carte EGA
          call trouve         ;Inscrit les données dans le vecteur

          cmp  ah,MONO        ;Connexion sur écran monochrome ?
          je   is_mono        ;OUI --> ni MDA ou HGC

          mov  frame.cga_possi,0 ;Carte CGA impossible
          jmp  short tega_end ;Termine la  routine

is_mono:  mov  frame.mono_possi,0;Comme la carte EGA est sur moniteur
                              ;monochrome, ni MDA ni HGC ne peuvent
                              ;être installées

tega_end: ret                 ;Retour au programme appelant

test_ega  endp

;---------------------------------------------------------------------
;-- TEST_CGA: Détermine si une carte CGA est installée

test_cga  proc near

          mov  dx,3D4h        ;Adr port reg d'adresse CRTC pour CGA
          call test_6845      ;Teste si 6845 CRTC installé
          jc   tega_end       ;NON --> Termine le test

          mov  al,CGA         ;OUI, une carte CGA est installée
          mov  ah,COLOR       ;Toujours un moniteur couleur avec CGA
          jmp  trouve         ;Inscrit les données dans le vecteur

test_cga  endp

;---------------------------------------------------------------------
;-- TEST_MONO: Teste la présence d'une carte MDA ou HGC

test_mono proc near

          mov  dx,3B4h        ;Adr. port du reg. adr. CRTC pour MONO
          call test_6845      ;Teste si 6845 CRTC installé
          jc   tega_end       ;NON --> Termine le test

          ;-- Une carte d'écran monochrome est installée ---------------
          ;-- Le test suivant fait la distinction entre MDA et HGC ----

          mov  dl,0BAh        ;Port d'état MONO en 3BAh
          in   al,dx          ;Teste le port d'état
          and  al,80h         ;Ne garde que le bit 7
          mov  ah,al          ;et le ranger en AH

          ;-- Si le contenu du bit 7 du port d'état est modifié au ----
          ;-- cours de l'un des tests suivants, c'est qu'il s'agit ----
          ;-- d'une carte HGC                                       ----

          mov  cx,8000h       ;32768 itérations au maximum
test_hgc: in   al,dx          ;Teste le port d'état
          and  al,80h         ;Ne garde que le bit 7
          cmp  al,ah          ;Son contenu a-t-il été modifié ?
          jne  is_hgc         ;Bit 7 = 1 --> c'est une carte HGC
          loop test_hgc       ;Poursuit la boucle

          mov  al,MDA         ;Le bit 7 n'est pas à 1 --> carte MDA
          jmp  set_mono       ;Fixe les paramètres

is_hgc:   mov  al,HGC         ;Le bit 7 est à 1 -->  une carte HGC
set_mono: mov  ah,MONO        ;MDA et HGC seulement sur écran MONO
          jmp  trouve         ;Fixe les paramètres

test_mono endp

;---------------------------------------------------------------------
;-- TEST_6845: retourne un indicateur de retenue à 1 si aucun 6845 ne --
;--            figure à l'adresse de port en DX                      --

test_6845 proc near

          mov  al,0Ah         ;Appelle le registre 10
          out  dx,al      ;Numéro de reg. dans le reg. d'adresse CRTC
          inc  dx         ;DX maintenant sur registre de données CRTC

          in   al,dx          ;Recherche le contenu du registre 10
          mov  ah,al          ;et le range en AH

          mov  al,4Fh         ;Ecrit dans le registre 10
          out  dx,al          ;une valeur quelconque

          mov  cx,100         ;Petite boucle d'attente pour que le
wait:     loop wait           ;6845 puisse réagir

          in   al,dx          ;Relit le contenu du registre 10
          xchg al,ah          ;Intervertit AH et AL
          out  dx,al          ;Restaure l'ancienne valeur

          cmp  ah,4Fh         ;La valeur écrite a-t-elle été lue ?
          je   t6845_end      ;OUI --> Termine le test

          stc                 ;NON --> Met à 1 indicateur de retenue

t6845_end: ret                ;Retour au programme appelant

test_6845 endp

;---------------------------------------------------------------------
;-- TROUVE : enregistre dans le vecteur vidéo le type de carte vidéo ---
;--          (en AL) et le type dumoniteur (en AH)                    ---
```

```
;
;trouve    proc near
;
;         mov bx,df             ;Adresse de la structure active
;         cmp word ptr [bx],0   ;A-t-on déjà identifié un sys. vidéo ?
;         je  set_data          ;NON --> Données dans la struct active
;
;         add bx,2              ;OUI, adresse de la structure désactivée
;
;set_data: mov [bx],ax           ;Place les données dans la structure
;         ret                   ;Retour au programme appelant
;
;trouve   endp
;
;;-------------------------------------------------------------------------
;
;code     ends                  ;Fin du segment de code
;        end                   ;Fin du programme
;
```

4.4. Structure fondamentale d'une carte vidéo

L'une a pour rôle de générer l'affichage du texte, l'autre ne se contente pas d'un mode graphique faiblard et la troisième crée des images ressemblant à des photos : les différences entre les divers standards vidéo sont frappantes dans le monde PC. Chaque type de carte vidéo, qu'il s'agisse d'une carte MDA ordinaire ou d'une carte moderne Super VGA, chacune fonctionne selon un principe déterminé. Avant d'examiner les différents standards vidéo dans les sections suivantes, notez d'abord ici quelques informations relatives à la structure fondamentale et le mode de fonctionnement des cartes vidéo. Vous apprendrez ainsi :

- ✓ comment un moniteur crée une image vidéo,
- ✓ comment le contrôleur CRT gère la structure de l'écran,
- ✓ quelles sont les tâches des registres du contrôleur CRT et
- ✓ quels sont les éléments d'une carte vidéo.

La seconde partie du chapitre traite de la RAM vidéo qui est un élément essentiel pour la création d'une image vidéo et la programmation des cartes vidéo.

Construction de l'écran à travers le moniteur

Le moniteur se trouve au bout de la chaîne qui conduit une image de la RAM vidéo à la réalisation sur l'écran. Contrairement à la carte vidéo, il s'agit d'une machine "inintelligente", qui ne peut donc pas être programmée. Tous les moniteurs utilisés dans le domaine du PC sont ce qu'on appelle des "raster scan devices", c'est-à-dire des machines sur lesquelles l'écran se compose d'une multitude de petits points disposés dans une grille rectangulaire.

Lors de la construction de l'image, le rayon électronique du tube cathodique balaye chaque point et le fait briller s'il doit être visible sur l'écran. Concrètement, cela est réalisé en activant le rayon électronique frappant ce point et en faisant ainsi briller une particule de phosphore sur le tube cathodique.

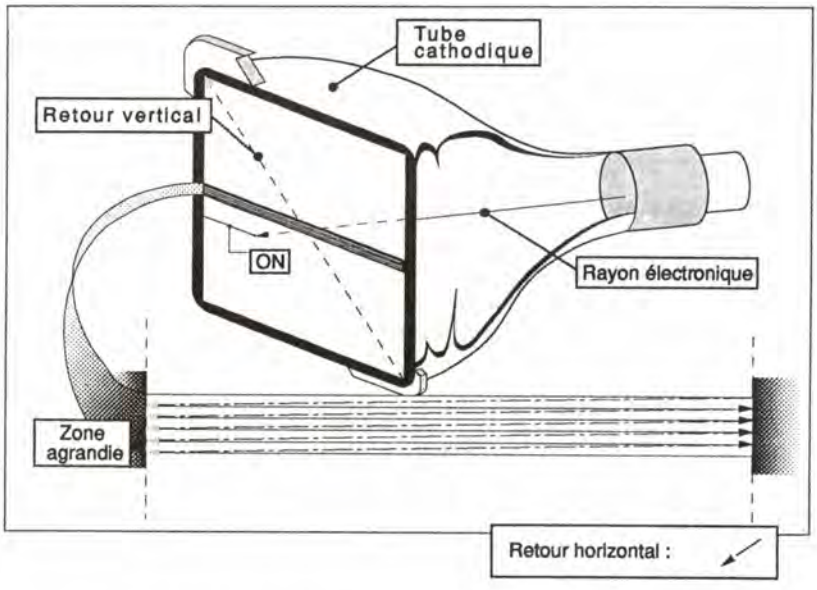
Un seul rayon électronique suffit pour construire l'image sur un moniteur monochrome, alors que sur un moniteur couleur trois rayons électroniques doivent balayer simultanément l'écran. Un point d'image ne se compose pas dans ce cas d'une mais de trois

particules de phosphore dans les trois couleurs fondamentales rouge, vert et bleu. Un rayon éléctronique est donc chargé de chacune de ces couleurs. En combinant ces trois couleurs avec une intensité diversifiée lors de ce mélange, on peut obtenir toutes les couleurs.

Comme une particule de phosphore ionisée ne brille que très peu de temps, l'écran doit être entièrement balayé de cette façon de nombreuses fois par seconde pour produire à nos yeux l'impression d'une image fixe. Sur les moniteurs de PC, ce balayage est effectué entre 50 et 70 fois par seconde, la qualité de l'image étant liée à la fréquence de répétition.

La construction de l'image commence toujours par le coin supérieur gauche de l'écran. De là, le rayon électronique se déplace horizontalement vers la droite sur la première ligne de grille. A la fin de cette ligne, il revient au début de la ligne mais descend en même temps d'une ligne. Commence alors la construction de la deuxième ligne de la grille, à la fin de laquelle se déplace à nouveau, toujours de la même façon, vers le début de la ligne suivante. Une fois que la fin de l'écran a été ainsi atteinte, le rayon saute à nouveau au coin supérieur gauche de l'écran et la construction de l'écran peut reprendre. Cette procédure s'appelle retour horizontal ou "horizontal retrace" et retour vertical ou "vertical retrace".

Il faut toutefois noter que le déplacement du rayon électronique n'est pas commandé par l'écran lui-même mais par le contrôleur de la carte vidéo, à travers certains canaux de signal spéciaux, comme nous allons l'expliquer bientôt.



Parcours du rayon électronique sur l'écran

La résolution du moniteur est naturellement directement fonction du nombre de lignes et de colonnes de la grille que balaye le rayon électronique lors de la construction de l'image. Il va donc de soi qu'un moniteur ne disposant que d'une grille de 200 lignes sur 640 colonnes ne pourra offrir une résolution aussi élevée qu'une carte EGA ou VGA dotée de 640x480 points. Voici les résolutions dont disposent en principe les quatre types de moniteur utilisés sur le PC :

Résolutions des différents types de moniteur		
Moniteur	**Lignes**	**Colonnes**
MDA/Hercules	350	720
CGA	200	640
EGA	350	640
VGA*	350	640
Super VGA*	600	800
Multisync	variable, jusqu'à 800	variable, jusqu'à 1200
* Ces moniteurs existent en modes monochrome et couleur		

Le contrôleur CRT

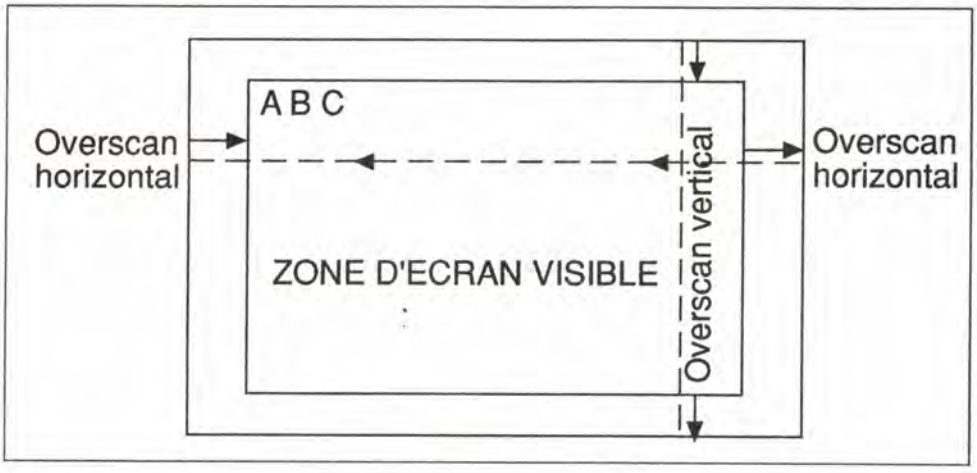
Le coeur d'une carte vidéo est constitué par le contrôleur CRT, qui tire son nom de la désignation anglaise de l'écran, Cathode Ray Tube. On l'appelle cependant également contrôleur d'écran, contrôleur vidéo ou simplement CRTC.

Il contrôle le déroulement des opérations à l'intérieur de la carte vidéo et génère les signaux dans le temps dont le moniteur a besoin pour construire l'image. Il est également chargé de contrôler le crayon optique, de produire le curseur clignotant de l'écran et de contrôler la RAM vidéo.

Pour lancer la construction d'une ligne de grille par le moniteur, le CRTC sort au début de chaque ligne un signal dit "display enable", qui active le rayon électronique. Au cours du déplacement vers la droite de ce rayon, à travers les différentes colonnes de la ligne, le CRTC gère les différents signaux pour le ou les rayons électroniques, de sorte que les points d'image apparaissent à l'endroit voulu sur l'écran. Lorsque la fin de la ligne est atteinte, un signal de synchronisation horizontal est sorti pour amener le rayon électronique sur le bord gauche de la ligne de grille suivante. Le signal display enable est cependant désactivé auparavant pour que le retour du rayon électronique ne produise pas une ligne visible sur l'écran. Une fois atteint le début de la ligne de grille suivante, ce signal est à nouveau et la construction de la ligne suivante débute.

Le rayon électronique ayant besoin de moins de temps pour ce retour qu'il n'en faut au CRTC pour rechercher de nouvelles informations dans la RAM vidéo et pour les préparer, une petite pause intervient. Le rayon électronique ne pouvant cependant être arrêté, il en résulte ce qu'on appelle un overscan, qui se traduit par une marge à gauche

et à droite du contenu véritable de l'écran. Cet effet secondaire, au fond indésirable, a tout de même l'intérêt d'assurer que le contenu de l'écran ne soit pas recouvert par une partie du cadre du moniteur. Cela pourrait en effet tout à fait se produire si l'écran était entièrement employé à l'affichage des données. En laissant activé le rayon électronique pendant qu'il parcourt ce cadre, on peut faire apparaître un cadre de l'écran en couleur.



Formation d'un cadre lors de la construction de l'écran

Une fois que le rayon électronique a atteint la fin de la dernière ligne de grille, il doit être ramené dans le coin supérieur gauche de l'écran. Ici encore, le signal display enable est désactivé et un signal de synchronisation vertical est alors sorti. Une fois le rayon parvenu sur le coin supérieur gauche de l'écran, le signal display enable est à nouveau activé et une nouvelle construction de l'écran démarre.

Comme pour le retour horizontal, un fossé dans le temps apparaît et un overscan est également généré, qui se traduit par un cadre vertical de l'écran.

Les registres du contrôleur CRT

La succession dans le temps des différents signaux n'est cependant pas imposée de façon définitive et elle varie aussi d'un mode vidéo à l'autre. C'est pourquoi le CRTC dispose de toute une série de registres décrivant les sorties de signaux et leur agencement dans le temps. Les pages suivantes seront consacrées à la description de la structure et de la programmation de ces registres. Nous concentrerons notre attention sur les registres du contrôleur vidéo 6845 de Motorola, qui équipe les cartes graphiques MDA, CGA et Hercules. La carte EGA dispose comme CRTC d'un circuit spécial LSI (large scale integration), dont les registres présentent une structure un peu plus complexe, en raison même des possibilités plus variées de cette carte. Les techniques ici décrites peuvent toutefois généralement être aussi appliquées à cette carte.

Les registres du contrôleur vidéo 6845 de Motorola		
Registre	**Signification**	**Accès**
00h	Total de caractères horizontalement	Ecriture
01h	Caractères affichés horizontalement	Ecriture
02h	Signal de synchronisation horizontal après ... caractères	Ecriture
03h	Durée du signal horizontal de synchronisation en caractères	Ecriture
04h	Total de caractères verticalement	Ecriture
05h	Nombre ajusté de caractères verticalement	Ecriture
06h	Caractères affichés verticalement	Ecriture
07h	Signal de synchronisation vertical après ... caractères	Ecriture
08h	Mode d'entrelacement	Ecriture
09h	Nombre de lignes de grille (de balayage) par ligne de l'écran	Ecriture
0Ah	Ligne de départ du curseur clignotant de l'écran	Ecriture
0Bh	Ligne de fin du curseur clignotant de l'écran	Ecriture
0Ch	Adresse de départ de la page écran affichée (octet de poids fort)	Ecriture
0Dh	Adresse de départ de la page écran affichée (octet de poids faible)	Ecriture
0Eh	Adresse de caractère du curseur clignotant de l'écran (octet de poids fort)	Lecture & écriture
0Fh	Adresse de caractère du curseur clignotant de l'écran (octet de poids faible)	Lecture & écriture
10h	Position du crayon optique (octet de poids fort)	Lecture
11h	Position du crayon optique (octet de poids faible)	Lecture

L'accès à ces registres ainsi qu'à tous les autres registres de la carte vidéo se fait à travers ce qu'on appelle les ports, à l'aide des instructions du langage machine IN et OUT. Les différents registres du CRTC ne peuvent cependant être appelés directement, mais seulement par l'intermédiaire d'un registre spécial d'adresse. Le numéro du registre CRTC appelé doit, à cet effet, être sorti sur le port du registre d'adresse. Le contenu du registre peut ensuite être tiré d'un registre spécial de données à l'aide de l'instruction du langage machine IN. Pour écrire une valeur dans le registre adressé, cette valeur doit être transférée dans le registre de données avec l'instruction OUT et le CRTC la transférera automatiquement de là dans le registre adressé. Ces deux registres figurent toujours à des adresses de port consécutives mais ces adresses peuvent varier d'une carte vidéo à l'autre.

Dans la description des différentes cartes vidéo, nous vous proposerons chaque fois des tables décrivant le contenu des différents registres du CRTC sous les divers modes vidéo. Nous allons donc vous montrer maintenant, dans un exemple, comment le contenu de

ces registres est calculé et comment les différents registres peuvent être combinés entre eux. Si vous reproduisez ces calculs sur votre calculatrice de poche ou sur un PC, vous constaterez que certains ne produisent pas un résultat entier. Les registres du CRTC ne pouvant recevoir que des valeurs entières, il faut donc chaque fois arrondir à la valeur immédiatement inférieure ou supérieure.

Les différents calculs à effectuer reposent sur la largeur de bande et la fréquence de balayage horizontal ou vertical d'un moniteur.

Largeurs de bande et fréquences de balayage des différents systèmes vidéo

Système vidéo	Résolution	Largeur de bande	Fréquence de balayage verticale	Fréquence de balayage horizontale
MDA	720 * 350	16,257 MHz	50 Hz	18,43 KHz
CGA	640 * 200	14,318 MHz	60 Hz	15,75 KHz
HGC	720 * 350	16,257 MHz	50 Hz	18,43 KHz
EGA	640 * 350	16,257 MHz	60 Hz	21,85 KHz
	640 * 200	14,318 MHz	60 Hz	15,75 KHz
	720 * 350	16,257 MHz	50 Hz	18,43 KHz
VGA	640 * 400	25,175 MHz	70 Hz	31,50 KHz
	720 * 400	28,322 MHz	70 Hz	31,50 KHz
	640 * 480	25,175 MHz	60 Hz	31,50 KHz
	640 * 350	25,175 MHz	70 Hz	31,50 KHz
Super VGA	800 * 600	28,322 MHz	56 Hz	35,20 KHz
	800 * 600	30,425 MHz	60 Hz	37,88 KHz
	1024 * 768	50,350 MHz	60 Hz	48,00 KHz

(Unités : Hz=hertz, KHz=kilohertz, MHz=mégahertz)

Dans la présentation ci-dessus, la largeur de bande indique le nombre de points que le rayon électronique du tube cathodique balaye en une seconde. On parle également de fréquence de points ou Dot Rate. La fréquence de balayage vertical indique le nombre de répétitions de l'image par seconde, alors que la fréquence de balayage horizontal indique le nombre de lignes de grille que le rayon électronique parcourt en une seconde.

Comme troisième élément du tableau ci-dessus, la fréquence de balayage horizontale se rapport au nombre de lignes de grille parcourues par le rayon électronique en une seconde. Il dépend de la largeur de bande et du nombre de points d'image par ligne.

D'après le tableau précédent, les cartes MDA, CGA et Hercules n'utilisent qu'une seule largeur de bande contrairement aux cartes EGA et VGA qui ont besoin de deux largeurs

de bande différentes. Dans les cartes Super VGA, on rencontre trois largeurs de bande différentes en plus des largeurs VGA normales.

Le nombre de largeurs de bande soutenues par une carte vidéo est spécifié généralement sur la carte-mère. Une horloge individuelle est utilisée en fait pour chaque largeur de bande et donne la mesure au contrôleur CRT dans le vrai sens du terme. Les puces à quartz se reconnaissent facilement sur la carte car elles sont argentées contrairement aux autres puces qui sont noires. Le taux de fréquence est en outre indiqué au-dessus de ces dernières.

Exemple de calcul

Considérons le mode graphique de la carte Hercules qui ne peut être activé que par la programmation directe des divers registres CRTC. Normalement, le BIOS se charge d'initialiser les registres lorsque vous activez un mode vidéo par la fonction 00h du BIOS vidéo. Malgré tout, il est intéressant de calculer les diverses valeurs pour les différents registres CRTC en prenant l'exemple du mode de texte 80*25 caractères sur une carte CGA.

Vous devez vous servir du tableau précédent pour trouver une entrée pour la résolution de 80*25 caractères. Le contrôleur CRT et le moniteur ne reconnaissent pas en effet les caractères, mais n'acceptent que les points d'image. Les caractères se composent en fait de points d'image contenus dans une matrice carrée. Pour afficher un caractère sur l'écran, le contrôleur CRT le décompose en points à travers sa matrice carrée.

Dans une carte CGA, la matrice carrée s'élève à 8*8 points, si bien qu'une résolution écran de 80*25 caractères de texte correspond à une résolution graphique de 640*200 points. Pour ce mode, la largeur de bande atteint 14,318 MHz avec une fréquence de répétition d'image de 60 Hz et une fréquence de balayage horizontale de 15,740 KHz.

En divisant tout d'abord la largeur de bande par la fréquence de balayage horizontal, on obtient le nombre de points d'image par ligne de balayage.

```
  Largeur de bande                  14,318 MHz
/ Fréquence de balayage horizontal  15,750 KHz
-----------------------------------------------
  Points d'image par ligne             909
```

Comme la plupart des registres CRTC ne contiennent pas des valeurs exprimées en points d'image mais en caractères, cette valeur doit être convertie en nombre de caractères par ligne. Elle doit pour cela être divisée par la largeur de la matrice de caractère, qui est de 8 points d'image sur la carte CGA.

```
  Points d'image par ligne  909
/ Points par caractère        8
--------------------------------
  Caractères par ligne      114
```

Cette valeur, diminuée de 1, doit être inscrite dans le premier registre du CRTC. Elle désigne donc le nombre total de caractères par ligne. Dans le deuxième registre doit être chargé le nombre de caractères devant effectivement apparaître sur chaque ligne de l'écran. En mode de texte 80x25 caractères, il s'agit bien sûr de 80.

En soustrayant du nombre total le nombre de caractères effectivement affichés, on obtient le nombre de caractères qui pourraient être sortis durant le retour horizontal et l'overscan. Il s'agit en l'occurrence de 34 caractères.

Dans le quatrième registre du CRTC doit être inscrite la durée du retour horizontal du rayon électronique. Ce registre ne reçoit toutefois pas directement la durée en temps mais le nombre de caractères qui auraient pu être sortis dans cet intervalle de temps. Cette valeur ne peut être fixée librement par la carte vidéo car elle dépend des caractéristiques du moniteur. Elle représente généralement entre 5% et 15% du nombre total de caractères par ligne. Pour un moniteur couleur, il s'agit exactement de 10 caractères.

Il reste donc encore 24 caractères pour l'overscan ou, plus précisément, pour le cadre horizontal de l'écran. C'est le troisième registre du CRTC, qui définit après combien de caractères commence le retour horizontal du rayon électronique, qui décide de la répartition de ces caractères entre les marges gauche et droite de l'écran. Le BIOS prévoit ici la valeur 90, de sorte que 10 caractères sont encore sortis pour le cadre de l'écran à la suite des caractères affichés. Les 14 caractères restants se retrouvent de ce fait avant le début de la ligne suivante et forment ainsi le bord gauche de l'écran.

Les calculs pour les paramètres verticaux, c'est-à-dire le nombre de lignes verticalement, la position du signal de synchronisation, etc., se déroulent suivant le même principe que les calculs des paramètres horizontaux.

Les calculs commencent par le nombre de lignes de grille par écran, qui est obtenu en divisant le nombre de lignes mises en place par seconde par le nombre de régénérations de l'image par seconde.

```
  Fréquence de balayage horizontal  15,750 KHz
/ Régénérations de l'image              60  Hz
-----------------------------------------------
  Lignes de grille                     262
```

Les caractères ayant en mode de texte de la carte CGA non seulement 8 points de largeur mais aussi 8 points de hauteur, c'est à nouveau une division par 8 qui nous fournira le nombre de lignes de texte par écran.

```
  Lignes de grille          262
/ Points par caractère        8
-------------------------------
  Lignes par écran           32
```

Ce résultat doit être décrémenté d'une unité et être chargé dans le cinquième registre du CRTC. On inscrit dans le septième registre le nombre de lignes affichées (25) par écran. Comme cela représente 7 lignes de moins qu'il n'y en a effectivement, ces lignes supplémentaires seront à nouveau employées pour le retour vertical et l'overscan, le retour vertical du rayon électronique débutant après la 28ème ligne.

La hauteur de chaque caractère, qui joue un rôle capital dans ces calculs, doit être inscrite dans le registre 9 du CRTC, après avoir été décrémentée d'une unité (nous obtiendrons donc 7 en l'occurrence). Elle désigne en même temps la limite supérieure du domaine dans lequel doivent être comprises les valeurs des registres 10 et 11, qui indiquent respectivement la première et la dernière lignes de grille entre lesquelles le curseur clignotant de l'écran doit recouvrir le caractère placé sur sa position. Le contenu des registres 14 et 15, qui indique la position du curseur clignotant de l'écran, définit de quel caractère il s'agit. Ces deux valeurs ne désignent toutefois pas les ligne et colonne de l'emplacement du curseur, mais plutôt sa distance par rapport au coin supérieur gauche de l'écran. Cette valeur est obtenue en multipliant le numéro de la ligne devant recevoir le curseur par le nombre de colonnes par ligne et en ajoutant à ce produit le numéro de la colonne où devra apparaître le curseur. L'octet de poids fort de ce résultat doit alors être chargé dans le registre 14, l'octet de poids faible dans le registre 15.

Structure d'une carte vidéo

Dans le cadre de la construction d'écran, le contrôleur CRT prend en charge toute une série de tâches, mais il ne peut pas les réaliser tout seul. Une carte vidéo se compose ainsi d'autres unités de fonction travaillant en collaboration avec le contrôleur CRT et liées entre elles. La figure suivante montre une représentation simplifiée d'une carte vidéo.

Le point de départ de la réalisation d'une image est toujours constitué par la RAM vidéo, une mémoire RAM figurant sur la carte vidéo et contenant des informations sur les caractères à afficher et sur leur mode d'affichage (leur couleur). A la RAM vidéo accède d'abord le générateur de caractères, qui lit l'un après l'autre les caractères à sortir dans la RAM vidéo et construit à l'aide d'une table de modèles de caractères les motifs de bits qui devront représenter ensuite le caractère voulu sur l'écran. C'est également de la RAM vidéo que le contrôleur d'attribut tire les informations sur le mode d'affichage du caractère et prépare ainsi les couleurs ou attributs appropriés (inversé, souligné, etc.). Les deux circuits retransmettent les informations préparées au contrôleur de signal, qui les traduit en signaux (de canal) appropriés, qui, une fois transmis au moniteur, produiront une image. Le contrôleur de signal est lui-même commandé par le contrôleur CRT, qui constitue le cerveau d'une carte vidéo et représente, avec le moniteur et la RAM vidéo, un des principaux éléments du système vidéo.

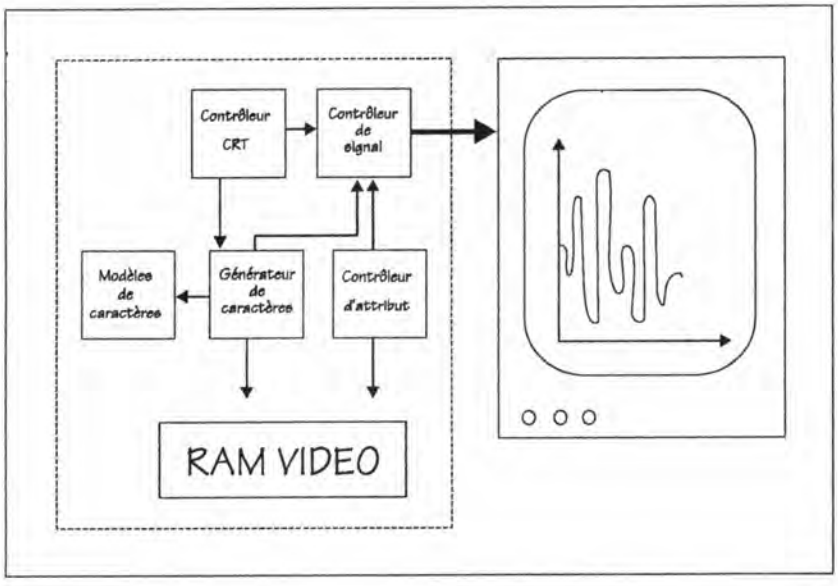


Schéma d'une carte vidéo

Le fractionnement de la carte vidéo en diverses unités était représenté parfaitement sur les premières cartes vidéo parce que les différents supports de fonctions étaient intégrés dans divers IC. La miniaturisation croissante des composants a fait que pratiquement tous les composants se retrouvent réunis dans un seul IC. Cela vaut également pour les cartes EGA et VGA dont la structure est plus complexe que le schéma représenté plus haut.

Vous en saurez davantage sur la structure concrète des différentes cartes vidéo et de leurs disparités dans les sections suivantes. Il convient pour l'instant d'examiner la RAM vidéo qui occupe une place prépondérante dans la programmation des cartes vidéo et constitue le seul lien entre les divers standards vidéo en mode de texte.

4.4.1. La RAM vidéo

La RAM vidéo est l'endroit où toutes les cartes vidéo, depuis la MDA jusqu'à la Super VGA, viennent stocker les informations d'image aussi bien en mode de texte qu'en mode graphique. Pour la programmation directe d'une carte graphique, c'est-à-dire le passage par le BIOS en ROM, il important de connaître la structure et l'état de la RAM vidéo. Dans les divers modes graphiques des différentes cartes vidéo, la RAM vidéo est organisée de manière variée. Mais en modes de texte, sa structure est identique dans toutes les cartes vidéo.

Cette section décrit le mode d'accès à la RAM vidéo à partir de programmes en langage évolué et répond à des questions telles que :

- ✓ Où se trouve la RAM vidéo dans la zone d'adressage du PC ?
- ✓ Quand et comment accéder à la RAM vidéo ?
- ✓ Structure de la RAM vidéo en mode de texte

Pour de plus amples informations sur l'organisation de la RAM vidéo dans les divers modes graphiques, veuillez vous reporter aux sections suivantes.

La RAM vidéo dans la zone d'adressage du PC

Vous savez certainement qu'il n'est pas indispensable de placer la mémoire RAM du PC sur la carte-mère mais qu'elle peut être logée sur l'une des cartes d'extension. Tous les standards vidéo connus jusqu'à présent dans le monde PC sont utilisables dans ce cas dans la mesure où ils réservent un espace RAM variable dans la zone d'adressage du PC et s'en servent ensuite comme RAM vidéo.

Bien que les différentes cartes vidéo déterminent l'endroit où cette RAM vidéo doit se situer dans la zone d'adressage du PC, elles n'ont pas toutefois carte blanche. En effet, la place disponible n'est pas tellement importante parce que les développeurs des premiers PC ont été obligés de loger la RAM, la ROM et les extensions à l'intérieur d'un Mo. Le résultat est que les cartes vidéo n'ont pu bénéficier que des segments de mémoire A et B commençant aux adresses de segment A000h et B000h offrant respectivement 64 Ko.

Pour permettre à l'utilisateur d'installer simultanément une carte vidéo monochrome et couleur s'il le souhaitait, ces deux types de cartes vidéo divisent le segment B en deux parties identiques. Les cartes vidéo monochromes, c'est-à-dire les cartes MDA et Hercules, occupent les cellules B000:0000 à B000:7FFF. A partir de B800:0000 (identique à B000:8000), la RAM vidéo appartient aux cartes couleur, notamment CGA, EGA et VGA.

La situation n'est pas toujours évidente dans le cas des cartes EGA et VGA. Par exemple, les cartes EGA peuvent également être connectées sur un moniteur MDA et se comporter alors comme une carte MDA. Sa RAM vidéo ne commence plus à B800h, mais B000h, tout comme une carte vidéo monochrome. Il en est de même des cartes VGA non connectées sur un moniteur MDA mais capables pourtant de simuler des cartes MDA sur un moniteur VGA tout à fait normal. Dans ce cas, la RAM vidéo commence déjà à B000h.

Les 32 Ko dont dispose une carte vidéo ne sont pas toujours utilisées en totalité pour sa RAM vidéo. Les cartes MDA occupent seulement les 4 premiers Ko de la RAM vidéo. Quant aux cartes CGA, elles occupent les 16 premiers Ko, soit près de la moitié de la zone disponible. Les cartes Hercules consomment la totalité des 32 Ko et sont en outre

capables d'utiliser la seconde moitié du segment B, soit la zone mémoire commençant à B800:0000, pour une seconde page graphique. Généralement, cette faculté peut être activée à l'aide d'un commutateur DIP sur la carte pour qu'elle libère la zone commençant à B800:0000 en faveur d'une carte couleur.

Les cartes EGA et VGA s'approprient également la totalité de la RAM vidéo. Elles disposent habituellement de 256 Ko de RAM vidéo et davantage dont seulement les 32 premiers Ko sont accessibles par le segment B. La section 4.8.2 montre comment accéder la partie restante de la RAM vidéo avec ces cartes.

Adressage de la RAM vidéo

Sachant que la RAM vidéo se situe dans la zone d'adressage normale du PC, elle peut être réclamée comme une mémoire RAM tout à fait ordinaire. Pour construire une image vidéo sur sa base, les divers composants de la carte vidéo achèvent de lire la totalité de la RAM vidéo avec une fréquence de répétition d'image pouvant atteindre 70 fois par seconde.

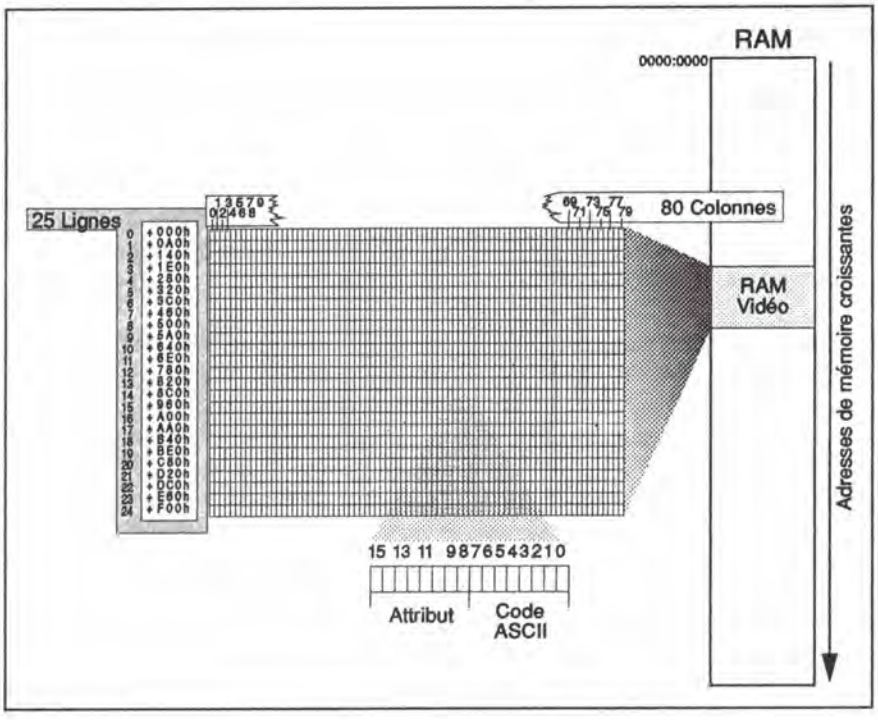
Pendant la durée de cet accès, aucun accès simultané à travers un programme ou le BIOS en ROM n'est autorisé. La plupart des cartes vidéo ont d'ailleurs repris leur place dans l'électronique pour éviter des collisions lors de l'accès à la RAM vidéo. Les premiers partisans de la carte originale CGA d'IBM constituent ici la seule exception. Dans ce cas, le programme concerné ou le BIOS en ROM doit prendre les mesures nécessaires pour éviter les collisions. Reportez-vous à la section 4.7 pour de plus amples informations.

Structure de la RAM vidéo en mode de texte

L'organisation de la RAM vidéo en mode texte constitue le seul consensus, relativement infime, entre les divers types de cartes vidéo. Comme la grande majorité des programmes fonctionnent en mode texte, les programmeurs ont la possibilité d'écrire directement les caractères dans la RAM vidéo à l'aide d'une routine de sortie unique sans être obligés de tenir compte du type de la carte vidéo installée. Tous les programmes renommés depuis Lotus 123 jusqu'à dBASE III se servent de cette technique car la vitesse d'affichage obtenue est supérieure à celle des fonctions BIOS.

A la fin de ce chapitre, vous trouverez trois programmes écrits en BASIC, Pascal et C. Ils présentent une routine pour l'accès direct à la RAM vidéo en mode texte. Vous pouvez évidemment augmenter la vitesse de cette routine en la convertissant tout simplement en Assembleur, mais dans ce cas la routine devient extrêmement rapide.

Pour mieux comprendre le fonctionnement des routines, il convient de se représenter concrètement la structure de la RAM vidéo comme le montre la figure suivante :



Structure standard de la RAM vidéo en mode de texte

Comme le montre la figure précédente, chaque position de l'écran occupe deux octets dans la RAM vidéo. Le premier des deux octets, avec une adresse d'offset paire, contient le code ASCII du caractère à afficher. Du fait de l'emploi de 8 bits pour définir ce code de caractère, 256 caractères différents peuvent être affichés. L'annexe G présente les 256 caractères du jeu de caractères du PC. Le code ASCII est suivi de l'octet d'attribut, toujours donc à une adresse d'offset impaire, qui définit l'apparence du caractère sur l'écran. Ce code est scindé par le contrôleur d'attribut en deux quartets, le quartet de plus fort poids (bits 4 à 7) définissant le fond du caractère, alors que le caractère de plus faible poids (bits 0 à) définit le premier plan du caractère. Ainsi sont obtenues deux valeurs entre 0 et 15, qui seront interprétées en fonction du type de moniteur connecté. Sur un moniteur couleur (avec une carte CGA ou EGA), ces deux valeurs servent à sélectionner une couleur parmi les 16 possibles. Chaque caractère de l'écran peut ainsi disposer d'une couleur de fond et d'une couleur de premier plan individuelles. Sur un moniteur monochrome (avec une carte graphique MDA, EGA ou Hercules), il n'est naturellement pas possible de sortir de couleurs. L'affichage normal des caractères peut donc, dans ce cas, uniquement être agrémenté par un affichage particulièrement clair, par un affichage en inversion vidéo ou souligné.

Pour accéder aux caractères de la RAM vidéo, il faut naturellement en connaître la disposition à l'intérieur de cette zone de mémoire. Cette disposition n'a pas été choisie

au hasard mais découle, au contraire, de façon très logique de la disposition des caractères sur l'écran.

C'est donc par le premier caractère de l'écran, le caractère situé dans le coin supérieur gauche de l'écran, que commence la RAM vidéo. Ce caractère figure donc à la position d'offset 0000h. Il est suivi dans la RAM vidéo, à la position d'offset 0002h, du caractère placé immédiatement à droite sur l'écran. Ainsi se suivent les 80 caractères de la première ligne de l'écran. Comme chaque caractère occupe 2 octets, ces 80 caractères occupent donc les 160 premiers octets de la RAM vidéo. Immédiatement après figure le premier caractère de la deuxième ligne de l'écran.

Chaque ligne de l'écran occupant 160 octets en mémoire, l'adresse de base d'une ligne dans la RAM vidéo peut être aisément obtenue, en multipliant le numéro de ligne (en comptant à partir de 0) par 160. Pour parvenir du début de chaque ligne à un caractère quelconque à l'intérieur de cette ligne, il suffit d'additionner à cette valeur la distance du caractère au début de la ligne. Chaque caractère occupant 2 octets, on multiplie donc le numéro de colonne (toujours en comptant à partir de 0) par 2, puis on additionne ce produit au produit précédent, et on obtient ainsi la position d'offset du caractère voulu dans la RAM vidéo. Ces calculs peuvent aisément être récapitulés dans une formule :

```
Position d'offset(Colonne,Ligne)=Ligne * 160 + Colonne * 2
```

Sur l'adresse d'offset ainsi décrite, on trouve le code ASCII du caractère souhaité - correspondant à un octet de son octet d'attribut ou de couleur.

Modes de texte étendus

Les cartes CGA, EGA et VGA ne connaissent pas seulement le mode texte 80*25 caractères auquel la formule précédente fait référence. En dehors du mode texte 40*25 caractères des cartes CGA, les cartes EGA et VGA soutiennent également les modes texte 80 caractères avec plus de 25 lignes de texte. Une carte EGA reconnaît 43 lignes, une carte VGA 50 lignes pouvant être converties en 43 lignes grâce à quelques astuces. Les cartes Super VGA soutiennent de nombreux modes de texte avec une résolution pouvant atteindre 132*80 caractères.

Dans ces modes également, la structure de la RAM vidéo reste identique à celle présentée plus haut. Mais la formule concernant le calcul de l'adresse d'offset doit s'apter à un caractère parce que dans la RAM vidéo les différentes lignes occupent plus de mémoire. Il suffit tout simplement d'adapter le facteur 160 en le remplaçant par le double de la largeur de lignes en caractères. En mode de texte 40*25 caractères de la carte CGA, il doit par exemple être de 80.

Lorsque vous développez des programmes, vous n'êtes pas obligé de faire attention à ces modes si vous ne les activez pas explicitement en programmant la carte vidéo. En

règle générale, un programme démarre en mode de texte 80*25 caractères à partir de l'interface DOS et ne doit reconnaître que ce mode.

Accès à diverses pages d'écran

Jusqu'à présent, la formule offset ignorait le fait que les cartes Hercules, CGA, EGA et VGA soutenaient plusieurs pages d'écran parce que la RAM vidéo contient plus de place qu'une page vidéo unique. Pour adresser plusieurs pages d'écran, il convient donc d'augmenter la formule en ajoutant l'offset de départ dans l'adresse d'offset de la page concernée.

Il ne faut pas oublier ici que les diverses pages d'écran sont organisées en mode de texte 80*25 caractères à l'intérieur de la RAM vidéo selon un intervalle de 4 Ko (4096 octets). Entre les différentes pages d'écran réclamant 4000 octets (80*25*2), il reste toutefois 96 octets inutilisés que vous pouvez utiliser à d'autres fins.

En mode de texte 80*25 caractères, la formule offset s'énonce comme suit, compte tenu de la page d'écran :

```
Position d'offset (Colonne,Ligne) = Ligne * 160 + Colonne * 2 + Page *
4096
```

où la première page d'écran porte le numéro 0.

En mode de texte 40*25 caractères, il faut remplacer le facteur 4096 par 2048 parce que dans ce cas les différentes pages d'écran se succèdent avec un intervalle de 2 Ko. La procédure est la même dans les modes de texte EGA et VGA où 43 à 50 lignes sont affichées sur l'écran. Mais ici le facteur est de 8.

Programmes d'exemple pour l'accès direct à la RAM vidéo en mode de texte

Les trois programmes d'exemple suivants ont tous pour but de réaliser une routine sortant une chaîne sur l'écran à l'aide d'un accès direct à la RAM vidéo. Bien que chaque programme soit très différent du fait des différences existant entre les langages BASIC, Pascal et C, les trois programmes se composent malgré tout des mêmes éléments fondamentaux.

Chaque programme comprend donc, en dehors de la routine de sortie elle-même, une sorte de routine d'initialisation, qui détermine l'adresse de segment de la RAM vidéo. Cette routine a pour cela recours à une astuce qui consiste à tester le contenu d'une variable dans la zone des variables du BIOS. Cette variable n'a, à première vue, aucun rapport avec l'adresse de segment de la RAM vidéo puisqu'elle contient l'adresse du

registre d'adresse du CRTC. En y regardant de plus près, on s'aperçoit cependant qu'il existe un rapport direct entre ces deux informations car ce registre figure toujours à l'adresse de port 3B4h sur les cartes monochromes, de même que la RAM vidéo figure toujours à l'adresse de segment B000h sur ces cartes. Cette relation s'applique également aux cartes vidéo, sur lesquelles le registre d'adresse du CRTC figure toujours à l'adresse de port 3D4h et la RAM vidéo à l'adresse de segment B800h. En déterminant l'adresse de port du registre d'adresse du CRTC on peut donc du même coup déduire l'adresse de segment de la RAM vidéo. Une fois cette adresse connue, elle est placée dans une variable globale et la routine d'initialisation est terminée.

Les trois programmes contiennent après cette routine la routine de sortie elle-même, qui se sert de l'adresse de segment déterminée précédemment pour accéder à la RAM vidéo. En outre, lors de chaque appel, cette routine recherche l'adresse de départ de la page écran actuellement affichée. Elle s'assure ainsi que les sorties apparaissent effectivement sur l'écran et n'aboutissent pas dans une page écran cachée. La routine utilise à cet effet une autre variable de la zone de variables du BIOS. Il s'agit de la variable CRT_START, qui figure à l'adresse 0040:004E et désigne l'adresse d'offset de la page écran affichée par rapport à l'adresse d'offset 0000h.

Une fois cette adresse connue, on peut accéder à la RAM vidéo. La façon dont cela est réalisé concrètement dépend cependant largement du langage utilisé. Examinons donc chaque programme en détail.

La réalisation en C

D'un point de vue purement informatique, il s'agit de la solution la plus élégante des trois car la RAM vidéo est ici traitée comme une variable tout à fait normale. A cet effet est tout d'abord définie la structure VELB, qui décrit la paire d'attributs ASCII exactement comme elle figure dans la RAM vidéo. On crée comme pointeur sur cette structure un nouveau type de données appelé VP. Il est essentiel à cet égard que ces pointeurs soient du type FAR car ces structures se trouvent à l'intérieur de la RAM vidéo et donc en dehors du segment de données du C. Avec les modèles de mémoire plus petits, avec un adressage NEAR, elles ne pourraient donc être appelées sans que le mot d'instruction FAR soit explicitement spécifié.

La variable globale VPTR est mise en place à l'intérieur de la routine d'initialisation INIT_DPRINT comme un pointeur sur la première paire d'attributs ASCII dans la page 0 de la RAM vidéo. Au sein de la routine de sortie au sens strict, la fonction DPRINT, elle servira de base à l'adressage des caractères à l'intérieur de la RAM vidéo.

Dans la fonction de sortie DPRINT, l'adresse de la position de sortie sur l'écran transmise est chargée dans le pointeur LPTR. A cet effet, le contenu de la variable globale VPTR est tout d'abord chargé dans ce pointeur, après quoi l'adresse d'offset de la page écran affichée (tirée de la variable dans la zone BIOS) est additionnée à ce contenu. Il ne faut pas oublier à ce propos que le contenu des variables BIOS doit être casté dans un pointeur BYTE car le contenu de la variable BIOS ne correspond pas à la structure VELB mais à des octets. A défaut d'un opérateur CAST approprié, le compilateur C

produirait un code qui, avant l'addition, multiplierait le contenu de la variable BIOS par la longueur de la structure VELB (2 octets) et fournirait donc un résultat erroné.

Au pointeur ainsi calculé doit encore être ajoutée la position de sortie qui a été communiquée à la fonction DPRINT sous forme de coordonnées de ligne et de colonne. La RAM vidéo est à cet effet considérée comme une sorte de vecteur dont les 2000 éléments sont constitués par la structure VELB. Comme l'adresse de base de ce vecteur a déjà été déterminée dans LPTR, il ne reste plus qu'à calculer l'index de ce vecteur. Ce calcul s'effectue en multipliant l'ordonnée de ligne par 80 (colonnes par ligne) et en additionnant au résultat l'ordonnée de colonne. Le résultat final est un pointeur sur la position de sortie dans la RAM vidéo, qui peut être traité comme n'importe quel autre pointeur C.

Ce pointeur est alors utilisé pour parcourir la chaîne à sortir : lors de chaque parcours de la boucle, le pointeur est fixé sur la prochaine structure VELB et donc sur la prochaine paire d'attributs ASCII dans la RAM vidéo. Il sert à y écrire la couleur transmise à la fonction DPRINT et le code ASCII du caractère de la chaîne transmise actuellement traité. L'opération se répète jusqu'à ce que le dernier caractère de la chaîne, le caractère de fin soit atteint.

Listing : DVIC.C

```
/*****************************************************************************
*                              D V I D E O                                   *
*----------------------------------------------------------------------------*
*   Fonction       : Démonstration de l'accès direct à la                    *
*                    mémoire d'écran                                         *
*----------------------------------------------------------------------------*
*   Auteur         : MICHAEL TISCHER                                         *
*   Développé le   : 01/10/1988                                              *
*   Dernière MAJ   : 01/02/1992                                              *
*----------------------------------------------------------------------------*
*   (MICROSOFT C)                                                            *
*   Compilation    : CL /AS DVIC.C                                           *
*   Appel          : DVIC                                                    *
*----------------------------------------------------------------------------*
*   (BORLAND TURBO C)                                                        *
*   Compilation    : par la commande RUN de l'EDI                            *
*                    (sans fichier de projet )                               *
*****************************************************************************/

/*--- Fichiers en-tête ---------------------------------------------------*/

#include <dos.h>
#include <stdlib.h>
#include <string.h>
#include <stdarg.h>
#include <bios.h>

/*--- Typedef ------------------------------------------------------------*/

typedef unsigned char   BYTE;           /* Bricolage d'un type BYTE */
typedef struct velb far *VP;    /* Pointeur FAR sur la mémoire écran */
typedef BYTE            BOOL;           /* Comme BOOLEAN en Pascal */

/*--- Structures ---------------------------------------------------------*/

struct velb
{                       /* Décrit une position de l'écran en 2 octets */
  BYTE caractere,                               /* Le code ASCII */
       attribut;                        /* L'attribut correspondant */
};

/*--- Macros -------------------------------------------------------------*/

/*-- MK_FP forme un pointeur FAR sur un objet à partir d'une ---------*/
/*-- adresse de segment et d'une adresse d'offset --------------------*/

#ifndef MK_FP                   /* MK_FP n'a pas encore été défini ? */
  #define MK_FP(seg, ofs) ((void far *)(((unsigned long)(seg)<<16)|(ofs)))
#endif

#define COULEUR(PP, FD)   ((PP << 3) + FD)

/*--- Constantes ---------------------------------------------------------*/

#define TRUE  (0==0)            /* Constantes pour travailler avec BOOL */
#define FALSE !TRUE

/*-- Les constantes suivantes fournissent des pointeurs --------------*/
/*-- du segment des variables du BIOS à partir du segment 0x40 -------*/

#define CRT_START       ((unsigned far *) MK_FP(0x40, 0x4E))
#define ADDR_6845       ((unsigned far *) MK_FP(0x40, 0x63))

#define NORMAL          0x07            /* Attributs de caractère */
#define CLAIR           0x0F            /* pour une carte d'écran */
#define INVERSE         0x70                    /* monochrome */
#define SOULIGNE        0x01
#define CLIGNOTANT      0x80

#define NOIR            0x00        /* Attributs couleur couleur */
#define BLEU            0x01
#define VERT            0x02
#define CYAN            0x03
#define ROUGE           0x04
#define MAGENTA         0x05
#define BRUN            0x06
#define GRISCLAIR       0x07
#define GRISFONCE       0x01
#define BLEUCLAIR       0x09
#define VERTCLAIR       0x0A
#define CYANCLAIR       0x0B
#define ROUGECLAIR      0x0C
#define MAGENTACLAIR    0x0D
#define JAUNE           0x0E
#define BLANC           0x0F

/*--- Variables globales -------------------------------------------------*/

VP      vptr;           /* Premier caractère de la mémoire d'écran */
```

```
/************************************************************************
*  Fonction         : D P R I N T                                       *
**----------------------------------------------------------------------**
*  Fonction         : Affiche une chaîne dans la mémoire d'écran.       *
*                                                                       *
*  Param. en entrée : - COLONNE  = La colonne de sortie.                *
*                     - LIGNE    = La ligne de sortie.                  *
*                     - COULEUR  = Attribut pour les caractères.        *
*                     - STRING   = Pointeur sur la chaîne.              *
*  Valeur de retour : Aucune                                            *
************************************************************************/

void dprint(BYTE colonne, BYTE ligne, BYTE couleur, char * string)
{
  register VP    lptr;                       /* Pointeur de comptage */
  register BYTE  i;                        /* Compteur de caractères */

 /*-- Pointeur sur la position de sortie dans la mémoire d'écran -----*/
  lptr = (VP) ((BYTE far *) vptr + *CRT_START) + ligne * 80 + colonne;
  for( i=0 ; *string ; ++lptr, ++i )           /* Parcourt la chaîne */
  {
    lptr->caractere = *(string++);        /* Transfère le caractère */
    lptr->attribut = couleur;                    /* Fixe l'attribut */
  }
}

/************************************************************************
*  Fonction         : I N I T _ D P R I N T                             *
**----------------------------------------------------------------------**
*  Fonction         : Détermine le segment de la mémoire d'écran        *
*                     pour DPRINT:                                      *
*  Param. en entrée : Aucun                                             *
*  Valeur de retour : Aucune                                            *
*  Infos            : L'adresse de segment de la mémoire d'écran        *
*                     est placée dans la variable globale VPTR.         *
************************************************************************/

void init_dprint()
{
  vptr = (VP) MK_FP( (*ADDR_6845 == 0x3B4) ? 0xB000 : 0xB800, 0 );
}

/************************************************************************
*  Fonction         : C L S                                             *
**----------------------------------------------------------------------**
*  Fonction         : Efface l'écran à l'aide de DPRINT.                *
*                                                                       *
*  Param. en entrée : - COULEUR    = Attribut pour les caractères.      *
*  Valeur de retour : Aucune                                            *
************************************************************************/

void cls( BYTE couleur )
{
  static char lignevide[81] =
  { ' ',' ',' ',' ',' ',' ',' ',' ',' ',' ',' ',' ',' ',' ',' ',' ',
    ' ',' ',' ',' ',' ',' ',' ',' ',' ',' ',' ',' ',' ',' ',' ',' ',
    ' ',' ',' ',' ',' ',' ',' ',' ',' ',' ',' ',' ',' ',' ',' ',' ',
    ' ',' ',' ',' ',' ',' ',' ',' ',' ',' ',' ',' ',' ',' ',' ',' ',
    ' ',' ',' ',' ',' ',' ',' ',' ',' ',' ',' ',' ',' ',' ',' ',' ','\0'
  };

  register BYTE     i;                     /* Compteur d'itérations */

  for( i=0; i<24; ++i )            /* Parcourt les différentes lignes */
    dprint(0, i, couleur, lignevide);      /* Affiche une ligne vierge*/
}

/************************************************************************
*  Fonction         : N O K E Y                                         *
**----------------------------------------------------------------------**
*  Fonction         : Teste si une touche a été actionnée.              *
*  Param. en entrée : Aucun                                             *
*  Valeur Return    : TRUE si une touche a été actionnée, sinon         *
*                     FALSE.                                            *
************************************************************************/

BOOL nokey()
{
#ifdef __TURBOC__                                 /* Est-on en TURBO-C ? */
  return( bioskey( 1 ) == 0 );  /* OUI, teste le clavier avec le BIOS */
#else                                /* On travaille avec Microsoft C */
  return( _bios_keybrd( _KEYBRD_READY ) == 0 );  /* Test avec le BIOS */
#endif
}

/************************************************************************/
/**                          PROGRAMME PRINCIPAL                       **/
/************************************************************************/

void main()
{
 BYTE firstcol,          /* Couleur pour le premier carré sur l'écran */
      couleur,                      /* Couleur pour le carré courant */
      colonne,                      /* Position d'affichage actuelle */
      ligne;

  init_dprint();        /* Détermine le segment de la mémoire d'écran */
  cls( COULEUR(NOIR, VERT) );                        /* Efface l'écran */
  dprint(22, 0, BLANC, "DVIDEO  - (c) 1988.92 by Michael Tischer");
  firstcol = NOIR;                            /* Commencer par le NOIR */

  while( nokey() )              /* Attend qu'une touche soit actionnée */
  {
    if (++firstcol > BLANC)              /* Dernière couleur atteinte ? */
      firstcol = BLEU;                    /* OUI, on continue avec BLEU */
    couleur = firstcol;      /* Fixe la première couleur sur l'écran */

  /*-- Remplit l'écran avec des pavés --------------------------------*/

    for ( colonne=0; colonne < 80; colonne += 4)
      for (ligne=1; ligne < 24; ligne += 2)
      {
        dprint( colonne, ligne,   couleur, "");
        dprint( colonne, ligne+1, couleur, "");
        couleur = ++couleur & 15;
      }
  }
}
```

La réalisation en Pascal

A l'aide du mot d'instruction ABSOLUTE ou en intégrant une petite routine assembleur, il serait possible, sous Turbo Pascal également, de traiter la RAM vidéo comme une variable normale. Turbo Pascal offre cependant une autre possibilité encore plus simple.

Pour accéder à des zones de mémoire situées en dehors du segment de données du programme Turbo Pascal mais dont les adresses de segment et d'offset sont connues, Turbo Pascal met à votre disposition les tableaux MEMW et MEM. Le tableau MEM se compose d'octets, le tableau MEMW de mots. Ces deux tableaux n'existent pas vraiment, ils sont simplement plaqués virtuellement sur la totalité de la mémoire, mais cela n'entame en rien leur utilité pratique.

Vous pouvez aussi bien écrire des valeurs dans l'un des deux tableaux que lire les valeurs du tableau, grâce aux instructions suivantes :

```
MEMW[ Adresse de segment : Adresse d'offset ] := Expression
```

ou

```
Variable := MEMW[ Adresse de segment : Adresse d'offset ]
```

Il semblerait plus logique, en l'occurrence, de travailler avec le tableau MEM puisque la procédure de sortie travaille chaque fois avec un caractère ASCII différent alors que l'attribut reste constant. La procédure de sortie DPrint a pourtant recours au tableau MEMW car sur des ordinateurs 16 bits les accès à la mémoire sur 16 bits sont exécutés plus rapidement que deux accès 8 bits consécutifs.

Pour l'accès au tableau MEMW, l'adresse de segment de la RAM vidéo est tirée par DPrint de la variable VSeg, qui est initialisée au début du programme par la procédure InitDPrint. Comme nous l'avons indiqué plus haut, cette procédure utilise pour cela le contenu de la variable BIOS contenant l'adresse de port du registre d'adresse du CRTC. La déclaration de cette variable, comme celle des autres variables BIOS utilisées à l'intérieur de DPrint, s'effectue de façon très commode en Turbo Pascal, grâce à l'existence du mot d'instruction ABSOLUTE car les deux variables peuvent ainsi être manipulées à l'intérieur du programme comme n'importe quelle autre variable globale.

Pour l'accès à la RAM vidéo dans le cadre de la procédure de sortie DPrint, l'adresse d'offset à l'intérieur du tableau MEMW est calculée à partir de l'adresse de départ de la page écran et des coordonnées transmises, après multiplication de l'ordonnée de ligne par 160 et de l'ordonnée de colonne par 2. Cette adresse d'offset est ensuite augmentée de 2 au fur et à mesure du parcours de la chaîne à sortir. Elle avance ainsi chaque fois d'une paire ASCII-attribut vers la droite.

Notez toutefois que Turbo Pascal utilise WRITE ou des procédures similaires lors des sorties pour accéder directement à l'écran, dès l'instant où on intègre l'unité CRT dans un programme sans fixer explicitement la variable DIRECTVIDEO sur FALSE.

Listing : DVIP.PAS

```
{*********************************************************************}
{*                          D V I P                                  *}
{*-------------------------------------------------------------------*}
{*   Fonction       : Montre comment accéder directement à la        *}
{*                    mémoire d'écran en Turbo Pascal.               *}
{*-------------------------------------------------------------------*}
{*   Auteur         : MICHAEL TISCHER                                *}
{*   Développé le   :  2/10/1987                                     *}
{*   Dernière MAJ   : 14/01/1991                                     *}
{*********************************************************************}

program DVIP;

Uses Crt, Dos;                          { Intègre les unités CRT et DOS }

const NORMAL      = $07;        { Définit les attributs de caractère }
      CLAIR       = $0f;                  { pour une carte d'écran }
      INVERSE     = $70;                        { monochrome }
      SOULIGNE    = $01;
      CLIGNOTANT  = $80;

      NOIR        = $00;                  { Attributs de couleur }
      BLEU        = $01;
      VERT        = $02;
      CYAN        = $03;
      ROUGE       = $04;
      MAGENTA     = $05;
      BRUN        = $06;
      GRISCLAIR   = $07;
      GRISFONCE   = $01;
      BLEUCLAIR   = $09;
      VERTCLAIR   = $0A;
      CYANCLAIR   = $0B;
      ROUGECLAIR  = $0C;
      MAGENTACLAIR = $0D;
      JAUNE       = $0E;
      BLANC       = $0F;

type TextTyp = string[80];

var VSeg : word;                { Adresse de segment de la mémoire d'écran }

{*********************************************************************}
{* InitDPrint: Détermine le segment de la mémoire d'écran pour DPrint *}
```

```
{* Entrée : Aucune                                                    *}
{* Sortie : Aucune                                                    *}
{**********************************************************************}

procedure InitDPrint;

var CRTC_PORT : word absolute $0040:0063;{ variable dans seg. var. BIOS }

begin
  if CRTC_PORT = $3B4 then          { Adaptateur monochrome connecté ? }
    VSeg := $B000                { OUI, la mémoire d'écran est en B000:0000 }
  else                           { Non, ce doit être un adaptateur couleur }
    VSeg := $B800;                 { La mémoire d'écran est en B800:0000 }
end;

{**********************************************************************}
{* DPrint: Ecrit une chaine directement dans la mémoire d'écran.      *}
{* Entrée : - COLONNE : colonne d'affichage                           *}
{*          - LIGNE   : ligne d'affichage                             *}
{*          - COULEUR : couleur (ou attribut) des caractères.         *}
{*          - STROUT  : chaîne à afficher                             *}
{* Sortie : Aucune                                                    *}
{**********************************************************************}

procedure DPrint( Colonne, Ligne, Couleur : byte; StrOut : TextTyp);

var PAGE_OFS : word absolute $0040:$004E;  { Variable seg. var. BIOS }
    Offset   : word;  { Pointeur sur la position d'affichage courante }
    i, j     : byte;                          { Compteurs d'itérations }
    Attribut : word;                           { Attribut d'affichage }

begin
  Offset := Ligne * 160 + Colonne * 2 + PAGE_OFS;
  Attribut := Couleur shl 8;{ Octet fort pour accès à la mémoire écran }
  i := length( StrOut );          { Détermine la longueur de la chaîne }
  for j:=1 to i do                              { Parcourt la chaîne }
    begin { Ecrit caractère et attribut directement dans mémoire écran }
      memw[VSeg:Offset] := Attribut or ord( StrOut[j] );
      Offset := Offset + 2;  { Passe au prochain couple ASCII-attribut }
    end;
end;

{**********************************************************************}
{* Démo   : Démonstration du fonctionnement de DPrint.                *}
{* Entrée : Aucune                                                    *}
{* Sortie : Aucune                                                    *}
{**********************************************************************}

procedure demo;

var Colonne,                              { Position d'affichage courante }
    Ligne,
    Couleur   : integer;

begin
  TextBackGround( NOIR );                      { Colore le fond en noir }
  ClrScr;                                                { Efface l'écran }
  DPrint( 22, 0, BLANC, 'DVIP  - (c) 1988, 1992 by Michael Tischer');
  Randomize;              { Active le générateur de nombres aléatoires }
  while not KeyPressed do  { Répète opération jusqu'à touche actionnée }
    begin
      Colonne := Random( 76 );              { Colonne, ligne et couleur }
      Ligne := Random( 22 ) + 1;              { sont choisis au hasard }
      Couleur := Random( 14 ) + 1;
      DPrint( Colonne, Ligne,   Couleur, '');    { Affiche un pavé }
      DPrint( Colonne, Ligne+1, Couleur, '');
    end;
  ClrScr;                                  { Efface à nouveau l'écran }
end;

{**********************************************************************}
{**                        PROGRAMME PRINCIPAL                       **}
{**********************************************************************}

begin
  InitDPrint;                    { Initialise l'affichage par DPrint }
  Demo;                        { Démontre le fonctionnement de DPrint }
end.
```

La réalisation en BASIC

La version BASIC n'est pas satisfaisante car elle est encore plus lente que l'instruction PRINT, qui est déjà suffisamment lente. Nous ne vous la proposons que par souci d'exhaustivité, dans la mesure où elle constitue tout de même un bel exemple de la façon d'accéder à la totalité de la mémoire adressable par le processeur 8088.

Ce sont les instructions DEF SEG, PEEK et POKE qui constituent le coeur de l'accès à la mémoire.

DEF SEG sert à définir l'adresse de segment du segment 64 Ko "actuel", alors que PEEK et POKE permettent de lire ou d'écrire des octets dans ce segment. La routine d'initialisation à partir de la ligne numéro 50000 a recours à cette technique pour définir tout d'abord le segment de variables du BIOS comme le segment actuel. Elle lit ensuite à cet endroit l'adresse de port du registre d'adresse CRTC à l'aide de deux instructions PEEK et calcule sur la base de cette information l'adresse de segment de la RAM vidéo qu'elle affecte à la variable VR.

A l'intérieur de la routine de sortie elle-même, cette technique est également utilisée à partir de la ligne numéro 51000 pour définir avec DEF SEG la RAM vidéo comme le segment actuel. Auparavant est toutefois calculée l'adresse d'offset sur la RAM vidéo, en lisant l'adresse de départ de la page écran actuelle dans la zone de variables du BIOS et en y additionnant l'adresse d'offset de la position de sortie à l'intérieur de la RAM vidéo. Comme dans l'exemple en Pascal, l'adresse d'offset de la position de sortie est obtenue en additionnant à l'ordonnée de ligne (variable LIGNE%) multipliée par 160 l'ordonnée de colonne (COLONNE%) multipliée par 2.

Listing : DVIB.BAS

```
'*************************************************************************
'*                                DVIB                                   *
'*-----------------------------------------------------------------------*
'*  Fonction     : montre comment accéder directement à la mémoire écran *
'*-----------------------------------------------------------------------*
'*  Auteur       : MICHAEL TISCHER                                       *
'*  Développé le : 06.05.1991                                            *
'*  Dernière MAJ : 06.05.1991                                            *
'*************************************************************************
DECLARE SUB InitDPrint ()
DECLARE SUB Demo ()
DECLARE SUB DPrint (Colonne%, Ligne%, Couleur%, StrOut AS STRING)

CONST NORMAL = &H7          'Définit les attributs d'un écran monochrome
CONST CLAIR = &HF
CONST INVERSE = &H70
CONST SOULIGNE = &H1
CONST CLIGNOTANT = &H80

CONST NOIR = &H0            'Définit les couleurs d'une carte couleur
CONST BLEU = &H1
CONST VERT = &H2
CONST CYAN = &H3
CONST ROUGE = &H4
CONST MAGENTA = &H5
CONST BRUN = &H6
CONST GRISCLAIR = &H7
CONST GRISFONCE = &H8
CONST BLEUCLAIR = &H9
CONST VERTCLAIR = &HA
CONST CYANCLAIR = &HB
CONST ROUGECLAIR = &HC
CONST MAGENTACLAIR = &HD
CONST JAUNE = &HE
CONST BLANC = &HF

DIM SHARED VSeg AS LONG                 'Segment de la mémoire d'écran

CALL InitDPrint                         'Initialise la sortie par DPrint
CALL Demo                               'Démonstration de DPrint
END

'*************************************************************************
'* Demo    : Démonstration de DPrint                                     *
'* Entrée  : néant                                                       *
'* Sortie  : néant                                                       *
'*************************************************************************
SUB Demo

DIM Colonne AS INTEGER                          'Colonne d'affichage
DIM Ligne AS INTEGER                            'Ligne d'affichage
DIM Couleur AS INTEGER                          'Attribut de l'affichage

RANDOMIZE TIMER             'Initialise le générateur de nombre aléatoire
IF VSeg = &HB800 THEN           'A-t-on branché un adaptateur couleur ?
  CLS   'Efface l'écran
  CALL DPrint(22, 0, BLANC, "DVIB - (c) 1988, 1992 by Michael Tischer ")
  DO
    Colonne = INT(76 * RND)                     'Colonne au hasard
    Ligne = INT(22 * RND) + 1                   'Ligne au hasard
    Couleur = INT(14 * RND) + 1                 'Couleur au hasard
    CALL DPrint(Colonne, Ligne, Couleur, "")    'Affiche un pavé
  LOOP UNTIL INKEY$ <> ""  'répète l'opération jusqu'à frappe une touche
ELSE                                    'On a un adaptateur monochrome
  CLS                                           'Efface l'écran
  CALL DPrint(22, 0, INVERSE, "DVIB - (c) 1988, 92 by Michael Tischer ")
  DO
    Colonne = INT(76 * RND)                     'Colonne au hasard
    Ligne = INT(22 * RND) + 1                   'Ligne au hasard
    SELECT CASE INT(4 * RND)                    'Attribut au hasard
      CASE 0
        Couleur = NORMAL
      CASE 1
        Couleur = CLAIR
      CASE 2
        Couleur = INVERSE
      CASE 3
        Couleur = CLIGNOTANT OR INVERSE         'plus visible
    END SELECT
    CALL DPrint(Colonne, Ligne, Couleur, "")    'Affiche un pavé
  LOOP UNTIL INKEY$ <> ""  ' répète opération jusqu'à frappe une touche
END IF
END SUB

'*************************************************************************
'* DPrint  : Ecrit une chaîne directement dans la mémoire d'écran        *
'* Entrée  :  - Colonne : colonne d'affichage                            *
'*            - Ligne   : ligne d'affichage                              *
'*            - Couleur : Couleur(Attribut) des caractères               *
'*            - SrtOut  : Chaîne à afficher                              *
'* Sortie  : néant                                                       *
'*************************************************************************
SUB DPrint (Colonne%, Ligne%, Couleur%, StrOut AS STRING)

DIM Offset AS INTEGER               'Offset où il faut écrire le caractère
DIM Compteur AS INTEGER                         'Compteur d'itérations

DEF SEG = &H40                  'Segment de la zone des variables du BIOS
Offset = PEEK(&H4E) + PEEK(&H4F) * 256      'Adresse de début de la page
Offset = Offset + Ligne% * 160 + Colonne% * 2   'Offset du premier car.
DEF SEG = VSeg                          'Segment de la mémoire d'écran
FOR Compteur = 1 TO LEN(StrOut)                 'parcourt la chaîne
  POKE Offset, ASC(MID$(StrOut, Compteur, 1))   'ASCII en mémoire écran
  POKE Offset + 1, Couleur%                     ' ainsi que la couleur
  Offset = Offset + 2                   'Offset du prochain caractère
NEXT
END SUB

'*************************************************************************
'* InitDPrint : Lit le segment pour DPrint                               *
'* Entrée     : néant                                                    *
'* Sortie     : Met le segment de la mémoire d'écran dans la variable    *
'*              globale VSeg                                             *
'*************************************************************************
SUB InitDPrint

DEF SEG = &H40                          'Segment des variables du BIOS
IF PEEK(&H63) + PEEK(&H64) * 256 = &H3B4 THEN   'Adaptateur monochrome
  VSeg = &HB000                 'La mémoire d'écran commence en B000:0000
ELSE                                    'Adaptateur couleur
  VSeg = &HB800                 'La mémoire d'écran commence en B800:0000
END IF
END SUB
```

4.5. La carte monochrome IBM (MDA)

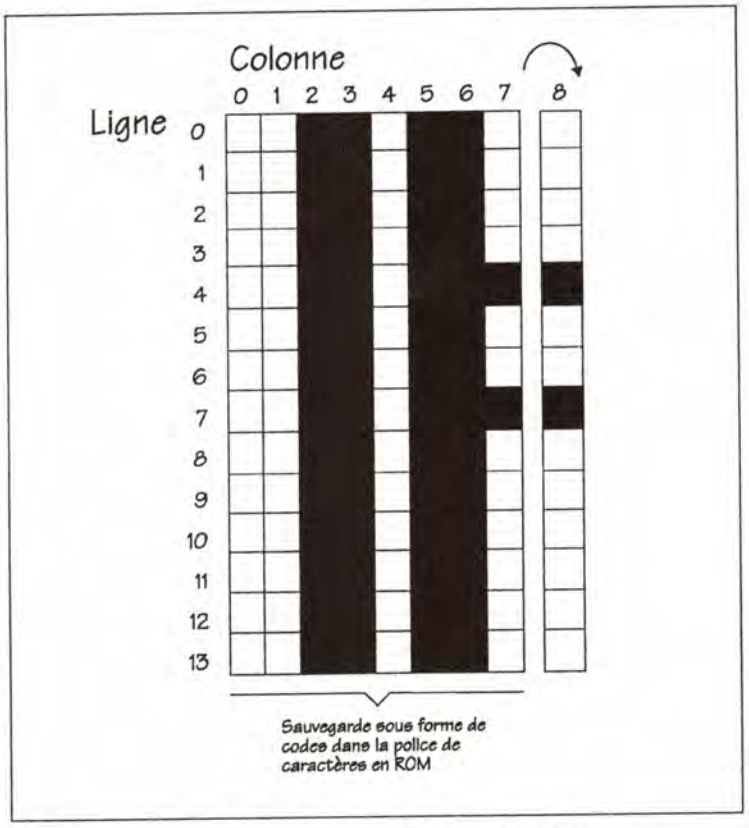
La carte MDA lancée avec le PC en 1981 est aujourd'hui rarement utilisée. Mais les standards qu'elle avait imposés ne sont pas encore morts. Ils survivent en effet sous la forme de Hercules Graphics Card qui est largement compatible avec la MDA. La carte MDA était très appréciée pendant les premiers jours du PC. Elle arrivait même à surpasser la carte CGA en raison de son prix peu élevé, mais aussi parce que l'image qu'elle reproduisait sur l'écran était d'une meilleure qualité. L'utilisateur qui pouvait se passer des couleurs et des techniques graphiques n'hésitait pas à choisir la carte MDA.

Cette section décrit en détail les performances de la carte MDA et explique le mode de programmation de ses différents registres et les tâches qu'ils accomplissent. Nous allons étudier les thèmes suivants :

- ✓ Affichage de texte avec une carte MDA
- ✓ Etat et taille de la RAM vidéo
- ✓ Construction de l'octet d'attribut en mode de texte
- ✓ Fonction des registres d'état et de contrôle
- ✓ Accès au contrôleur CRT

Affichage de texte avec une carte MDA

L'obtention d'une qualité d'affichage meilleure par rapport à la carte CGA résulte de l'utilisation d'une matrice 9*14. Elle permet d'afficher les caractères avec une résolution élevée. Le format de cette matrice est d'autant plus extraordinaire qu'un générateur de caractères contenant les modèles de bits de chaque caractère ne permet pas de produire des caractères d'une largeur de plus de 8 points. Or avec le jeu de caractères IBM, cela devrait avoir pour effet une légère disjonction entre les caractères de cadre horizontaux. C'est pourquoi un circuit a été implanté sur cette carte qui permet d'éviter cet inconvénient en copiant tout simplement le huitième point dans le neuvième point pour tous les caractères dont le code ASCII est compris entre B0h et DFh (Ce sont essentiellement des caractères de cadre). Pour les caractères de cadre qui doivent être reliés horizontalement au caractère suivant, on peut ainsi parvenir, en fixant le huitième point d'une ligne, à ce que le neuvième point soit également allumé.



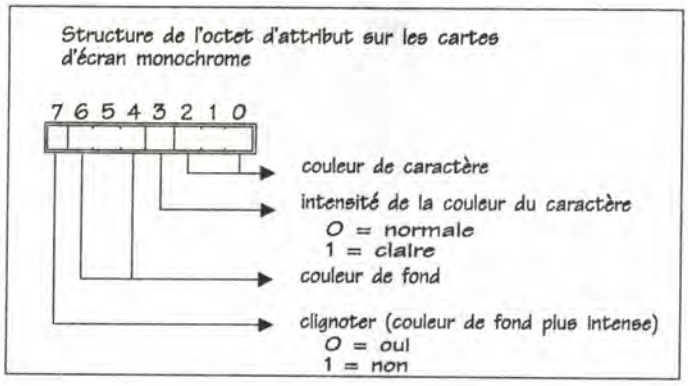
Affichage de caractère sur une carte monochrome dans une matrice 9x14

Le générateur de caractères n'a donc pas besoin de plus d'un octet par ligne de caractère puisque chacun des huit points est codé à l'aide d'un bit. Il aura donc besoin de 14 octets pour un caractère. La capacité mémoire requise pour le jeu de caractères complet sera donc d'un peu moins de 4 Ko, qui sont implantés dans un circuit ROM de la carte. Pour une raison assez mystérieuse, cette carte ne comporte toutefois non pas 4 Ko mais bien 8 Ko, de sorte que 4 Ko sont inutilisés.

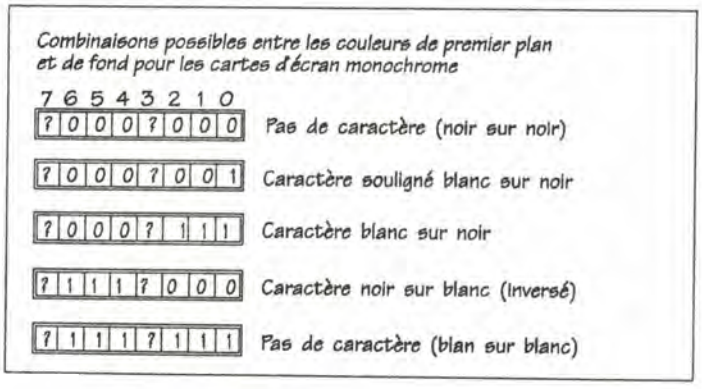
La RAM vidéo de la carte commence à l'adresse B000:0000 et elle occupe 4 Ko (4096 octets). Comme l'écran ne dispose que de 2000 emplacements de caractères, 4000 octets seulement sont utilisés pour le stockage de ces caractères et les 96 octets en queue de la RAM vidéo sont inutilisés et à votre entière disposition.

Structure de l'octet d'attribut en mode de texte

Comme le montre la figure suivante, sur la carte d'écran monochrome, les bits 0 à 2 et 4 à 6 de l'octet d'attribut d'un caractère définissent respectivement les couleurs de premier plan et de fond du caractère.



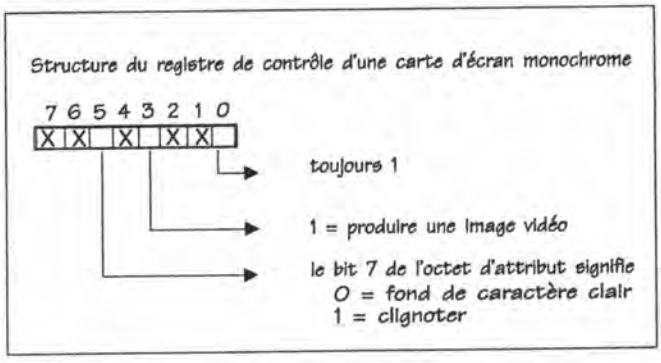
N'importe quelle combinaison peut être chargée dans ces bits, mais contrairement à la carte CGA, la carte MDA ne reconnaît que les combinaisons prédéfinies suivantes :



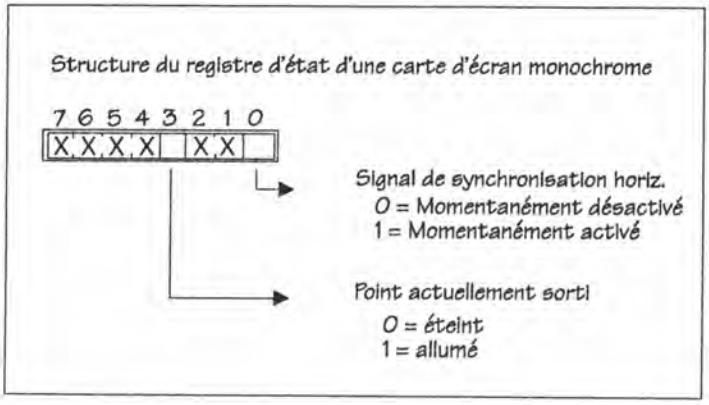
En dehors de ces combinaisons de bits, le bit 3 et le bit 7 de l'octet d'attribut peuvent encore être fixés individuellement. Le bit 3 fixe l'intensité de l'affichage du premier plan. S'il est mis, le caractère sera affiché avec une intensité particulière de la couleur de premier plan. La signification du bit 7 dépend dépend du contenu du registre dit de contrôle, que nous allons bientôt découvrir. Suivant le réglage opéré dans ce registre, ce bit aura pour effet de faire clignoter le caractère ou de le faire afficher avec une intensité particulière de la couleur de fond.

Fonction des registres d'état et de contrôle

Outre les registres déjà présentés du CRTC, la carte d'écran monochrome dispose encore de deux autres registres, le registre de contrôle et le registre d'état.



Le registre de contrôle, qui figure sur le port 3B8h, sert à contrôler différentes fonctions de la carte d'écran monochrome. Comme le montre la figure ci-dessus, seuls les bits 0, 3 et 5 jouent un rôle. Le bit 0 commande la résolution de l'écran. Bien que la carte ne soutienne qu'une seule résolution (25 lignes sur 80 colonnes), ce bit doit être fixé sur 1 lors de l'initialisation du système général, faute de quoi l'ordinateur sombrerait dans une boucle d'attente perpétuelle. Le bit 3 commande la mise en place d'une image visible sur l'écran. S'il est fixé sur 0, l'écran apparaît tout en noir et le curseur clignotant disparaît. S'il est à nouveau fixé sur 1, l'image revient sur l'écran. Le bit 5 a une fonction comparable puisqu'il sert à autoriser ou à interdire le clignotement de tous les caractères dont le bit 7 de l'octet d'attribut est mis. S'il contient la valeur 0, ces caractères ne clignoteront pas mais seront affichés sur une couleur de fond particulièrement claire, c'est-à-dire que le bit 7 de l'octet d'attribut fonctionnera alors comme bit d'intensité pour le fond. Il est malheureusement seulement possible d'écrire dans ce registre mais impossible de le lire. Un programme ne pourra donc jamais déceler si l'écran est, par exemple, actuellement activé ou non. La valeur standard de ce registre est 29h, c'est-à-dire que les 3 bits significatifs sont fixés sur 1.



Seuls les bits 0 à 3 de ce registre sont utilisés mais tous les autres bits doivent toujours contenir la valeur 1. A l'inverse du registre de contrôle, ce registre peut être lu mais il n'est pas possible d'y écrire.

Le bit 3 contient la valeur du point écran (pixel) sur lequel est actuellement placé le rayon électronique. Un 1 signale ici que le point doit être visible sur l'écran, 0 que l'écran doit rester noir dans cet emplacement.

Accès au contrôleur CRT

Le registre d'adresse du 6845 figure sur la carte d'écran monochrome à l'adresse de port 3B4h, le registre de données à l'adresse de port 3B5h. Bien que ces registres figurent à des adresses consécutives, il n'est pas possible de sortir en une fois, sur le port 3B4h, le numéro du registre à adresser et le nouveau contenu de ce registre. Il faut en effet sortir ces deux valeurs à l'aide de deux instructions OUT 8 bits distinctes, entre lesquelles une petite pause doit être observée (5 ou 6 cycles d'horloge), pour que le CRTC puisse réagir à la sortie dans le registre d'adresse. Dans les programmes assembleur, cette pause est généralement obtenue en faisant suivre l'instruction OUT qui envoie le numéro au registre d'adresse le numéro du registre à adresser d'instructions de saut JMP $+2. Cette instruction ne change rien à l'exécution du programme, puisqu'elle donne le contrôle à l'exécution immédiatement suivante, mais elle permet de perdre le temps nécessaire au CRTC.

La programmation des registres du CRTC ne présente cependant d'intérêt sur cette carte que par rapport aux lignes de départ et de fin du curseur clignotant de l'écran et à sa position sur l'écran. Ces paramètres peuvent cependant être fixés plus aisément à l'aide des fonctions de l'interruption 10h du BIOS, qui présentent en outre l'avantage de n'être pas liées à un matériel particulier et de fonctionner également sans problème avec les autres cartes vidéo.

Pour tous les amateurs et bricoleurs souhaitant faire des expériences avec les différents paramètres de cette table, pour contempler au moins une fois dans leur vie un écran avec 81 colonnes ou avec 26 lignes, voici la définition de cette table pour le mode de texte 80x25 caractères.

Contenu des registres CRTC en mode de texte 80x25 caractères de la carte d'écran monochrome (MDA)

Registre	Signification	Contenu
00h	Total de caractères horizontalement	97
01h	Caractères affichés horizontalement	80
02h	Signal de synchronisation horizontal après ... caractères	82
03h	Durée du signal horizontal de synchronisation en caractères	15
04h	Total de caractères verticalement	25
05h	Nombre ajusté de caractères verticalement	6
06h	Caractères affichés verticalement	25
07h	Signal de synchronisation vertical après ... caractères	25
08h	Mode d'entrelacement	2
09h	Nombre de lignes de grille (de balayage) par ligne de l'écran	13
0Ah	Ligne de départ du curseur clignot.de l'écran	11
0Bh	Ligne de fin du curseur clignotant de l'écran	12
0Ch	Adresse de départ de la page écran affichée (octet de poids fort)	0
0Dh	Adresse de départ de la page écran affichée (octet de poids faible)	0
0Eh	Adresse de caractère du curseur clignotant de l'écran (octet de poids fort)	0
0Fh	Adresse de caractère du curseur clignotant de l'écran (octet de poids faible)	0
10h	Position du crayon optique (octet de poids fort)	*
11h	Position du crayon optique (octet de poids faible)	*

* Impossible sur la carte d'écran monochrome

Programme d'exemple

Vous trouverez dans le listing de programme suivant quelques routines pour l'accès rapide à la RAM vidéo et pour la programmation des registres du CRTC ainsi que des autres registres de la carte d'écran monochrome. Vous trouverez aussi, en dehors de ces routines, une routine de démonstration illustrant l'appel des principales routines du module.

Listing : VMONO.ASM

```
;*****************************************************************;
;*                          V M O N O                            *;
;*-------------------------------------------------------------- *;
;*  Fonction       : fournit quelques fonctions de base pour     *;
;*                   l'accès à l'écran monochrome.               *;
;*                                                               *;
;*-------------------------------------------------------------- *;
;*  Infos          : Toutes les fonctions divisent l'écran en    *;
;*                   colonnes 0 à 79 et lignes 0 à 24            *;
;*-------------------------------------------------------------- *;
;*  Auteur         : MICHAEL TISCHER                             *;
;*  développé le   : 11/08/1987                                  *;
;*  Dernière modif.: 01/04/1989                                  *;
;*-------------------------------------------------------------- *;
;*  Assemblage     : MASM VMONO;                                 *;
;*                   LINK VMONO;                                 *;
;*-------------------------------------------------------------- *;
;*  Appel          : VMONO                                       *;
;*****************************************************************;

;-- Constantes ---------------------------------------------------

REG_CONTR   = 0388h             ;Adr. de port du registre de contrôle
ADRESS_6845 = 0384h                     ;Registre d'adresse du 6845
DATA_6845   = 0385h                     ;Registre de données du 6845
VIO_SEG     = 0B000h                ;Adresse de segment de la RAM vidéo
CUR_START   = 10            ;N° registre CRTC: ligne de départ du curseur
CUR_END     = 11              ;N° registre CRTC: ligne de fin du curseur
CURPOS_HI   = 14              ;N° registre CRTC: octet fort pos. curseur
CURPOS_LO   = 15            ;N° registre CRTC: octet faible pos. curseur

DELAY       = 20000             ;Compteur de boucle pour boucle d'attente

;-- Pile ---------------------------------------------------------

stack   segment para stack              ;Définition du segment de pile

        dw 256 dup (?)                  ;La pile a une taille de 256 mots

stack   ends                                    ;Fin du segment de pile

;-- Données ------------------------------------------------------

data    segment para 'DATA'             ;Définition du segment de données

;-- Les données pour le programme de démonstration ---------------

str1    db "a" ,0
str2    db " >>> BIBLE PC <<< ",0
str3    db "   Fenêtre 1   ",0
str4    db "   Fenêtre 2   ",0
str5    db "          Le programme s'arrête lorsqu'on appuie sur "
        db "une touche....              ",0
initm   db 13,10
        db " VMONO (c) 1987,1989 by Michael Tischer "
        db 13,10,13,10
        db "Ce programme de démonstration ne tourne qu'avec une ",13,10
        db "carte d'écran monochrome (MDA). Si votre PC dispose",13,10
        db "d'un autre type de carte d'écran, veuillez maintenant",13,10
        db "entrer <S> pour arrêter le programme.",13,10
        db "Sinon veuillez frapper n'importe quelle autre touche",13,10
        db "pour lancer le programme ...",13,10,"$"

;-- Table avec les adresses d'offset des débuts de ligne ---------

lignes  dw  0*160, 1*160, 2*160, 3*160, 4*160, 5*160, 6*160
        dw  7*160, 8*160, 9*160,10*160,11*160,12*160,13*160
        dw 14*160,15*160,16*160,17*160,18*160,19*160,20*160
        dw 21*160,22*160,23*160,24*160

data    ends                                    ;Fin du segment de données

;-- Code ---------------------------------------------------------

code    segment para 'CODE'             ;Définition du segment de CODE

        assume cs:code, ds:data, es:data, ss:stack

;-- Il s'agit simplement du programme de démonstration -----------

demo    proc far

        mov  ax,data ;Rechercher adr. de segment du segment de données
        mov  ds,ax                              ;et charger dans DS
        mov  es,ax                              ;également dans ES

;-- Sortir message Init et attendre entrée -----------------------

        mov  ah,9               ;Numéro de fonction pour Sortir chaîne
        mov  dx,offset initm                    ;Adresse du message
        int  21h                        ;Appeler interruption du DOS

        xor  ah,ah              ;Numéro de fonction pour Lire touche
        int  16h                ;Appeler interruption clavier du BIOS
        cmp  al,"s"                             ;<s> entré ?
        je   fin                ;OUI --> Terminer programme à nouveau
        cmp  al,"S"                             ;<S> entré
        jne  startdemo                  ;NON --> Lancer démo

fin:    mov  ax,4C00h           ;N de fonction pour Terminer programme
        int  21h                        ;Appeler interruption DOS 21h
;-- Programme de démonstration de l'appel des fonctions ----------

startdemo label near
        mov  cx,0d00h                           ;Activer curseur plein
        call cdef
        call cls                                ;Vider l'écran

;-- Remplir l'écran avec caract. du jeu de caractères ASCII.

        xor  di,di              ;Commencer dans coin sup. gauche écran
        mov  si,offset str1             ;Adresse d'offset de chaîne 1
        mov  cx,2000            ;2000 caractères rentrent dans l'écran
        mov  al,07h                     ;Ecriture blanche sur fond noir
demo1:  call print                              ;Sortir chaîne
        inc  str1               ;Incr. caract. Ascii dans chaîne test
        jne  demo2                      ;Code NUL doit être interdit
        inc  str1
demo2:  loop demo1                              ;Répéter sortie

;-- Construire fenêtres 1 et 2 -----------------------------------
        mov  bx,0508h                   ;Coin sup. gauche fenêtre 1
        mov  dx,1316h                   ;Coin inf. droit fenêtre 1
        mov  ah,07h             ;Ecriture blanche sur fond noir
        call clear                      ;Vider fenêtre 1
        mov  bx,3C02h                   ;Coin sup. gauche fenêtre 2
        mov  dx,4A10h                   ;Coin inf. droit fenêtre 2
        call clear                      ;Vider fenêtre 2
        mov  bx,0508h                   ;Coin sup. gauche fenêtre 1
        call calo               ;Convertir en adresse d'offset
        mov  si,offset str3             ;Adresse d'offset chaîne 3
        mov  ah,70h             ;Ecriture noire sur fond blanc
        call print                      ;Sortir chaîne 3
        mov  bx,3C02h                   ;Coin sup. gauche fenêtre 2
        call calo               ;Convertir en adresse d'offset
        mov  si,offset str4             ;Adresse d'offset chaîne 4
        call print                      ;Sortir chaîne 4
        xor  di,di              ;Coin supérieur gauche de l'écran
        mov  si,offset str5             ;Adresse d'offset chaîne 5
        call print                      ;Sortir chaîne 5

;-- Sortir logo du programme -------------------------------------
        mov  bx,1E0Ch                   ;Colonne 30, ligne 12
        call calo               ;Convertir en adresse d'offset
        mov  si,offset str2             ;Adresse d'offset chaîne 2
        mov  ah,0F0h                    ;Clignoter en inversion
        call print                      ;Sortir chaîne 2

;-- Remplir les fenêtres de flèches ------------------------------
        xor  ch,ch              ;Octet fort du compteur sur 0
fleche: mov  bl,1                       ;Une étoile
fleche0: push bx                        ;Ranger BX sur la pile
        mov  di,offset str3     ;Ecrire ligne de flèche dans chaîne 3
        mov  cl,15              ;15 caractères en tout sur une ligne
        sub  cl,bl                      ;Calculer nombre d'espaces
        shr  cl,1               ;Diviser par 2 (Pour moitié gauche)
        or   cl,cl                      ;Pas d'espace ?
        je   fleche1                    ;OUI --> FLECHE1
        mov  al," "
        rep  stosb                      ;Ecrire espace dans chaîne 3
fleche1: mov  cl,bl             ;Nombre d'étoiles dans le compteur
        mov  al,"*"
        rep  stosb                      ;Ecrire étoile dans chaîne 3
        mov  cl,15              ;15 caractères en tout sur une ligne
        sub  cl,bl                      ;Calculer nombre d'espaces
        shr  cl,1               ;Diviser par 2 (pour moitié droite)
        or   cl,cl                      ;Pas d'espace ?
        je   fleche2                    ;OUI --> FLECHE2
        mov  al," "
        rep  stosb                      ;Ecrire espace dans chaîne 3
fleche2: mov  bx,0509h          ;Sous première ligne de fenêtre 1
        call calo               ;Convertir en adresse d'offset
        mov  si,offset str3             ;Adresse d'offset chaîne 3
        mov  ah,07h             ;Ecriture blanche sur fond noir
        call print                      ;Sortir chaîne 3
        mov  bx,3C10h           ;Dans ligne inférieure de fenêtre 2
```

```
        call calo                       ;Convertir en adresse d'offset
        call print                              ;Sortir chaine 3

;-- Faire une petite pause ----------------------------------------
        mov cx,DELAY                ;Charger dans compteur de boucle
attendre: loop attendre                 ;Compteur de boucle sur 0

;-- Scrolling de fenêtre 1 d'une ligne vers le bas ----------------
        mov bx,0509h                    ;Coin sup. gauche fenêtre 1
        mov dx,1316h                    ;Coin inf. droit fenêtre 1
        mov cl,1                                        ;Une ligne
        call scrolldn                   ;de défilement vers le bas

;-- Scrolling de fenêtre 2 d'une ligne vers le haut ---------------
        mov bx,3003h                    ;Coin sup. gauche fenêtre 2
        mov dx,4A10h                    ;Coin inf. droit fenêtre 2
        call scrollup                   ;Défilement vers le haut

;-- Une touche a-t-elle été actionnée ? (terminer le programme)

        mov ah,1                        ;N° fonction pour Tester touche
        int 16h                 ;Appeler interruption clavier du BIOS
        jne end_it              ;Touche actionnée --> Préparer fin prg

;-- NON. sortir ligne de flèches suivante -------------------------
        pop bx                                  ;Retirer BX de la pile
        add bl,2                ;2 étoiles de plus dans ligne suivante
        cmp bl,17                               ;17 atteint ?
        jne flecheO                     ;NON --> flèche suivante
        jmp fleche                      ;OUI --> nouvelle ligne

;-- Préparer fin du programme -------------------------------------
end_it:  xor ah,ah              ;Numéro de fonction pour Lire touche
        int 16h                 ;Appeler interruption clavier du BIOS

        mov cx,0000h                    ;Rétablir curseur normal
        call cdef
        call cls                                ;Vider l'écran
        jmp fin                         ;Sauter à la fin du programme

demo    endp

;== Viennent maintenant les fonctions proprement dites ==============

;-- SOUT: Désactive l'affichage écran -----------------------------
;-- Entrée   : Aucune
;-- Sortie   : Aucune
;-- Registres: AX et DX sont modifiés

sout    proc near

        mov dx,REG_CONTR  ;Adresse du registre de commande de l'écran
        in  al,dx                               ;En lire le contenu
        and al,11110111b                ;Bit 3 = 0 : Ecran désactivé
        out dx,al               ;Fixer nouvelle valeur (Ecran désactivé)

        ret                             ;Retour au programme d'appel

sout    endp

;-- SON: Active l'affichage écran ---------------------------------
;-- Entrée   : Aucune
;-- Sortie   : Aucune
;-- Registres: AX et DX sont modifiés

son     proc near

        mov dx,REG_CONTR    ;Adresse du registre de commande d'écran
        in  al,dx                               ;En lire le contenu
        or  al,8                        ;Bit 3 = 1 : Ecran activé
        out dx,al               ;Fixer nouvelle valeur (Ecran activé)
        ret                             ;Retour au programme d'appel

son     endp

;-- CDEF: Fixe les lignes de départ et de fin du curseur ----------
;-- Entrée   : CL = ligne de départ
;--            CH = ligne de fin
;-- Sortie   : Aucune
;-- Registres: AX et DX sont modifiés

cdef    proc near

        mov al,CUR_START                ;Registre : ligne de départ
        mov ah,cl                       ;Ligne de départ dans AH
        call setvc              ;Communiquer au contrôleur vidéo
        mov al,CUR_END                  ;Registre : ligne de fin
        mov ah,ch                       ;Ligne de fin dans AH
        jmp short setvc         ;Communiquer au contrôleur vidéo
cdef    endp

;-- SETBLINK: fixe le curseur clignotant de l'écran ---------------
;-- Entrée   : DI = Adresse d'offset du curseur(en unités de caractères)
;-- Sortie   : Aucune
;-- Registres: BX, AX et DX sont modifiés

setclign proc near

        mov bx,di                       ;Transférer offset dans BX
        mov al,CURPOS_HI    ;Registre : octet fort offset du curseur
        mov ah,bh                       ;Octet fort de l'offset
        call setvc                  ;Communiquer au contrôleur vidéo
        mov al,CURPOS_LO  ;Registre : octet faible offset du curseur
        mov ah,bl                       ;Octet faible de l'offset

;-- Le passage à SETVC se fait automatiquement --------------------
setclign endp

;-- SETVC: fixe un octet dans l'un des registres du contrôleur vidéo ---
;-- Entrée   : AL = Numéro du registre
;--            AH = Nouveau contenu du registre
;-- Sortie   : Aucune
;-- Registres: DX et AL sont modifiés

setvc   proc near

        mov dx,ADRESS_6845              ;Adresse du registre d'index
        out dx,al                       ;Envoyer numéro du registre
        jmp short $+2                           ;Petite pause I/O
        inc dx                          ;Adresse du registre d'index
        mov al,ah                               ;Contenu dans AL
        out dx,al                       ;Fixer nouveau contenu
        ret                             ;Retour au programme d'appel

setvc   endp

;-- GETVC: lit un octet dans l'un des registres du contrôleur vidéo ----
;-- Entrée   : AL = Numéro du registre
;-- Sortie   : AL = Contenu du registre
;-- Registres: DX et AL sont modifiés

getvc   proc near

        mov dx,ADRESS_6845              ;Adresse du registre d'index
        out dx,al                       ;Envoyer numéro du registre
        jmp short $+2                   ;Petite pause d'entrée/sortie
        inc dx                          ;Adresse du registre d'index
        in  al,dx                       ;Transférer contenu dans AL
        ret                             ;Retour au programme d'appel

getvc   endp

;-- SCROLLUP: fait défiler une fenêtre de N lignes vers le haut --------
;-- Entrée   : BL = ligne haut à gauche
;--            BH = colonne haut à gauche
;--            DL = ligne bas à droite
;--            DH = colonne bas à droite
;--            CL = nombre de lignes de défilement
;-- Sortie   : Aucune
;-- Registres: seuls les FLAGS sont modifiés
;-- Infos    : les lignes de l'écran libérées sont vidées

scrollup proc near

        cld                     ;Augmenter comptage pour instr. de chaine

        push ax                         ;Sauver tous les registres modifiés
        push bx                                         ;sur la pile
        push di                                 ;Dans ce cas l'ordre doit
        push si                                         ;être respecté !
        push bx                         ;Ces trois registres sont retirés de
        push cx                                 ;la pile avant même la fin de
        push dx                                         ;la routine
        sub dl,bl                               ;Calculer nombre de lignes
        inc dl
        sub dl,cl               ;Retrancher nombre de lignes de défilement
        sub dh,bh                               ;Calculer nombre colonnes
        inc dh
        call calo                       ;Convertir haut à gauche en offset
        mov si,di                               ;Ranger adresse dans SI
        add bl,cl                       ;Première ligne dans fenêtre décalée
        call calo                       ;Convertir première ligne en offset
        xchg si,di                                      ;Echanger SI et DI
        push ds                         ;Registre de segment sur la pile
        push es                                         ;sauver
        mov ax,VIO_SEG                  ;Adresse de segment de la RAM vidéo
        mov ds,ax                                       ;dans DS
        mov es,ax                                       ;et amener ES
supl:   mov ax,di                               ;Ranger DI dans AX
        mov bx,si                               ;Ranger SI dans BX
        mov cl,dh                       ;Nombre de colonnes dans le compteur
        rep movsw                               ;Décaler une ligne
        mov di,ax                               ;Retirer DI de AX
        mov si,bx                               ;Retirer SI de BX
        add di,160                      ;Fixer chaque fois sur ligne suivante
        add si,160
```

```
        dec  dl                         ;Toutes les lignes traitées ?
        jne  sup1                     ;NON --> décaler encore une ligne
        pop  es                  ;Retirer à nouveau registres de segment
        pop  ds                                          ;de la pile
        pop  dx                           ;Retirer coin inf. droit
        pop  cx                          ;Retirer nombre de lignes
        pop  bx                          ;Retirer coin sup. gauche
        mov  bl,dl                       ;Ligne inférieure dans BL
        sub  bl,cl                     ;Retrancher nombre de lignes
        inc  bl
        mov  ah,07h                      ;Couleur : noir sur blanc
        call clear                         ;Vider lignes libérées

        pop  si                            ;CX et DX ont déjà été
        pop  di                                        ;ramenés
        pop  bx
        pop  ax

        ret                              ;Retour au programme d'appel

scrollup endp

;-- SCROLLDN: fait défiler une fenêtre de N lignes vers le bas ---------
;-- Entrée   : BL = ligne haut à gauche
;--            BH = colonne haut à gauche
;--            DL = ligne bas à droite
;--            DH = colonne bas à droite
;--            CL = nombre de lignes de défilement
;-- Sortie   : Aucune
;-- Registres: seuls les FLAGS sont modifiés
;-- Infos    : les lignes de l'écran libérées sont vidées

scrolldn  proc near

        cld                         ;Augmenter comptage pour instr. chaîne

        push ax                       ;Sauver tous les registres modifiés
        push bx                                    ;sur la pile
        push di                          ;Dans ce cas l'ordre doit
        push si                                  ;être respecté !

        push bx                     ;Ces trois registres sont retirés de
        push cx                        ;la pile avant même la fin de
        push dx                                    ;la routine

        sub  dh,bh                          ;Calculer nombre colonnes
        inc  dh
        mov  al,bl                    ;Ranger ligne haut à gauche dans AL
        mov  bl,dl                       ;Ligne l.d. dans ligne l.g.
        call calo                     ;Convertir haut à gauche en offset
        mov  si,di                             ;Ranger adresse dans SI
        sub  bl,cl                  ;Retrancher nombre de l. de défilement
        call calo                     ;Convertir haut à gauche en offset
        xchg si,di                             ;Echanger SI et DI
        sub  dl,al                       ;Calculer nombre de lignes
        inc  dl
        sub  dl,cl              ;Retrancher nombre de lignes de défilement
        push ds                          ;Sauver registres de segment
        push es                                    ;sur la pile
        mov  ax,VIO_SEG           ;Adresse de segment de la RAM vidéo
        mov  ds,ax                                 ;dans DS
        mov  es,ax                               ;et amener ES
sdn1:   mov  ax,di                             ;Ranger DI dans AX
        mov  bx,si                             ;Ranger SI dans BX
        mov  cl,dh                 ;Nombre de colonnes dans le compteur
        rep  movsw                           ;Décaler une ligne
        mov  di,ax                           ;Retirer DI de AX
        mov  si,bx                           ;Retirer SI de BX
        sub  di,160               ;Fixer chaque fois sur ligne suivante
        sub  si,160
        dec  dl                         ;Toutes les lignes traitées ?
        jne  sdn1                     ;NON --> décaler encore une ligne
        pop  es                  ;Retirer à nouveau registres de segment
        pop  ds                                          ;de la pile
        pop  dx                           ;Retirer coin inf. droit
        pop  cx                          ;Retirer nombre de lignes
        pop  bx                          ;Retirer coin sup. gauche
        mov  dl,bl                       ;Ligne supérieure dans DL
        add  dl,cl                     ;Additionner nombre lignes
        dec  dl
        mov  ah,07h                      ;Couleur : noir sur blanc
        call clear                         ;Vider lignes libérées
        pop  si                            ;CX et DX ont déjà été
        pop  di                                        ;ramenés
        pop  bx
        pop  ax

        ret                              ;Retour au programme d'appel

scrolldn  endp

;-- CLS: Vider l'écran tout entier -------------------------------------
;-- Entrée   : Aucune
;-- Sortie   : Aucune
;-- Registres: seuls les FLAGS sont modifiés

cls     proc near

        mov  ah,07h                      ;Couleur est blanc sur noir
        xor  bx,bx                         ;Haut à gauche est (0/0)
        mov  dx,4F18h                      ;Bas à droite est (79/24)

;-- Le passage à Clear se fait automatiquement -------------------------

cls     endp

;-- CLEAR: remplit d'espaces une zone d'écran déterminée ---------------
;-- Entrée   : AH = attribut/couleur
;--            BL = ligne haut à gauche
;--            BH = colonne haut à gauche
;--            DL = ligne bas à droite
;--            DH = colonne bas à droite
;-- Sortie   : Aucune
;-- Registres: seuls les FLAGS sont modifiés

clear   proc near
        cld                     ;Augmenter comptage pour instr. de chaîne
        push cx                   ;Sauver sur la pile tous les registres
        push dx                       ;qui seront modifiés par la suite
        push si
        push di
        push es
        sub  dl,bl                          ;Calculer nombre de lignes
        inc  dl
        sub  dh,bh                          ;Calculer nombre colonnes
        inc  dh
        call calo                 ;Adresse d'offset du coin sup gauche
        mov  cx,VIO_SEG            ;Adresse de segment de la RAM vidéo
        mov  es,cx                                         ;dans ES
        xor  ch,ch                    ;Octet fort du compteur sur 0
        mov  al," "                                      ;Espace
clear1: mov  si,di                                 ;Ranger DI dans SI
        mov  cl,dh                ;Nombre de colonnes dans le compteur
        rep stosw                                    ;Sauver espace
        mov  di,si                                 ;Retirer DI de SI
        add  di,160                          ;Fixer dans ligne suivante
        dec  dl                           ;Toutes les lignes traitées ?
        jne  clear1                     ;NON --> vider encore une ligne
        pop  es                        ;Retirer de la pile les registres
        pop  di                                        ;sauvegardés
        pop  si
        pop  dx
        pop  cx
        ret                              ;Retour au programme d'appel

clear   endp

;-- PRINT: sort une chaîne sur l'écran ---------------------------------
;-- Entrée   : AH = Attribut/couleur
;--            DI = Adresse d'offset du premier caractère
;--            SI = Adresse d'offset de la chaîne par rapport à DS
;-- Sortie   : DI désigne position après dernier caractère sorti
;-- Registres: AL, DI et FLAGS sont modifiés
;-- Infos    : La chaîne doit être terminée par le caractère NUL.
;--            Les autres caractères de commande ne sont pas identifiés.

print   proc near

        cld                           ;Augmenter comptage pour instr. chaîne
        push si                           ;Sauver SI, DX et ES sur la pile
        push es
        push dx
        mov  dx,VIO_SEG           ;Adr. de seg. de la RAM vidéo d'abord
        mov  es,dx                                 ;dans DX puis dans ES
        jmp  print1                ;Lire premier caractère de la chaîne

print0: stosw                       ;Attribut et couleur dans RAM vidéo
print1: lodsb                      ;Lire caractère suivant de la chaîne
        or   al,al                                       ;Est-ce NUL
        jne  print0                                   ;NON --> Sortir

printe: pop  dx                        ;Retirer SI, DX et ES de la pile
        pop  es
        pop  si
        ret                              ;Retour au programme d'appel

print   endp

;-- CALO: convertit ligne et colonne en adresse d'offset ---------------
;-- Entrée   : BL = ligne
;--            BH = colonne
;-- Sortie   : DI = l'adresse d'offset
;-- Registres: DI et FLAGS sont modifiés

calo    proc near

        push ax                                  ;Sauver AX sur la pile
        push bx                                  ;Sauver BX sur la pile
```

```
;
;       shl  bx,1                       ;Colonne et ligne fois 2
;       mov  al,bh                              ;Colonne dans AL
;       xor  bh,bh                              ;Octet fort
;       mov  di,[lignes+bx]         ;Lire adresse d'offset de ligne
;       xor  ah,ah                  ;Octet fort pour offset colonne
;       add  di,ax              ;Additionner offset ligne et colonne
;
;       pop  bx                         ;Retirer BX de la pile
;
;       pop  ax                         ;Retirer AX de la pile
;       ret                     ;Retour au programme d'appel
;
;calo   endp
;
;;--- Fin ------------------------------------------------------------
;
;code   ends                             ;Fin du segment de CODE
;       end  demo           ;Commencer exécution du programme par DEMO
```

4.6. La carte graphique Hercules (HGC)

Dès 1982, un an seulement après l'apparition du PC sur le marché, une jeune entreprise d'informatique ambitieuse, la société américaine Hercules Computer Technology, eut l'idée de combler fort habilement une faille qu'IBM avait laissé se former sur le marché par une politique de marché inadaptée. Il n'existait en effet à l'époque que deux systèmes vidéo pour le PC. IBM diffusait d'une part un système monochrome qui ne permettait d'afficher sur l'écran que du texte, à l'exclusion de tout graphisme. Il y avait d'autre part un système couleur, qui permettait d'afficher aussi bien du texte que du graphisme, mais qui offrait une lisibilité nettement inférieure à celle de la carte monochrome. L'utilisateur qui souhaitait avoir la possibilité d'afficher des dessins sans renoncer pour autant à des applications orientées texte, comme le traitement de texte par exemple, se trouvait donc devant un dilemme car, du fait de la mauvaise qualité de l'image sur le système couleur, il était préférable de faire tourner ces applications sur un système monochrome. Pour nombre d'utilisateurs, la seule solution consistait à se doter des deux cartes vidéo, et donc également de deux moniteurs. Il s'agissait toutefois là d'une solution non seulement onéreuse mais anti-ergonomique car elle ne laissait plus guère subsister de place sur le bureau.

C'est pourquoi on imagina chez Hercules de combiner un affichage de texte excellent avec la possibilité d'affichage graphique. C'est ainsi que fut développée la Hercules Graphics Card. Sur un moniteur monochrome ordinaire, cette carte offre un mode de texte 80x25 caractères et un mode graphique d'une résolution de 720x348 points. Cette carte se tailla très vite un franc succès, au point qu'elle représente aujourd'hui, de fait, un standard industriel. Elle ouvrit la voie à une évolution qui a vu les fabricants de cartes vidéo indépendants occuper une part de marché toujours plus importante dans ce secteur, de sorte qu'IBM elle-même est maintenant largement distancée.

Les cartes graphiques Hercules, équipées généralement d'un port d'imprimante parallèle, continuent de jouer un rôle important dans la zone inférieure du PC.

Au cours de cette section consacrée à la programmation des cartes Hercules, vous étudierez les thèmes suivants :

- ✓ Absence du soutien BIOS
- ✓ La RAM vidéo de la carte Hercules
- ✓ Les registres de la carte Hercules
- ✓ Le contrôleur CRT de la carte Hercules
- ✓ Définition et programmation du mode graphique

Absence du soutien BIOS

La non reconnaissance du BIOS pèse lourd sur les épaules de la carte Hercules parce que le BIOS IBM ne travaille qu'en collaboration avec les cartes IBM MDA et CGA. En ce qui concerne l'affichage, la carte Hercules surmonte cet inconvénient puisqu'elle est totalement compatible avec la carte MDA.

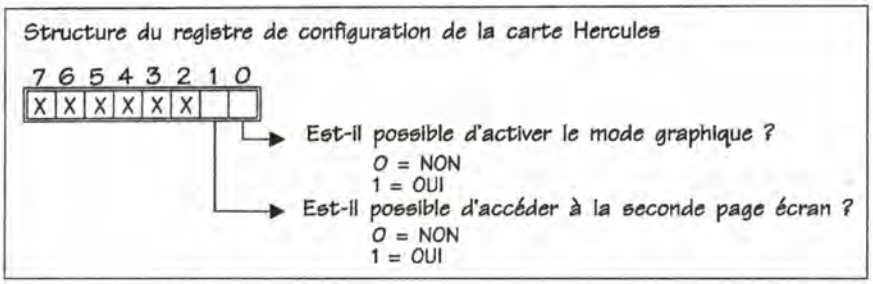
Dans le secteur graphique, cette absence se fait nettement sentir car il n'existe aucune fonction BIOS permettant d'utiliser la carte en mode graphique. En outre, le BIOS n'est pas en mesure de définir ou réclamer des points graphiques dans ce mode. Mais les routines en Assembleur présentées au cours de cette section aident parfaitement à réaliser ces deux tâches.

La RAM vidéo de la carte Hercules

Pour pouvoir afficher également les dessins point par point, qui requièrent beaucoup de mémoire, la carte graphique Hercules dispose de 64 Ko de RAM, répartis en deux pages écran. Chacune des deux pages écran dispose de 32 Ko pour recevoir une page graphique entière ou bien une page de texte (qui a toutefois uniquement besoin de 4 Ko). La première page écran s'étend de l'adresse B000:0000 à l'adresse B000:7FFF. Elle est immédiatement suivie de la seconde page écran qui occupe les cellules de mémoire B000:8000 à B000:FFFF. Cette zone étant normalement utilisée par les cartes couleurs (CGA, EGA et VGA), il existe une méthode permettant de masquer cette zone au moyen d'un commutateur DIP. Ainsi, seule la première page écran reste disponible.

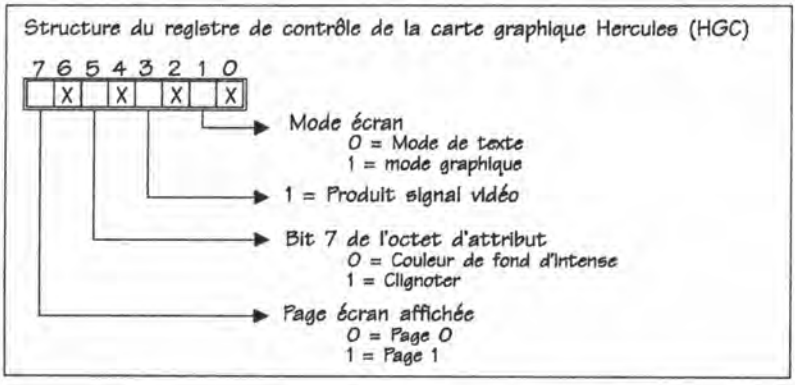
Par rapport à une carte monochrome ordinaire, un registre supplémentaire, le registre de configuration, a été rajouté, qui peut être appelé à travers le port 3BFh. Il est possible d'y écrire mais non de le lire. Seuls les bits 0 et 1 de ce registre sont significatifs. Le premier indique si l'activation du mode graphique est autorisée (1) ou interdite (0) et le second si la seconde page écran peut être autorisée (1) ou non.

Pour éviter tout conflit avec d'autres cartes vidéo (surtout avec la carte couleur), les deux bits sont fixés sur 0 lors du lancement du système, de sorte qu'a priori le graphisme ne peut être utilisé, ni la seconde page d'écran. Pour accéder à ces possibilités de la carte Hercules, un programme devra donc, dans un premier temps, configurer la carte à l'aide du registre de configuration. Le registre de contrôle existait déjà sur la carte monochrome mais son utilisation est quelque peu modifiée.



Les registres de la carte Hercules

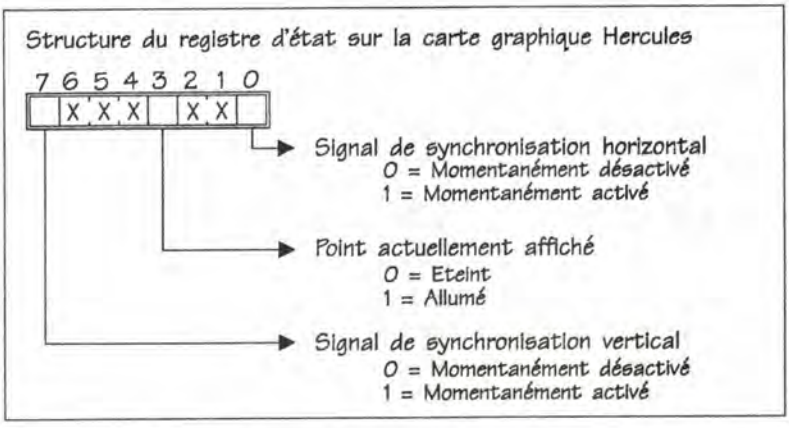
Du point de vue de la structure des divers registres et leur mode d'adressage, la carte Hercules ressemble considérablement à la carte MDA. Grâce à la possibilité d'effectuer un affichage graphique, il ne reste plus que quelques bits inutilisés par la carte IBM, qui offrent un intérêt quelconque dans les registres d'état et de contrôle.



Contrairement à la carte IBM, le bit 0 n'est pas utilisé ici et il n'est donc pas nécessaire de le fixer sur 1 lors du lancement initial du système. Le bit 1 sélectionne l'affichage de texte ou bien l'affichage graphique. Comme nous le verrons par la suite, il ne suffit toutefois pas de modifier cette valeur pour passer du mode de texte au mode graphique ou inversement. Il faut en effet également redéfinir les registres internes du 6845. La construction de l'image doit être suspendue pendant cette opération pour éviter que le 6845 ne produise des signaux "désordonnés" au cours de sa re-programmation.

Par rapport à la carte IBM, le bit 7 est venu s'ajouter aux bits significatifs. Il spécifie en effet laquelle des deux pages écran est affichée sur l'écran. S'il vaut 0, ce sera la première page, s'il vaut 1 ce sera la seconde page. Quelle que soit la page écran actuellement affichée, vous pouvez toutefois, à tout moment, écrire ou lire des valeurs dans n'importe laquelle des deux pages écran. Notez bien qu'il est seulement possible d'écrire dans ce registre, mais non de le lire. (Toute tentative de lecture fournira systématiquement la valeur non significative FFh). Il ne serait donc pas possible de désactiver l'écran sans

modifier le mode ou la page écran activée, en lisant simplement le contenu du registre d'état puis en annulant le bit 3. Si vous voulez annuler le bit 3, vous êtes donc contraint de fixer à nouveau le mode d'affichage et la page écran, sans pouvoir savoir si les valeurs ainsi fixées ne risquent pas de modifier l'état antérieur.



Par rapport à la carte IBM, seule la signification du bit 7 a été modifiée. Il est fixé sur 0 chaque fois que le 6845 envoie un signal de synchronisation verticale à l'écran, pour lancer une nouvelle construction de l'écran.

Le contrôleur CRT de la carte Hercules

Comme sur la carte MDA, le registre d'adresse du CRTC figure sur la carte Hercules à l'adresse de port 3B4h et le registre de données à l'adresse de port 3B5h. Contrairement à la MDA, toutefois, le contenu du registre AX peut ici être chargé dans les registres du CRTC à l'aide d'une instruction OUT 16 bits. La sortie se fait alors à travers le registre d'adresse, le registre AL du processeur devant contenir le numéro du registre du CRTC à adresser et le registre AH le nouveau contenu de ce registre.

La table suivante indique quelles valeurs doivent être chargées dans les différents registres sous les deux modes disponibles.

Contenu des registres CRTC en mode de texte 80x25 et en mode graphique de la carte graphique Hercules

Registre	Signification	Texte	Graphique
00h	Total de caractères horizontalement	97	53
01h	Caractères affichés horizontalement	80	45
02h	Signal de synchronisation horizontal après ... caractères	82	46
03h	Durée du signal horizontal de synchronisation en caractères	15	7

Contenu des registres CRTC en mode de texte 80x25 et en mode graphique de la carte graphique Hercules

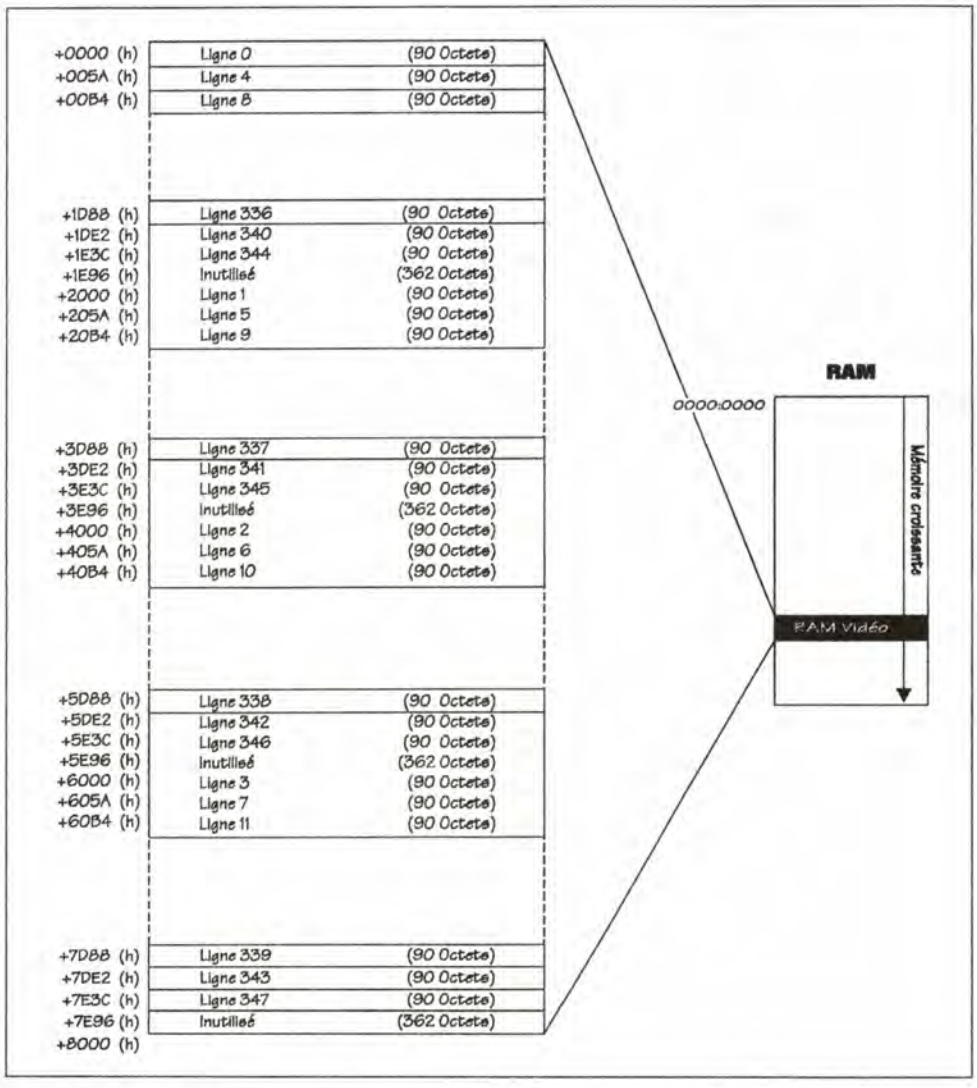
Registre	Signification	Texte	Graphique
04h	Total de caractères verticalement	25	91
05h	Nombre ajusté de caractères verticalement	6	2
06h	Caractères affichés verticalement	25	87
07h	Signal de synchronisation vertical après ... caractères	25	87
08h	Mode d'entrelacement	2	2
09h	Nombre de lignes de grille (de balayage) par ligne de l'écran	13	3
0Ah	Ligne de départ du curseur clignot.de l'écran	11	0
0Bh	Ligne de fin du curseur clignotant de l'écran	12	0
0Ch	Adresse de départ de la page écran affichée (octet de poids fort)	0	0
0Dh	Adresse de départ de la page écran affichée (octet de poids faible)	0	0
0Eh	Adresse de caractère du curseur clignotant de l'écran (octet de poids fort)	0	0
0Fh	Adresse de caractère du curseur clignotant de l'écran (octet de poids faible)	0	0
10h	Position du crayon optique (octet de poids fort)	?	?
11h	Position du crayon optique (octet de poids faible)	?	?

Définition et programmation du mode graphique

Lors du lancement du système, le BIOS charge automatiquement les valeurs pour le mode de texte 80x25 caractères dans ces registres, alors qu'il n'est pas possible de fixer le mode graphique à l'aide du BIOS. Contrairement aux autres cartes vidéo dont les modes sont tous soutenus par le BIOS, il est ici indispensable de prendre en main soi-même la programmation des registres CRTC.

Comme nous l'avons déjà indiqué, la carte Hercules affiche en mode graphique 348 lignes de 720 points. Chaque point de l'écran correspond à un bit de la RAM vidéo. Si le bit correspondant vaut 1, le point sera visible sur l'écran, sinon il restera sombre.

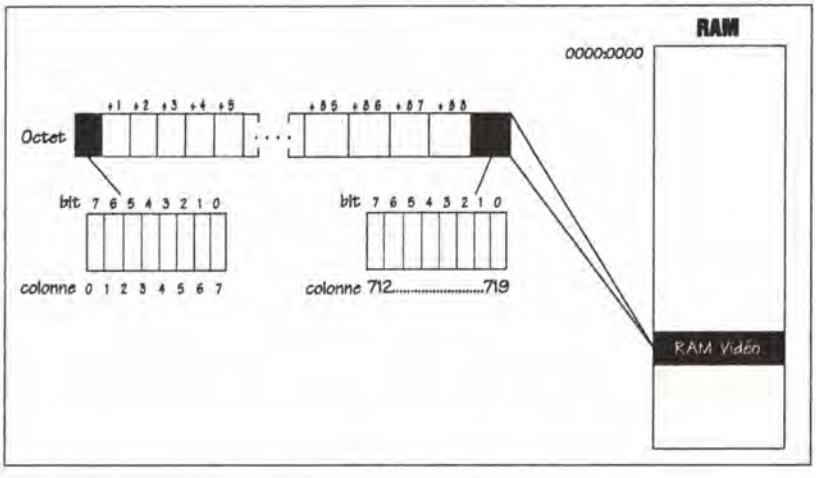
La figure suivante vous permet de mesurer la relative complexité de la structure de la RAM vidéo en mode graphique :



Rapport entre la RAM vidéo et l'image apparaissant sur l'écran (ex : 1ère page écran)

Il semblerait logique que les modèles de bits des différentes lignes soient placés à la suite dans la RAM vidéo. Il n'en est pourtant rien. La RAM vidéo de 32 Ko est en effet divisée en 4 zones de 8 Ko. La première zone de mémoire comporte les modèles des bits pour toutes les lignes dont les numéros sont des multiples entiers de 4 (0, 4, 8, 12 etc...). Les trois zones de mémoire restantes reçoivent respectivement les trois lignes placées entre celles de cette première zone. Ce seront donc les lignes 1, 5, 9, 13, etc... pour la seconde zone de la mémoire, les lignes 2, 6, 10, 14, etc... pour la troisième et enfin les lignes 3, 7, 11, 15, etc... pour la dernière zone. Lorsque le 6845 génère une image, il va donc chercher dans le premier bloc de données les informations pour la ligne zéro,

dans le second bloc celles pour la ligne 1, etc... Une fois qu'il a obtenu les informations du quatrième bloc de données, pour la ligne numéro 3, il revient au bloc de données pour lire les informations de la cinquième ligne, la ligne numéro 4. A l'intérieur des différents blocs de mémoire, chaque ligne occupe 90 octets. Chaque point exige en effet un bit et 720 points exigent donc 720 bits ou 90 octets. Les 90 octets du premier bloc de mémoire fournissent donc le modèle de bits de la ligne zéro de l'écran, les 90 octets suivants le modèle de bits de la ligne quatre, etc... L'octet numéro zéro d'un groupe de 90 octets correspond aux 8 premières colonnes (de points écran) d'une ligne de l'écran, l'octet numéro un (c'est-à-dire le second octet) définissant les colonnes 8 à 15, etc... A l'intérieur de chaque octet, le bit 7 correspond au point écran le plus à gauche et le bit 0 au point écran le plus à droite, comme en notation binaire usuelle.



Rapport entre les 90 octets d'une ligne et l'image apparaissant sur l'écran

Si nous numérotons les points écran d'une ligne de 0 à 719 et ceux d'une ligne de 0 à 347, nous aboutissons à la formule suivante pour calculer l'adresse de l'octet contenant les informations d'un point de coordonnées (X,Y), par rapport au début de la page écran :

```
Adresse = 2000h * (Y mod 4) + 90 * int(Y/4) + int(X/8)
```

Pour calculer le numéro du bit à l'intérieur de cet octet qui correspond au point voulu, nous pouvons utiliser la formule suivante :

```
Numéro du bit = 7 - (X mod 8)
```

Programme d'exemple

Voici à nouveau, pour terminer, un programme dont les routines vous permettront d'exploiter pleinement toutes les possibilités de la carte graphique Hercules. Les différentes routines qui composent ce programme se distinguent de celles proposées pour la carte monochrome par le fait qu'elles permettent d'accéder aux deux pages écran et qu'elles soutiennent le mode graphique.

Listing : VHERC.ASM

```
;*****************************************************************;
;*                          V H E R C                            *;
;*---------------------------------------------------------------*;
;*   Fonction       : fournit quelques fonctions de base pour    *;
;*                    l'accès à la HERCULES GRAPHICS CARD        *;
;*---------------------------------------------------------------*;
;*   Infos          : Toutes les fonctions divisent l'écran en   *;
;*                    colonnes 0 à 79 et lignes 0 à 24           *;
;*                    en mode de texte et en colonnes 0 à 719 et *;
;*                    lignes 0 à 347 en mode graphique           *;
;*---------------------------------------------------------------*;
;*   Auteur         : MICHAEL TISCHER                            *;
;*   développé le   : 11/08/1987                                 *;
;*   Dernière modif.: 1/05/1989                                  *;
;*---------------------------------------------------------------*;
;*   Assemblage     : MASM VHERC;                                *;
;*                    LINK VHERC;                                *;
;*---------------------------------------------------------------*;
;*   Appel          : VHERC                                      *;
;*****************************************************************;

;-- Constantes ---------------------------------------------------

REG_CONTR   = 0388h             ;Adr. de port du registre de contrôle
ADRESS_6845 = 0384h                     ;Registre d'adresse du 6845
DATA_6845   = 0385h                     ;Registre de données du 6845
REG_CONFIG  = 038Fh                      ;Registre de configuration
VIO_SEG     = 0B000h            ;Adresse de segment de la RAM vidéo
CUR_START   = 10            ;N° registre CRTC: ligne de départ du curseur
CUR_END     = 11              ;N° registre CRTC: ligne de fin du curseur
CURPOS_HI   = 14              ;N° registre CRTC: octet fort pos. curseur
CURPOS_LO   = 15             ;N° registre CRTC: octet faible pos. curseur

DELAY       = 20000          ;Compteur de boucle pour boucle d'attente

;-- Macros -------------------------------------------------------

setmode   macro modus                        ;Fixer registre de contrôle

      mov  dx,REG_CONTR          ;Adresse du registre de commande écran
      mov  al,modus                 ;Nouveau mode dans le registre AL
      out  dx,al                 ;Envoyer mode au registre de commande

      endm

setvc     macro                          ;Ecrit une valeur dans l'un des
                                                      ;registres CRTC
                                        ;Entrée : AL = numéro de registre
                                  ;         AH = valeur pour le registre

      mov  dx,ADRESS_6845              ;Adresse du registre d'index
      out  dx,ax                 ;Sortir n° registre et nouvelle valeur

      endm

;-- Pile ---------------------------------------------------------

stack     segment para stack             ;Définition du segment de pile

      dw 256 dup (?)                 ;La pile a une taille de 256 mots

stack     ends                                  ;Fin du segment de pile

;-- Données ------------------------------------------------------

data      segment para 'DATA'           ;Définition du segment de données

;-- Les données pour le programme de démonstration ---------------

initm     db 13,10,"VHERC (c) 1987, 1988 by Michael Tischer",13,10,13,10
      db "Ce programme de démonstration ne tourne qu'avec une"
      db " carte graphique",13,10,"HERCULES (HGC). Si votre PC"
      db " possède un autre type de carte écran.",13,10
      db "veuillez entrer maintenant <s> pour arrêter le programme"
      db ".",13,10,"Sinon veuillez actionner n'importe quelle autre"
      db " touche pour lancer le programme...",13,10,"$"

str1  db 1,17,16,2,7,0
str2  db 2,16,17,1,7,0

domes     db "Vous allez assister maintenant à une petite démonstration"
      db " graphique, suivie",13,10,"d'une démo de texte. Ces "
      db "deux démonstrations peuvent être interrompues.",13,10
      db "en actionnant une touche. Veuillez actionner une touche"
      db " pour lancer le programme...",13,10,"$"

;-- Table avec les adresses d'offset des débuts de ligne ---------
lignes dw  7*160, 8*160, 9*160,10*160,11*160,12*160,13*160
       dw 14*160,15*160,16*160,17*160,18*160,19*160,20*160
       dw 21*160,22*160,23*160,24*160

graphit   db 35h, 2Dh, 2Eh, 07h, 5Bh, 02h  ;Valeurs de registres pour le
      db 57h, 57h, 02h, 03h, 00h, 00h                   ;mode graphique

textt db 61h, 50h, 52h, 0Fh, 19h, 06h      ;Valeurs de registres pour le
      db 19h, 19h, 02h, 00h, 08h, 0Ch                   ;mode graphique

data  ends                                  ;Fin du segment de données

;-- Code ---------------------------------------------------------

code  segment para 'CODE'                ;Définition du segment de CODE

      assume cs:code, ds:data, es:data, ss:stack

;-- Il s'agit simplement du programme de démonstration -----------

demo  proc far

      mov  ax,data                        ;Rechercher adr. de segment du
      mov  ds,ax                    ;segment de données et charger dans DS
      mov  es,ax                                    ;également dans ES

            ;-- Sortir message Init et attendre entrée ----------

      mov  ah,9                     ;Numéro de fonction pour Sortir chaîne
      mov  dx,offset initm                        ;Adresse du message
      int  21h                              ;Appeler interruption du DOS

      xor  ah,ah                     ;Numéro de fonction pour Lire touche
      int  16h                      ;Appeler interruption clavier du BIOS
      cmp  al,"s"                                   ;<s> entré ?
      je   fin                      ;OUI --> Terminer programme à nouveau
      cmp  al,"S"                                   ;<S> entré
      jne  startdemo                              ;NON --> Lancer démo

fin:  mov  ax,4C00h                  ;N° fonction pour Terminer programme
      int  21h                            ;Appeler interruption DOS 21h

        ;-- Programme de démonstration de l'appel des fonctions ----

startdemo label near
      mov  ah,9                     ;Numéro de fonction pour Sortir chaîne
      mov  dx,offset domes                        ;Adresse du message
      int  21h                              ;Appeler interruption du DOS
```

```
;       xor  ah,ah                      ;Numéro de fonction pour Lire touche
;       int  16h                        ;Appeler interruption clavier du BIOS
;
;               ;-- Initialiser mode graphique ----------------------------
;
;       mov  al,11b                     ;Graphisme possible et incruster pag 2
;       call config                                         ;Configurer
;       xor  bp,bp                              ;Accéder à page écran 0
;       call graphi                             ;Passer en mode graphique
;       xor  al,al
;       call cgr                                ;Vider page graphique 0
;
;               ;-- Construire dessin sur l'écran -----------------------
;
;       xor  bx,bx                              ;Commencer par coin supérieur
;       xor  dx,dx                                      ;gauche de l'écran
;       mov  ax,347                             ;Points verticalement
;       mov  cx,719                             ;Points horizontalement
;gr1:   push cx                                 ;Ranger points horizontalement
;       mov  cx,ax                              ;Points vertical. dans compteur
;       push ax                                 ;Ranger points vertical. sur pile
;gr2:   call spix                                        ;Fixer point
;       inc  dx                                         ;Incrémenter ligne
;       loop gr2                                        ;Tracer ligne
;       pop  ax                                 ;Retirer points vertical. de la pile
;       sub  ax,3                               ;Ligne suivante 3 points de moins
;       pop  cx                                 ;Retirer points horizontal. de la pile
;       push cx                                 ;Ranger points horizontalement
;       push ax                                 ;Ranger points vertical. sur pile
;gr3:   call spix                                        ;Fixer point
;       inc  bx                                         ;Incrémenter colonne
;       loop gr3                                        ;Tracer ligne
;       pop  ax                                 ;Retirer points vertical. de la pile
;       pop  cx                                 ;Retirer points horizontal. de la pile
;       sub  cx,6                               ;Ligne suivante 6 points de moins
;       push cx                                 ;Ranger points horizontalement
;       mov  cx,ax                              ;Points vertical. dans compteur
;       push ax                                 ;Ranger points vertical. sur pile
;gr4:   call spix                                        ;Fixer point
;       dec  dx                                         ;Décrémenter ligne
;       loop gr4                                        ;Tracer ligne
;       pop  ax                                 ;Retirer points vertical. de la pile
;       sub  ax,3                               ;Ligne suivante 3 points de moins
;       pop  cx                                 ;Retirer points horizontal. de la pile
;       push cx                                 ;Ranger points horizontalement
;       push ax                                 ;Ranger points vertical. sur pile
;gr5:   call spix                                        ;Fixer point
;       dec  bx                                         ;Incrémenter colonne
;       loop gr5                                        ;Tracer ligne
;       pop  ax                                 ;Retirer points vertical. de la pile
;       pop  cx                                 ;Retirer points horizontal. de la pile
;       sub  cx,6                               ;Ligne suivante 6 points de moins
;       cmp  ax,5                               ;Ligne vertic. supérieure à 5
;       ja   gr1                                        ;OUI --> continuer
;
;       xor  ah,ah                      ;N° fonction pour Attendre touche
;       int  16h                        ;Appeler interruption clavier du BIOS
;
;               ;-- Initialiser mode de texte ----------------------------
;
;       call text                                       ;Activer mode texte
;       mov  cx,0d0dh                           ;Activer curseur plein
;       call cdef
;       call cls                                        ;Vider l'écran
;
;               ;-- Sortir chaînes dans la page écran 0 ------------------
;
;       xor  bx,bx                      ;Commencer dans coin sup. gauche écran
;       call calo                               ;Convertir en adresse d'offset
;       mov  si,offset str1                     ;Adresse d'offset de chaîne 1
;       mov  cx,16*25                           ;Chaîne longue de 5 caractères
;demo1: call print                                       ;Sortir chaîne
;       loop demo1
;
;               ;-- Sortir chaînes dans la page écran 1 ------------------
;
;       inc  bp                                         ;Traiter page écran 1
;       xor  bx,bx                      ;Commencer dans coin sup. gauche écran
;       call calo                               ;Convertir en adresse d'offset
;       mov  si,offset str2                     ;Adresse d'offset de chaîne 1
;       mov  cx,16*25                           ;Chaîne longue de 5 caractères
;demo2: call print                                       ;Sortir chaîne
;       loop demo2
;
;demo3:    setmode 10001000b                     ;Afficher page de texte 1
;
;               ;-- Petite pause ----------------------------------------
;
;       mov  cx,DELAY                           ;Charger compteur de boucle
;pause: loop pause                                       ;Compter jusqu'à 65536
;
;       setmode 00001000b                       ;Afficher page de texte 0
;
;               ;-- Petite pause ----------------------------------------
;
;       mov  cx,DELAY                           ;Charger compteur de boucle
;pause1: loop pause1                                     ;Compter jusqu'à 65536
;
;       mov  ah,1                               ;N° fonction pour Tester touche
;       int  16h                        ;Appeler interruption clavier du BIOS
;       je   demo3                              ;Pas de touche --> alors continuer
;
;                       ;-- Préparer fin du programme ---------------------
;
;       xor  ah,ah                      ;Numéro de fonction pour Lire touche
;       int  16h                        ;Appeler interruption clavier du BIOS
;
;       mov  bp,0                                       ;Page écran 1
;       call cls                                        ;Vider l'écran
;       mov  cx,000Ch                           ;Rétablir curseur normal
;       call cdef
;       call cls                                        ;Vider l'écran
;       jmp  fin                                ;Sauter à la fin du programme
;
;demo    endp
;
;;== Viennent maintenant les fonctions proprement dites ==========
;
;;-- CONFIG: configure la carte HERCULES -------------------------
;;-- Entrée  : AL : Bit 0 = 0 : Seul affichage texte possible
;;--                       1 : Affichage graphique possible également
;;--              Bit 1 = 0 : RAM pour page écran 2 déconnectée
;;--                       1 : RAM pour page écran 2 connectée
;;-- Sortie  : Aucune
;;-- Registres: AX et DX sont modifiés
;
;config  proc near
;
;       mov  dx,REG_CONFIG              ;Adresse du registre de configuration
;       out  dx,al                                      ;Fixer nouvelle valeur
;       ret                                     ;Retour au programme d'appel
;
;config  endp
;
;;-- TEXT: Activer l'affichage de texte ---------------------------
;;-- Entrée  : Aucune
;;-- Sortie  : Aucune
;;-- Registres: AX et DX sont modifiés
;
;text    proc near
;
;       mov  si,offset textt            ;Adresse offset de table de registres
;       mov  bl,00100000b               ;Afficher page 0, mode texte, clignote
;       jmp  short vcprog                       ;Reprogrammer contrôleur vidéo
;
;text    endp
;
;;-- GRAPHSME: Activer l'affichage graphique ----------------------
;;-- Entrée  : Aucune
;;-- Sortie  : Aucune
;;-- Registres: AX et DX sont modifiés
;
;graphi  proc near
;
;       mov  si,offset graphit          ;Adresse offset de table de registres
;       mov  bl,00000010b                       ;Afficher page 0, mode graphique
;
;graphi  endp
;
;;-- VCPROG: programme le contrôleur vidéo ------------------------
;;-- Entrée  : SI = Adresse d'une table de registres
;;--           BL = Valeur pour registre de commande de l'écran
;;-- Sortie  : Aucune
;;-- Registres: AX, SI, BH, DX et FLAGS sont modifiés
;
;vcprog  proc near
;
;       setmode bl                              ;Bit 3 = 0 : Ecran désactivé
;
;       mov  cx,12                                      ;12 registres sont fixés
;       xor  bh,bh                              ;Commencer par le registre 0
;vcp1:  lodsb                                   ;Lire valeur de registre dans table
;       mov  ah,al                                      ;Valeur de registre dans AH
;       mov  al,bh                                      ;Numéro de registre dans AL
;       setvc                                   ;Transmettre valeur au contrôleur
;       inc  bh                                         ;Appeler registre suivant
;       loop vcp1                                       ;Fixer autres registres
;
;       or   bl,8                                       ;Bit 3 = 1 : Ecran activé
;       setmode bl                                      ;Fixer nouveau mode
;       ret                                     ;Retour au programme d'appel
;
;vcprog  endp
;
;;-- CDEF: Fixe les lignes de départ et de fin du curseur --------
;;-- Entrée  : CL = ligne de départ
```

```
;--          CH = ligne de fin
;-- Sortie   : Aucune
;-- Registres: AX et DX sont modifiés

codef     proc near

      mov  al,CUR_START                ;Registre : ligne de départ
      mov  ah,cl                         ;Ligne de départ dans AH
      setvc                          ;Communiquer au contrôleur vidéo
      mov  al,CUR_END                    ;Registre : ligne de fin
      mov  ah,ch                         ;Ligne de fin dans AH
      setvc                          ;Communiquer au contrôleur vidéo
      ret

codef     endp

;-- SETBLINK: fixe le curseur clignotant de l'écran ----------
;-- Entrée   : DI = Adresse d'offset du curseur
;-- Sortie   : Aucune
;-- Registres: BX, AX et DX sont modifiés

setclign  proc near

      mov  bx,dI                       ;Transférer offset dans BX
      mov  al,CURPOS_HI          ;Reg. : octet fort d'offset du curseur
      mov  ah,bh                         ;Octet fort de l'offset
      setvc                          ;Communiquer au contrôleur vidéo
      mov  al,CURPOS_LO          ;Reg. : octet faible offset du curseur
      mov  ah,bl                         ;Octet faible de l'offset
      setvc                              ;Transmettre au CRTC
      ret

setclign  endp

;-- GETVC: lit un octet dans l'un des registres du contrôleur vidéo ----
;-- Entrée   : AL = Numéro du registre
;-- Sortie   : AL = Contenu du registre
;-- Registres: DX et AL sont modifiés

getvc     proc near

      mov  dx,ADRESS_6845              ;Adresse du registre d'index
      out  dx,al                        ;Envoyer numéro du registre
      jmp  $+2                        ;Petite pause d'entrée/sortie
      inc  dx                          ;Adresse du registre d'index
      in   al,dx                        ;Transférer contenu dans AL
      ret                               ;Retour au programme d'appel

getvc     endp

;-- SCROLLUP: fait défiler une fenêtre de N lignes vers le haut -----
;-- Entrée   : BL = ligne haut à gauche
;--            BH = colonne haut à gauche
;--            DL = ligne bas à droite
;--            DH = colonne bas à droite
;--            CL = nombre de lignes de défilement
;--          : BP = Numéro de la page écran (0 ou 1)
;-- Sortie   : Aucune
;-- Registres: seuls les FLAGS sont modifiés
;-- Infos    : les lignes de l'écran libérées sont vidées

scrollup  proc near

      cld                            ;Augmenter comptage pour instr. chaîne

      push ax                          ;Sauver tous les registres modifiés
      push bx                                    ;sur la pile
      push dI                                ;Dans ce cas, l'ordre doit
      push sI                                    ;être respecté !

      push bx                          ;Ces trois registres sont retirés de
      push cx                                ;la pile avant même la fin de
      push dx                                            ;la routine
      sub  dl,bl                             ;Calculer nombre de lignes
      inc  dl
      sub  dl,cl                       ;Retrancher nombre de lignes de défil
      sub  dh,bh                             ;Calculer nombre colonnes
      inc  dh
      call calo                          ;Convertir haut à gauche en offset
      mov  sI,dI                               ;Ranger adresse dans SI
      add  bl,cl                       ;Première ligne dans fenêtre décalée
      call calo                        ;Convertir première ligne en offset
      xchg sI,dI                                 ;Echanger SI et DI
      push ds                                ;Sauver registre de segment
      push es                                            ;sur la pile
      mov  ax,VIO_SEG                  ;Adresse de segment de la RAM vidéo
      mov  ds,ax                                         ;dans DS
      mov  es,ax                                     ;et amener ES
supl: mov  ax,dI                                 ;Ranger DI dans AX
      mov  bx,sI                                 ;Ranger SI dans BX
      mov  cl,dh                       ;Nombre de colonnes dans le compteur
      rep movsw                                      ;Décaler une ligne
      mov  dI,ax                                      ;Retirer DI de AX
      mov  sI,bx                                      ;Retirer SI de BX
      add  dI,160                    ;Fixer chaque fois sur ligne suivante
      add  sI,160
      dec  dl                                ;Toutes les lignes traitées ?
      jne  supl                            ;NON --> décaler encore une ligne
      pop  es                              ;Retirer à nouveau reg. de segment
      pop  ds                                                    ;de la pile
      pop  dx                                        ;Retirer coin inf. droit
      pop  cx                                        ;Retirer nombre de lignes
      pop  bx                                        ;Retirer coin sup. gauche
      mov  bl,dl                                     ;Ligne inférieure dans BL
      sub  bl,cl                                   ;Retrancher nombre de lignes
      inc  bl
      mov  ah,07h                                    ;Couleur : noir sur blanc
      call clear                                       ;Vider lignes libérées

      pop  sI                                        ;CX et DX ont déjà été
      pop  dI                                                    ;ramenés
      pop  bx
      pop  ax

      ret                                          ;Retour au programme d'appel

scrollup  endp

;-- SCROLLDN: fait défiler une fenêtre de N lignes vers le bas ------
;-- Entrée   : BL = ligne haut à gauche
;--            BH = colonne haut à gauche
;--            DL = ligne bas à droite
;--            DH = colonne bas à droite
;--            CL = nombre de lignes de défilement
;--          : BP = Numéro de la page écran (0 ou 1)
;-- Sortie   : Aucune
;-- Registres: seuls les FLAGS sont modifiés
;-- Infos    : les lignes de l'écran libérées sont vidées

scrolldn  proc near

      cld                            ;Augmenter comptage pour instr. chaîne

      push ax                          ;Sauver tous les registres modifiés
      push bx                                    ;sur la pile
      push dI                                ;Dans ce cas, l'ordre doit
      push sI                                    ;être respecté !

      push bx                          ;Ces trois registres sont retirés de
      push cx                                ;la pile avant même la fin de
      push dx                                            ;la routine

      sub  dh,bh                             ;Calculer nombre colonnes
      inc  dh
      mov  al,bl                       ;Ranger ligne haut à gauche dans AL
      mov  bl,dl                               ;Ligne i.d. dans ligne i.g.
      call calo                          ;Convertir haut à gauche en offset
      mov  sI,dI                               ;Ranger adresse dans SI
      sub  bl,cl                       ;Retrancher nombre de l. de défilement
      call calo                          ;Convertir haut à gauche en offset
      xchg sI,dI                                 ;Echanger SI et DI
      sub  dl,al                             ;Calculer nombre de lignes
      inc  dl
      sub  dl,cl                       ;Retrancher nombre de lignes de défil.
      push ds                                ;Sauver registre de segment
      push es                                            ;sur la pile
      mov  ax,VIO_SEG                  ;Adresse de segment de la RAM vidéo
      mov  ds,ax                                         ;dans DS
      mov  es,ax                                     ;et amener ES
sdn1: mov  ax,dI                                 ;Ranger DI dans AX
      mov  bx,sI                                 ;Ranger SI dans BX
      mov  cl,dh                       ;Nombre de colonnes dans le compteur
      rep movsw                                      ;Décaler une ligne
      mov  dI,ax                                      ;Retirer DI de AX
      mov  sI,bx                                      ;Retirer SI de BX
      sub  dI,160                    ;Fixer chaque fois sur ligne suivante
      sub  sI,160
      dec  dl                                ;Toutes les lignes traitées ?
      jne  sdn1                            ;NON --> décaler encore une ligne
      pop  es                              ;Retirer à nouveau registre de segment
      pop  ds                                                    ;de la pile
      pop  dx                                        ;Retirer coin inf. droit
      pop  cx                                        ;Retirer nombre de lignes
      pop  bx                                        ;Retirer coin sup. gauche
      mov  dl,bl                                     ;Ligne supérieure dans DL
      add  dl,cl                                   ;Additionner nombre de lignes
      dec  dl
      mov  ah,07h                                    ;Couleur : noir sur blanc
      call clear                                       ;Vider lignes libérées

      pop  sI                                        ;CX et DX ont déjà été
      pop  dI                                                    ;ramenés
      pop  bx
      pop  ax

      ret                                          ;Retour au programme d'appel
```

```
;
;scrolldn  endp
;
;;-- CLS: Vider l'écran tout entier -------------------------
;;-- Entrée   : BP = Numéro de la page écran (0 ou 1)
;;-- Sortie   : Aucune
;;-- Registres: seuls les FLAGS sont modifiés
;
;cls       proc near
;
;       mov  ah,07h                          ;Couleur est blanc sur noir
;       xor  bx,bx                             ;haut à gauche est (0/0)
;       mov  dx,4F18h                          ;bas à droite est (79/24)
;
;            ;-- Le passage à Clear se fait automatiquement ------------
;
;cls       endp
;
;;-- CLEAR: remplit d'espaces une zone d'écran déterminée ------------
;;-- Entrée   : AH = attribut/couleur
;;--            BL = ligne haut à gauche
;;--            BH = colonne haut à gauche
;;--            DL = ligne bas à droite
;;--            DH = colonne bas à droite
;;--            BP = Numéro de la page écran (0 ou 1)
;;-- Sortie   : Aucune
;;-- Registres: seuls les FLAGS sont modifiés
;
;clear     proc near
;
;       cld                            ;Augmenter comptage pour instr. chaine
;       push cx                          ;Tous les registres modifiés par la
;       push dx                        ;suite doivent être sauvés sur la pile
;       push si
;       push di
;       push es
;       sub  dl,bl                                ;Calculer nombre de lignes
;       inc  dl
;       sub  dh,bh                                ;Calculer nombre colonnes
;       inc  dh
;       call calo                    ;Adresse d'offset du coin sup. gauche
;       mov  cx,VIO_SEG                ;Adresse de segment de la RAM vidéo
;       mov  es,cx                                          ;dans ES
;       xor  ch,ch                           ;Octet fort du compteur sur 0
;       mov  al," "                                        ;Espace
;clear1: mov  si,di                                  ;Ranger DI dans SI
;       mov  cl,dh                     ;Nombre de colonnes dans le compteur
;       rep stosw                                      ;Sauver espace
;       mov  di,si                                  ;Retirer DI de SI
;       add  di,160                              ;Fixer dans ligne suivante
;       dec  dl                                 ;Toutes les lignes traitées ?
;       jne  clear1                            ;NON --> vider encore une ligne
;
;       pop  es                              ;Retirer de la pile les registres
;       pop  di                                           ;sauvegardés
;       pop  si
;       pop  dx
;       pop  cx
;       ret                                        ;Retour au programme d'appel
;
;clear     endp
;
;;-- PRINT: sort une chaîne sur l'écran ----------------------
;;-- Entrée   : AH = Attribut/couleur
;;--            DI = Adresse d'offset du premier caractère
;;--            SI = Adresse d'offset de la chaîne par rapport à DS
;;--            BP = Numéro de la page écran (0 ou 1)
;;-- Sortie   : DI désigne position après dernier caractère sorti
;;-- Registres: AL, DI et FLAGS sont modifiés
;;-- Infos    : La chaîne doit être terminée par le caractère NUL.
;;--            Les autres caractères de commande ne sont pas identifiés.
;
;print     proc near
;
;       cld                            ;Augmenter comptage pour instr. chaîne
;       push si                              ;Sauver SI, DX et ES sur la pile
;       push es
;       push dx
;       mov  dx,VIO_SEG                ;Adresse de segment RAM vidéo d'abord
;       mov  es,dx                                      ;dans DX puis dans ES
;       jmp  print1                    ;Lire premier caractère de la chaîne
;
;print0: stosw                      ;Sauver attribut et couleur dans RAM vidéo
;print1: lodsb                          ;Lire caractère suivant de la chaîne
;       or   al,al                                          ;Est-ce NUL
;       jne  print0                                       ;NON --> Sortir
;
;printe:  pop  dx                       ;Retirer SI, DX et ES de la pile
;       pop  es
;       pop  si
;       ret                                        ;Retour au programme d'appel
;
;print     endp
;
;;-- CALO: convertit ligne et colonne en adresse d'offset ---------
;;-- Entrée   : BL = ligne
;;--            BH = colonne
;;--            BP = Numéro de la page écran (0 ou 1)
;;-- Sortie   : DI = l'adresse d'offset
;;-- Registres: DI et FLAGS sont modifiés
;
;calo      proc near
;
;       push ax                                    ;Sauver AX sur la pile
;       push bx                                    ;Sauver BX sur la pile
;
;       shl  bx,1                                ;Colonne et ligne fois 2
;       mov  al,bh                                       ;Colonne dans AL
;       xor  bh,bh                                         ;Octet fort
;       mov  di,[lignes+bx]                ;Lire adresse d'offset de ligne
;       xor  ah,ah                           ;Octet fort pour offset colonne
;       add  di,ax                        ;Additionner offset ligne et colonne
;       or   bp,bp                                       ;Page écran 0 ?
;       je   caloe                                    ;OUI --> adresse correcte
;
;       add  di,8000h                     ;Additionner 32 Ko pour page écran 1
;
;caloe: pop  bx                                     ;Retirer BX de la pile
;       pop  ax                                     ;Retirer AX de la pile
;       ret                                      ;Retour au programme d'appel
;
;calo      endp
;
;;-- CGR: Vider intégralement écran graphique ------------------
;;-- Entrée   : BP = Numéro de la page écran (0 ou 1)
;;--            AL = 00(h) : Effacer tous les points
;;--                 FF(h) : Fixer tous les points
;;-- Sortie   : Aucune
;;-- Registres: AH, BX, CX, DI et FLAGS sont modifiés
;
;cgr   proc near
;       push es                                    ;Sauver ES sur la pile
;       cbw                                              ;Etendre AL à AH
;       xor  di,di                           ;Adresse d'offset dans RAM vidéo
;       mov  bx,VIO_SEG                      ;Adresse de segment page écran 0
;       or   bp,bp                                       ;Vider page 1 ?
;       je   cgr1                                     ;NON --> vider page 0
;
;       add  bx,0800h                        ;Adresse de segment Page écran 1
;
;cgr1: mov  es,bx                        ;Adr de segment dans reg. de segment
;       mov  cx,4000h                      ;Une page représente 16 Ko de mots
;       rep stosw                                        ;Remplir page
;       pop  es                                    ;Retirer ES de la pile
;       ret                                      ;Retour au programme d'appel
;
;cgr   endp
;
;;-- SPIX: fixe un point sur l'écran graphique ----------------------
;;-- Entrée   : BP = Numéro de la page écran (0 ou 1)
;;--            BX = Colonne (0 à 719)
;;--            DX = Ligne (0 à 347)
;;-- Sortie   : Aucune
;;-- Registres: AX, DI et FLAGS sont modifiés
;
;spix  proc near
;
;       push es                                    ;Sauver ES sur la pile
;       push bx                                    ;Sauver BX sur la pile
;       push cx                                    ;Sauver CX sur la pile
;       push dx                                    ;Sauver DX sur la pile
;
;       xor  di,di                           ;Adresse d'offset dans RAM vidéo
;       mov  cx,VIO_SEG                      ;Adresse de segment page écran 0
;       or   bp,bp                                     ;Accéder à page 1 ?
;       je   spix1                                    ;NON --> accéder à page 0
;
;       add  cx,0800h                        ;Adresse de segment Page écran 1
;
;spix1: mov  es,cx                       ;Adr. de segment dans reg. de segment
;       mov  ax,dx                                  ;Amener ligne dans AX
;       shr  ax,1                            ;Décaler ligne 2 fois sur la droite
;       shr  ax,1                              ;ce qui revient à diviser par 4
;       mov  cl,90                                    ;Le facteur est 90
;       mul  cl                                    ;Multiplier ligne par 90
;       and  dx,11b                  ;Masquer tous les bits sauf bits 0 et 1
;       mov  cl,3                                        ;3 décalages
;       ror  dx,cl                           ;Rotation par la droite (* 2000(h))
;       mov  di,bx                                  ;Colonne dans DI
;       mov  cl,3                                        ;3 décalages
;       shr  di,cl                                     ;Diviser par 8
;       add  di,ax                                  ;+ 90 * int(Ligne/4)
;       add  di,dx                               ;+ 2000(h) * (Ligne mod 4)
;       mov  cl,7                                  ;7 décalages maximum
;       and  bx,7                                    ;Colonne mod 8
;       sub  cl,bl                                 ;7 - Colonne mod 8
```

```
        mov  ah,1                   ;Déterminer valeur bit du point
        shl  ah,cl
        mov  al,es:[di]                     ;Lire 8 points
        or   al,ah                          ;Fixer point
        mov  es:[di],al                     ;Réécrire 8 points

        pop  dx                     ;Retirer DX de la pile
        pop  cx                     ;Retirer CX de la pile
        pop  bx                     ;Retirer BX de la pile
        pop  es                     ;Retirer ES de la pile
        ret                                 ;Retour au programme d'appel

spix    endp

;--- Fin ------------------------------------------------------------

code    ends                                        ;Fin du segment de CODE
        end  demo                   ;Commencer exécution du prog. par DEMO
```

4.7. La carte couleur IBM (CGA)

Tout comme la carte MDA, la carte CGA est apparue avec le PC. Pendant longtemps, elle représentait le standard dans le domaine graphique. Mais l'apparition de la carte CGA a mis fin à son règne. Au départ, les cartes CGA ne furent pas beaucoup utilisées car elles présentaient des caractéristiques peu appréciables. Contrairement aux autres cartes, elles ne permettaient pas d'adresser directement la RAM vidéo. En revanche, elles créaient la célèbre impression de neige lorsqu'on accédait à la RAM sans l'aide du contrôleur CRT.

Ce chapitre montre comment éviter ces phénomènes. Ils sont toutefois inexistants dans les cartes CGA fabriquées ultérieurement et surtout avec l'émulation CGA à travers la carte EGA et VGA. Les thèmes suivants seront en outre traités dans les sections suivantes :

- ✓ Les modes texte de la carte
- ✓ L'octet d'attribut en mode texte
- ✓ Définition et programmation des modes graphiques
- ✓ Sélection de couleurs dans les modes graphiques
- ✓ Les registres de la carte CGA
- ✓ Le contrôleur CRT de la carte CGA

Les modes texte de la carte CGA

La carte CGA reconnaît deux modes texte dans lesquels 40*25 ou 80*25 caractères apparaissent sur l'écran. Compte tenu de la fonction d'initialisation du BIOS vidéo, ces modes portent les codes 1 (40*25) ou 3 (80*25). Il existe deux variantes dans chaque mode. Il s'agit des modes dans lesquels les couleurs de premier plan et de fond des caractères sont sélectionnées dans une palette de 16 couleurs. Dans les modes BIOS 0 (40*25) et 2 (80*25), un signal de couleur n'est pas transmis au moniteur si bien que les codes de couleur sont convertis automatiquement en niveaux de gris.

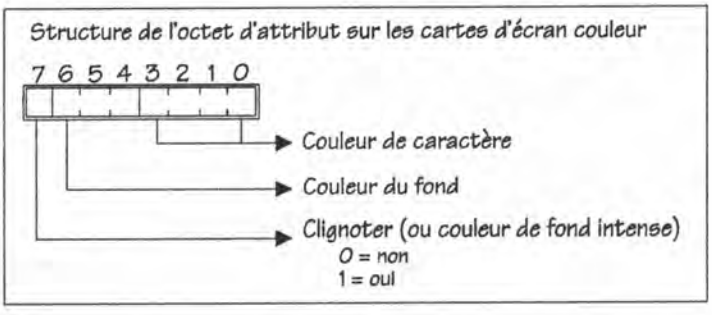
Le nombre de pages écran disponibles dépend de la résolution choisie. Dans les deux modes texte 80*25, on peut mettre en place quatre pages écran réparties dans les 16 Ko de la RAM vidéo. La première page écran commence donc pour le BIOS en B800:0000, la seconde en B800:1000, la troisième en B800:2000 et la dernière enfin en B800:3000. En mode 40*25, comme chaque page écran n'occupe que 2000 octets,

ce sont même 8 pages écran qui peuvent être sauvegardées. Elles commencent respectivement aux adresses B800:0000, B800:0800, B800:1000, B800:1800, B800:2000, etc.

L'affichage des différents caractères, toujours inclus dans une matrice 8*8, s'effectue indépendamment du mode de texte spécifié. La qualité de l'affichage procuré par une carte CGA offre une lisibilité nettement inférieure à celle des cartes vidéo monochromes MDA et HGC qui utilisent une matrice de caractères 9*14.

L'octet d'attribut en mode texte

Dans une carte CGA, l'octet d'attribut permet à un caractère de sélectionner les couleurs de premier plan et de fond parmi les 16 couleurs d'une palette prédéfinie.



Les 4 bits inférieurs de l'attribut spécifient l'une des 16 couleurs possibles pour le premier plan. La signification des 4 bits supérieurs varie suivant que le clignotement est ou non activé. Si c'est le cas, les bits 4 à 6 spécifient une des 8 premières couleurs de la palette de couleurs pour le fond, alors que l'état du bit 7 fixe si le caractère doit clignoter ou non. Si le clignotement est désactivé, les bits 4 à 7 fixent une des 16 couleurs possibles pour le fond.

Palette de couleurs de la carte couleur

Décimal	Hexadécimal	Binaire	Couleur
0	00h	0000b	Noir
1	01h	0001b	Bleu
2	02h	0010b	Vert
3	03h	0011b	Bleu turquoise
4	04h	0100b	Rouge
5	05h	0101b	Violet
6	06h	0110b	Marron

Palette de couleurs de la carte couleur			
Décimal	**Hexadécimal**	**Binaire**	**Couleur**
7	07h	0111b	Gris clair
8	08h	1000b	Gris sombre
9	09h	1001b	Bleu clair
10	0Ah	1010b	Vert clair
11	0Bh	1011b	Cobalt clair
12	0Ch	1100b	Rouge clair
13	0Dh	1101b	Violet clair
14	0Eh	1110b	Jaune
15	0Fh	1111b	Blanc

Les modes graphiques de la carte CGA

Comme nous l'avons indiqué plus haut, la carte couleur IBM soutient trois modes graphiques différents, dont deux seulement sont utilisés dans la pratique. Le mode "oublié" est l'affichage de 160*100 points avec 16 couleurs possibles par point. Si le 6845 est bien, en effet, capable de traiter une résolution de ce type, le reste de l'électronique de la carte ne soutient pas ce mode. On utilise souvent au contraire les deux modes graphiques avec une résolution de 320*200 points et de 640*200 points. Le premier permet un affichage graphique avec 4 couleurs, le second avec 2 couleurs seulement.

Le mode 320*200 points

Pour afficher du graphisme avec une résolution de 320*200 points avec 4 couleurs, la totalité des 16 Ko de mémoire RAM de la carte vidéo est requise. Il est, de ce fait, impossible de traiter en mémoire plus d'une page graphique à la fois. Sur les 4 couleurs possibles, une seule (la couleur du fond) peut toutefois être librement choisie parmi les 16 couleurs existantes. Elle s'applique automatiquement à l'ensemble de la page graphique. Les 3 autres couleurs sont tirées d'une des deux palettes de couleurs que le programme peut sélectionner et donc chacune comporte 3 couleurs.

```
Palette 1 :  Couleur 1 : Turquoise
             Couleur 2 : Violet
             Couleur 3 : Blanc
Palette 2 :  Couleur 1 : Vert
             Couleur 2 : Rouge
             Couleur 3 : Jaune
```

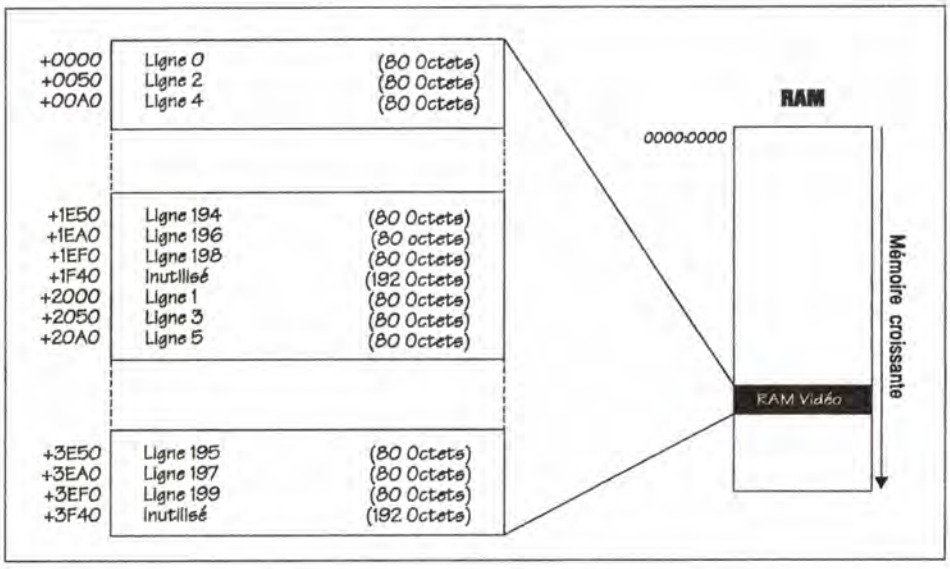
Après sélection d'une des deux palettes, on parvient donc bien à 4 couleurs disponibles et deux bits sont nécessaires pour coder la couleur de chaque point de l'écran. Il faut en effet 2 bits pour représenter quatre nombres, les valeurs 0 à 3. Voici l'équivalence entre les valeurs possibles et les couleurs sélectionnées :

```
0       00b = couleur de fond à choisir librement
1       01b = couleur 1 de la palette sélectionnée
2       10b = couleur 2 de la palette sélectionnée
3       11b = couleur 3 de la palette sélectionnée
```

La palette souhaitée peut être définie de deux manières : par la programmation directe du registre de sélection des couleurs décrit ci-après ou par l'appel d'une fonction BIOS particulière. Il s'agit de la sous-fonction 01h de la fonction 0Bh de l'interruption vidéo du BIOS. Avant l'appel de l'interruption vidéo, chargez le numéro de fonction 0Bh dans le registre AH, le numéro de sous-fonction 01h dans BH et le numéro de la palette souhaitée (0 ou 1) dans le registre BL.

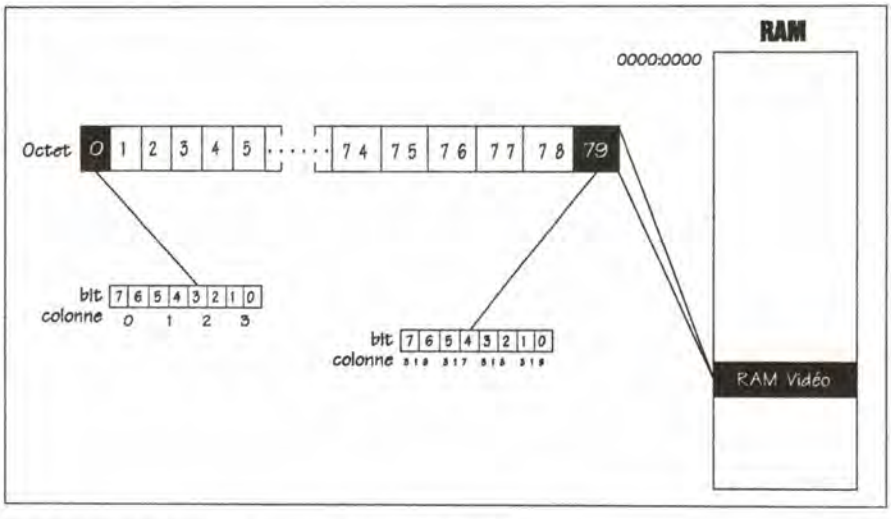
Dans ce mode, la structure de la RAM vidéo fait rappeler quelque peu l'organisation de la RAM vidéo en mode graphique avec la carte Hercules. Dans la carte CGA également, les différentes lignes graphiques ne suivent pas un ordre chronologique dans la mémoire. Les nombres pairs (0, 2, 4...) et les nombres impairs sont au contraire séparés en deux blocs mémoire. Le premier bloc commence à l'adresse B800:0000, le second à B800:2000.

Sous ce mode, la structure de la RAM vidéo n'est pas sans rappeler celle de la carte Hercules pour l'affichage graphique. Ici aussi, les différentes lignes graphiques ne sont pas placées dans leur ordre logique. Les lignes paires (0, 2, 4, ...) et impaires sont en effet écrites dans deux blocs de mémoire différents. Le premier bloc commence à l'adresse B800:0000 et le second en B800:2000.



Structure de la RAM vidéo en mode graphique (Système de blocs)

A l'intérieur des deux blocs, chaque ligne graphique occupe 80 octets puisque 4 points des 320 d'une ligne sont codés dans chaque octet. Le premier octet d'une ligne graphique (d'une ligne de 80 octets) correspond aux 4 premiers points d'une ligne graphique. Les bits 7 et 8 contiennent les informations de couleurs pour le point le plus à gauche et les bits 0 et 1 celles pour le point le plus à droite des quatre points représentés par l'octet.



Codage d'une ligne graphique en mode 320 points sur 200

Comme pour la carte Hercules, une formule permet de calculer l'adresse de l'octet contenant les informations pour un point donné de l'écran, par rapport à l'adresse de départ de la page écran. Si X désigne la colonne de l'écran (0 à 319) et Y la ligne de l'écran (0 à 199) :

```
Adresse = 2000h * (Y mod 2) + 80 * int(Y/2) + int(X/4)
```

Pour calculer la position des deux bits de cet octet représentant le point voulu, nous utiliserons la formule suivante :

```
Numéro de bit = 6 - 2 * (X mod 4)
```

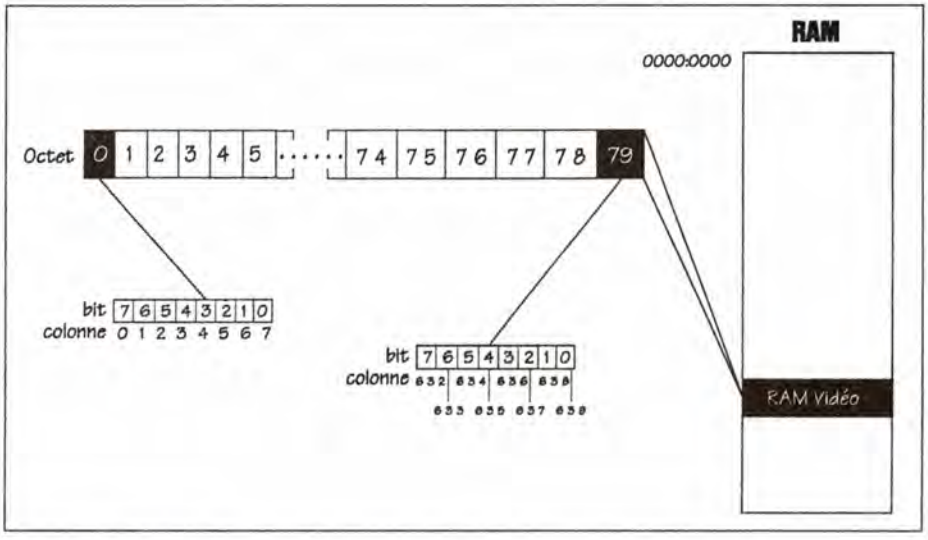
Si cette formule fournit 4 comme résultat, cela signifiera que les informations de couleur pour le point voulu sont codées dans les bits 4 et 5 de l'octet.

Le mode graphique 320 points sur 200 existe aussi dans une version où le signal de couleur est interdit ce qui fait apparaître des niveaux de gris. En raison de son fonctionnement et de la structure de sa RAM vidéo, ce mode ressemble au mode 4 mais porte le code 5 dans la nomenclature BIOS.

Le mode 640*200 points

Le mode haute résolution, avec une résolution de 640 points sur 200, ne permet d'afficher que 2 couleurs. La structure de la RAM vidéo sous ce mode est la même qu'en mode 320*200 points. Un seul bit suffit en effet à coder un point, ce qui compense le fait que chaque ligne comporte le double de points. Ce sont donc toujours 80 octets qui seront utilisés sous ce mode pour représenter une ligne de l'écran. Les formules pour accéder à un point de l'écran ressembleront donc beaucoup aux précédentes :

```
Adresse = 2000h * (Y mod 2) + 80 * int(Y/2) + int(X/8)

Numéro de bit = 7 - (X mod 8)
```



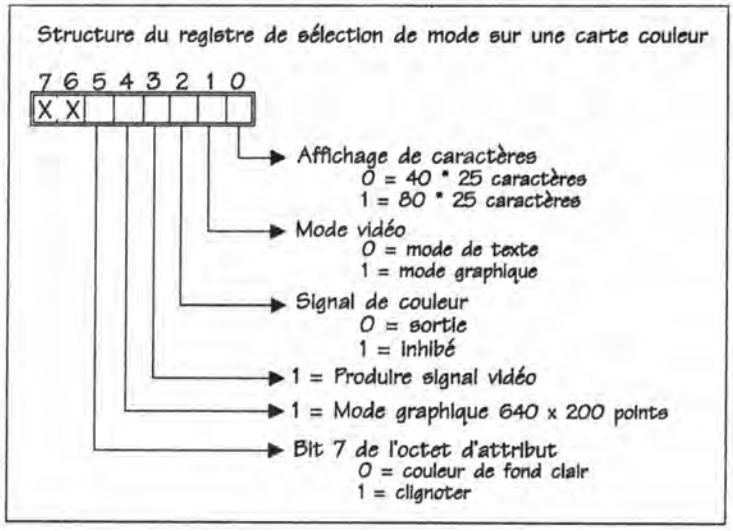
Codage d'une ligne graphique en mode 640 points sur 200

Dans ce mode, les points représentés par un bit de valeur 0 apparaissent en noir sur l'écran. Au contraire, si un bit est réglé, le point correspondant apparaît dans la couleur codée dans les quatre bits inférieurs du registre de sélection de couleur. Ce registre est décrit ci-après.

La couleur souhaitée peut cependant être définie sans programmer directement ce registre. Il suffit d'utiliser à cet effet d'une fonction du BIOS vidéo. Il s'agit en l'occurrence de la sous-fonction 00h de la fonction 0Bh. Pour son appel, elle attend le numéro de fonction 0Bh dans le registre AH, le numéro de sous-fonction 00h en BH et la couleur sélectionnée entre 0 et 15 dans le registre BL.

Les registres de la carte CGA

En dehors des divers registres du contrôleur CRT, la carte CGA dispose de trois registres supplémentaires désignés par registre de sélection de mode, registre de sélection d'état et registre de sélection de couleur. Le registre de sélection de mode est particulièrement intéressant car il détermine le mode dans lequel la carte CGA doit être utilisée. Il se situe à l'adresse 3D8h. On ne peut seulement écrire dans ce registre et non le lire. Généralement, on n'a pas besoin d'accéder directement à ce registre parce que le BIOS se charge de prendre les mesures nécessaires lors de l'initialisation d'un mode vidéo à travers la fonction 00h du BIOS vidéo.



Le bit 0 de ce registre définit, en mode de texte, si 80 caractères (colonnes) ou 40 doivent être affichés par ligne. Un 1 signifie ici 80 colonnes, 0 signifie 40 colonnes.

Le bit 1 permet de passer du mode de texte en mode graphique 320*200 points. Un 1 dans ce registre sélectionne le mode graphique, 0 active à nouveau le mode de texte.

Le bit 2 intéresse tous ceux qui veulent exploiter leur carte couleur avec un moniteur monochrome. Si ce bit contient en effet la valeur 1, la production d'un signal de couleur par le 6845 sera interdite et seules des images en noir et blanc pourront être réalisées.

Le bit 3 contrôle la construction de l'image : s'il contient la valeur 0, la construction de l'image sera interdite et l'écran restera noir. C'est ce qu'il est conseillé de faire chaque fois que l'on change de mode d'affichage, pour éviter que des signaux "anarchiques" ne soient envoyés au moniteur, ce qui risquerait de l'endommager.

Pour activer le mode graphique 640 points sur 200, le bit 4 doit être fixé sur 1.

Le bit 5 joue le même rôle que sur les cartes monochromes. S'il contient un 0, le clignotement est désactivé et le bit 7 de l'octet d'attribut est pris en compte comme bit supplémentaire pour la sélection de la couleur du fond. Dans ce cas, il est donc possible d'employer la totalité des 16 couleurs disponibles comme couleur du fond. Ce bit est cependant normalement fixé sur 1, de sorte que le clignotement des caractères est possible.

Comme vous le voyez, ce registre permet de sélectionner les différents modes de texte ou modes graphiques ainsi que l'affichage couleur ou noir et blanc de ces modes. Ce sont les bits 0, 1, 2 et 4 qui servent à cela. La table suivante montre comment ces bits doivent être fixés pour obtenir un mode déterminé :

Sélection du mode vidéo sur la carte couleur, à travers le registre de sélection de mode

7	6	5	4	3	2	1	0	
?	?	?	0	?	1	0	0	40 x 25 caractères de texte, pas de couleurs
?	?	?	0	?	0	0	0	40 x 25 caractères de texte avec 16 couleurs
?	?	?	0	?	1	0	0	80 x 25 caractères de texte, pas de couleurs
?	?	?	1	?	0	0	1	80 x 25 caractères de texte avec 16 couleurs
?	?	?	0	?	0	0	1	320 x 200 points graphiques, pas de couleurs
?	?	?	0	?	1	1	0	320 x 200 points graphiques avec 4 couleurs
?	?	?	1	?	1	1	0	640 x 200 points graphiques avec 2 couleurs

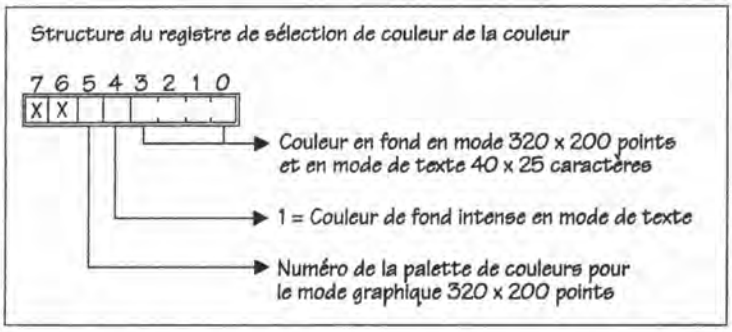
A ce registre vient s'ajouter aussi un registre d'état qui se rapproche beaucoup du registre d'état de la carte monochrome. La figure suivante montre la structure de ce registre, qui figure à l'adresse 3DAh. Il peut être lu mais on ne peut y écrire.

Structure du registre d'état d'une carte d'écran couleur

7	6	5	4	3	2	1	0
X	X	X	X				

- Signal de synchronisation horizontal
  - 0 = momentanément désactivé
  - 1 = momentanément activé
- 1 = Crayon optique inhibé
- 0 = Crayon optique activé
- Signal de synchronisation vertical
  - 0 = momentanément désactivé
  - 1 = momentanément activé

La carte CGA dispose également d'un registre, le registre de sélection de couleur qui ne figure pas, fort logiquement, sur les cartes vidéo monochromes. Il figure à l'adresse 3D9h et il est seulement possible d'y écrire mais pas de le lire.



La signification des différents bits de ce registre dépend du mode d'affichage. Dans les modes de texte, les 4 bits inférieurs sélectionnent la couleur du cadre de l'écran, en définissant le numéro de l'une des 16 couleurs disponibles. En mode graphique de 320 points sur 200, ils indiquent en outre la couleur de tous les points écran représentés par la combinaison de bits 00b (On parle alors également de couleur du fond).

La palette pour le mode graphique 320*200 points est sélectionnée à travers le bit 5. Si ce bit vaut 1, la première palette de couleur (turquoise, violet, blanc) sera sélectionnée, sinon ce sera la seconde (vert, jaune, rouge).

Synchronisation de l'accès à l'écran

Pour qu'un programme ne vienne pas perturber le contrôleur CRT lors de l'accès à la RAM vidéo, il est indispensable de synchroniser l'opération. Le bit 0 du registre d'état de la carte CGA apporte une solution. S'il contient la valeur 1, cela signifie que le rayon électronique se trouve déjà en retour horizontal. Puisqu'aucun point n'est encore sorti sur l'écran à ce moment précis, le contrôleur CRT n'accède pas à la RAM vidéo. Un programme peut donc adresser tranquillement la RAM vidéo à cet instant donné.

Pour éviter ce type de problèmes, il est conseillé de faire précéder tout accès à la RAM vidéo d'une carte CGA d'une brève séquence de code, composée de deux boucles, qui ne se terminera que précisément au moment où le CRTC commencera le retour horizontal du rayon électronique. Dans la première boucle, le contenu du registre d'état sera lu autant de fois que nécessaire jusqu'à ce qu'aucun retour horizontal ne soit en cours, c'est-à-dire jusqu'à ce que le bit 0 contienne la valeur 0. Lorsque cette condition sera remplie commencera le parcours de la seconde boucle, qui testera également le registre d'état de façon répétitive. Cela se fera jusqu'à ce que le bit 0 contienne la valeur 1 et indique ainsi qu'un retour horizontal du rayon électronique est actuellement en cours.

Cette procédure peut sembler un peu compliquée à première vue. La raison pour laquelle une première boucle attend la fin d'un retour horizontal n'a rien d'évident puisqu'il s'agit justement d'accéder à la RAM vidéo lors d'un retour horizontal. Cette première boucle

est destinée à s'assurer que la seconde boucle se termine bien au tout début d'un retour horizontal. Sans la première boucle, il pourrait fort bien arriver que la seconde boucle se termine au beau milieu du retour horizontal, voire peu avant qu'il ne s'achève. Or dans ce cas, la collision avec le CRTC qu'il s'agit d'éviter pourrait malgré tout se produire et l'on n'aurait donc abouti à rien. Il faut donc boire le calice jusqu'à la lie et faire précéder toutes les routines d'accès à la RAM vidéo de cette petite boucle de test, qui figure d'ailleurs le programme d'exemple présenté à la fin de cette section.

Il ne suffit cependant pas d'intégrer cette routine pour pouvoir accéder à la RAM vidéo aussi longtemps qu'on le souhaite car lorsque le retour horizontal se termine, au bout de 4 microsecondes environ, l'accès à la RAM vidéo doit être terminé. Le nombre d'octets pouvant être transférés dans l'intervalle entre la RAM vidéo et le processeur dépend de la rapidité du processeur et d'autres facteurs, comme par exemple la largeur du bus de données. Il est cependant conseillé de se placer dans la pire des hypothèses, c'est-à-dire en supposant qu'on a affaire à un PC exploité avec une fréquence de 4,77 MHz. Dans ce cas, ce sont exactement deux octets, autrement dit un caractère ASCII et son attribut, qui peuvent être transmis pendant le retour horizontal.

Une autre possibilité consiste naturellement à utiliser pour l'accès à la RAM vidéo, au lieu du retour horizontal, la phase de repos qui intervient lors du retour vertical. Cette phase durant 7 millisecondes, elle permet de transférer jusqu'à 800 octets. Le retour vertical est cependant, bien évidemment, beaucoup plus rare que le retour horizontal, au point que cette méthode est finalement moins rapide que l'attente du retour horizontal.

Ces deux procédés ralentissent naturellement considérablement la sortie sur écran. Cela est d'autant plus regrettable que cette méthode serait tout à fait inutile sur beaucoup de cartes CGA qui, comme les autres cartes vidéo présentées ici, disposent d'une RAM vidéo à laquelle le CRTC et le processeur peuvent accéder en même temps. Il vous faut donc organiser votre code programme de telle façon qu'un flag permette de contourner la séquence de code que nous venons de décrire. Le programme n'a bien sûr aucune possibilité de tester si la carte utilisée a tendance à trembloter, mais on peut confier ce soin à l'utilisateur en lui permettant de spécifier lors de l'appel du programme un Switch qui réponde à cette question. C'est ainsi que procède le programme d'exemple présenté à la fin de cette section.

Une méthode encore plus radicale pour éliminer tout tremblotement de l'écran consiste à interdire toute sortie d'image pour la durée de l'accès à la RAM vidéo. Cela n'est cependant conseillé que lorsque de nombreux accès à la RAM vidéo sont effectués immédiatement à la suite les uns des autres, comme cela se produit par exemple lors d'un scrolling de l'écran. C'est à cette méthode qu'a par exemple recours le BIOS du PC lors de l'appel des fonction 06h et 07h de l'interruption vidéo du BIOS (défilement de l'écran vers le haut ou vers le bas) lorsque la carte vidéo utilisée est une carte CGA. Cela est réalisé en désactivant le bit 3 du registre de sélection de mode avant l'accès à la RAM vidéo et en le réactivant après cet accès. Ce procédé produit un "effet de film muet", qu'on peut observer par exemple lorsqu'on fait défiler l'écran vers le haut à partir de l'environnement DOS.

Le contrôleur CRT de la carte CGA

Sur cette carte, l'accès aux 18 registres internes du 6845 se déroule exactement comme sur la carte MDA, à l'aide de deux instructions OUT 8 bits consécutives. La seule différence par rapport à la carte MDA est que les registres d'index et de données ne figurent pas ici aux adresses 3B4h et 3B5h mais en 3D4h et 3D5h respectivement. La table suivante récapitule les différentes valeurs que doivent présenter ces registres pour les différents modes d'affichage :

Contenu des registres CRTC en modes de texte 40x25 (T1), 80x25 (T2) et en modes graphiques (G) de la carte CGA

Registre	Signification	T1	T2	G
00h	Total de caractères horizontalement	56	113	56
01h	Caractères affichés horizontalement	40	80	40
02h	Signal de synchronisation horizontal après ... caractères	45	90	45
03h	Durée du signal horizontal de synchronisation en caractères	10	10	10
04h	Total de caractères verticalement	31	31	127
05h	Nombre ajusté de caractères verticalement	6	6	6
06h	Caractères affichés verticalement	25	25	100
07h	Signal de synchronisation vertical après ... caractères	28	28	112
08h	Mode d'entrelacement	2	2	2
09h	Nombre de lignes de grille (de balayage) par ligne de l'écran	7	7	1
0Ah	Ligne de départ du curseur clignot.de l'écran	6	6	6
0Bh	Ligne de fin du curseur clignotant de l'écran	7	7	7
0Ch	Adresse de départ de la page écran affichée (octet de poids fort)	0	0	0
0Dh	Adresse de départ de la page écran affichée (octet de poids faible)	0	0	0
0Eh	Adresse de caractère du curseur clignotant de l'écran (octet de poids fort)	0	0	0
0Fh	Adresse de caractère du curseur clignotant de l'écran (octet de poids faible)	0	0	0
10h	Position du crayon optique (octet de poids fort)	?	?	?
11h	Position du crayon optique (octet de poids faible)	?	?	?

? = dépend de la position du crayon optique

Contrairement à la carte MDA, qui ne dispose que de la place mémoire pour une page écran, l'adresse de base de la page écran affichée sur l'écran peut être ici fixée à l'aide des registres 12 et 13. Ils spécifient en effet cette adresse sous forme d'un offset par rapport au début des 16 Ko de RAM installés sur la carte (adresse de segment B800). Le registre 12 reçoit les 8 bits de poids fort et le registre 13 les 8 bits de poids faible de cet offset de 16 bits de largeur. Ces deux registres contiennent la valeur 0 après le lancement du système, de sorte que c'est la première page écran, celle qui commence à l'adresse B800:0000, qui est affichée sur l'écran.

Programme d'exemple

Le dernier des trois programmes de ce chapitre sert à accéder à la carte couleur. La seule différence importante par rapport aux deux programmes précédents réside dans le fait que l'accès à la RAM vidéo peut être synchronisé avec la construction de l'image par le contrôleur vidéo, comme nous l'avons décrit plus haut.

Listing : VCOL.ASM

```
;*****************************************************************;
;*                            V C O L                            *;
;*---------------------------------------------------------------*;
;*   Fonction      : fournit quelques fonctions de base pour     *;
;*                   accès à la carte écran couleur.             *;
;*                                                               *;
;*---------------------------------------------------------------*;
;*   Infos         : Toutes les fonctions considèrent que l'écran *;
;*                   comporte 80 colonnes numérotées de 0 à 79 et *;
;*                   25 lignes numérotées de 0 à 24 en mode texte, *;
;*                   640 lignes numérotées de 0 à 639 et 200 lignes *;
;*                   numérotées de 0 à 199 en mode graphique     *;
;*                   Le mode texte en 40 colonnes n'est pas supporté*;
;*---------------------------------------------------------------*;
;*   Auteur        : MICHAEL TISCHER                             *;
;*   Développé le  : 13.08.87                                    *;
;*   Dernière MAJ  : 21.04.89                                    *;
;*---------------------------------------------------------------*;
;*   Assemblage    : MASM VCOL;                                  *;
;*                   LINK VCOL;                                  *;
;*---------------------------------------------------------------*;
;*   Appel         : VCOL                                        *;
;*****************************************************************;

;-- Constantes ---------------------------------------------------

REG_CONTR   = 03D8h                 ; Adr. port du registre de commande
REG_COUL    = 03D9h     ; Adr. port de registre de sélection des couleurs
ADRESS_6845 = 03D4h                 ; Registre d'adresse du 6845
DATA_6845   = 03D5h                 ; Registre de données du 6845
VIO_SEG     = 0B800h                ; Segment de la mémoire d'écran
CUR_START   = 10        ; N° registre CRTC: ligne supérieure du curseur
CUR_END     = 11        ; N° registre CRTC: ligne inférieure du curseur
PAGECOUR_HI = 12              ; Octet fort de l'adresse de la page
PAGECOUR_LO = 13             ; Octet faible de l'adresse de la page
CURPOS_HI   = 14          ; N° registre CRTC: octet fort pos. curseur
CURPOS_LO   = 15         ; N° registre CRTC: octet faible pos. curseur

DELAY       = 20000      ; Compteur d'itérations pour boucle d'attente


;-- SETMODE: Macro pour régler le registre de commande de l'écran ------

setmode   macro mode

      mov  dx,REG_CONTR     ; Adresse du registre de commande de l'écran
      mov  al,mode                  ; Nouveau mode dans le registre AL
      out  dx,al                 ; Envoyer mode au registre de commande

          endm


;-- WAITRET: Attend que l'écran soit dessiné ---------------------------

waitret   macro
local     wr1                                              ; Label local

      mov  dx,3DAh            ; Adresse du registre d'état de l'écran
wr1:  in   al,dx                                    ; Lit son contenu
      test al,8                                  ; Retrace vertical ?
      je   wr1                                    ; NON --> attendre

          endm

;-- Pile ---------------------------------------------------------------

stack segment para stack              ; Définition du segment de pile

      dw 256 dup (?)                    ; La pile comporte 256 mots

stack ends                                ; Fin du segment de pile

;-- Données ------------------------------------------------------------

data      segment para 'DATA'         ; Définition du segment de données

;-- Données pour le programme de démonstration -------------------------

initm   db 13,10
        db " VCOL (c) 1987,1988 by Michael Tischer "
        db 13,10,13,10
        db "Ce programme de démonstration ne fonctionne qu'avec ",13,10
        db "une carte graphique CGA. Si votre PC est équipé d'un ",13,10
        db "autre type de carte, tapez S pour sortir du programme",13,10
        db "sinon tapez une autre touche pour lancer le "
        db "programme",13,10,"$"

strl    db 1,0

;-- Table des offsets des débuts de ligne ------------------------------

lignes  dw 0*160, 1*160, 2*160, 3*160, 4*160, 5*160, 6*160
        dw 7*160, 8*160, 9*160,10*160,11*160,12*160,13*160
        dw 14*160,15*160,16*160,17*160,18*160,19*160,20*160
        dw 21*160,22*160,23*160,24*160

grafit  db 38h, 28h, 2Dh, 0Ah, 7Fh, 06h          ; Valeurs de registre
        db 64h, 70h, 02h, 01h, 06h, 07h      ; pour les modes graphiques

textt   db 71h, 50h, 5Ah, 0Ah, 1Fh, 06h          ; Valeurs de registre
        db 19h, 1Ch, 02h, 07h, 06h, 07h           ; pour le mode texte
```

```
attn    db 0                            ; TRUE (<>0) si l'utilisateur
                                        ; indique le paramètre /F

data    ends                            ; Fin du segment de données

;-- Code ---------------------------------------------------------

code  segment para 'CODE'               ; Définition du segment de code

      assume cs:code, ds:data, es:data, ss:stack


;-- Ceci est simplement le programme de démonstration------------------


demo      proc far

      ;-- Détecte le paramètre /F dans la ligne de commande ------------

      mov  cl,ds:128            ; Lit nombre d'octets dans l. de comm.
      or   cl,cl                        ; Aucun paramètre ?
      je   switch1              ; OUI --> Evite l'exploitation des para
      mov  bx,129               ; BX pointe sur le premier octet de com
      mov  ch,bh                        ; Octet fort du compteur à 0

switch:
      cmp  [bx],"F/"                    ; Le paramètre est-il ici ?
      je   switch1                      ; OUI--> On a trouvé
      cmp  [bx],"f/"                    ; Se présente-t-il ainsi ?
      je   switch1                      ; OUI--> On a trouvé
      inc  bl                   ; BX pointe sur le caractère suivant
      loop switch                       ; Passe au caractère suivant

switch1:
      mov  ax,data                      ; Lit le segment de données
      mov  ds,ax                        ; et le charge en DS
      mov  es,ax                                ; et en ES

      mov  attn,cl                      ; Fixe l'indicateur WAIT

      ;-- Affiche le message et attend une frappe     ----------------

      mov  ah,9                 ; Num de la fonction "Afficher chaîne"
      mov  dx,offset fnitm                ; Adresse du message
      int  21h                  ; Appelle l'interruption de DOS

      xor  ah,ah                ; Numéro de la fonction "Lire touche"
      int  16h                  ; Appelle l'interruption clavier
      cmp  al,"s"                       ; A-t-on tapé <s> ?
      je   fin                  ; OUI--> on termine le programme
      cmp  al,"S"                       ; A-t-on tapé <S> ?
      jne  startdemo                ; NON  --> on démarre la démo

fin:  mov  ax,4C00h             ; Numéro de la fonction Terminer programme
      int  21h                  ; Appelle l'interruption 21h de DOS

      ;-- Programme de démonstration de l'appel des fonctions----------

startdemo  label near

      call grafhi               ; Active le mode graphique 320*200
      xor  al,al
      call cgr                          ; Efface l'écran

      xor  bx,bx                              ; Colonne 0
      xor  dx,dx                              ; Ligne 0
      mov  ax,199                       ; Nb points vertic.
      mov  cx,639                       ; Nb points horiz.

gr1:  push cx                     ; Empile nb points horiz.
      mov  cx,ax                  ; Nb points vertic. dans compteur
      push ax                     ; Empile nb points vertic.
      mov  al,1

gr2:  call pixhi                        ; Dessine un point
      inc  dx                           ; Incrémente la ligne
      loop gr2                          ; Trace une droite
      pop  ax                   ; Reprend nb points vertic.
      sub  ax,3                 ; Ligne suivante 3 points de moins
      pop  cx                   ; Reprend nb points horiz. holen
      push cx                           ; Empile nb points horiz.
      push ax                   ; Empile nb points vertic.
      mov  al,1

gr3:  call pixhi                        ; Dessine un point
      inc  bx                           ; Incrémente la colonne
      loop gr3                          ; Trace une droite
      pop  ax                   ; Reprend nb points vertical
      pop  cx                   ; Reprend nb points horizontal
      sub  cx,6                 ; Ligne suivante 6 points de moins
      push cx                           ; Empile nb points horizontal
      mov  cx,ax                ; Nb points vertic. dans compteur
      push ax                           ; Empile nb points vertical
      mov  al,1

gr4:  call pixhi                        ; Dessine un point
      dec  dx                           ; Décrémente la ligne
      loop gr4                          ; Trace une droite
      pop  ax                   ; Reprend nb points vertical
      sub  ax,3                 ; Ligne suivante 3 points de moins
      pop  cx                   ; Reprend nb points horizontal
      push cx                   ; Mémorise nb points horizontal
      push ax                   ; Mémorise nb points vertical
      mov  al,1

gr5:  call pixhi                        ; Dessine un point
      dec  bx                           ; Incrémente la colonne
      loop gr5                          ; Trace une droite
      pop  ax                   ; Reprend nb points vertical
      pop  cx                   ; Reprend nb points horizontal
      sub  cx,6                 ; Ligne suivante 6 points de moins
      cmp  ax,5                         ; Ligne verticale sup à 5
      ja   gr1                          ; OUI --> on continue

      xor  ah,ah                ; Numéro de la fonct. "Attendre frappe"
      int  16h                  ; Appelle l'interruption clavier

      call text                         ; Active le mode texte 80*25
      xor  bp,bp                        ; D'abord la page d'écran 0

demo1:
      mov  al,30h                             ; Code Ascii "0"
      or   ax,bp                ; Transforme numéro de page en ASCII
      mov  str1,al                      ; Mémorise dans chaîne
      call setcol                       ; Fixe la couleur
      call pge                  ; Active la page d'écran en BP
      call cls                          ; Efface l'écran
      xor  bx,bx                        ; Commence l'affichage
      call calo                         ; au coin sup gauche
      mov  cx,2000              ; Une page = 2000 caractères
      xor  ah,ah                ; On commence avec la couleur 0
      mov  si,offset str1               ; Offset de la chaîne 1

demo2:
      inc  ah                           ; Incrémente la couleur
      call print                        ; Affiche la chaîne 1
      loop demo2                ; Répète jusqu'à écran soit rempli

      xor  ah,ah                        ; Attend une touche
      int  16h                  ; Appelle l'interruption clavier
      inc  bp                           ; Incrémente la page d'écran
      cmp  bp,4                 ; A-t-on traité toutes les 4 pages ?
      jne  demo1                ; NON --> on passe à la suivante

      xor  bp,bp                        ; Active à nouveau la page 0
      call pge
      jmp  fin                                ; C'est terminé

demo      endp

;-- Voici les fonctions proprement dites ------------------------------

;-- TEXT     : Activer l'affichage de texte ----------------------------
;-- Entrée   : Aucune
;-- Sortie   : Aucune
;-- Registres: AX, SI, BH, DX et registre des indicateurs sont modifiés

text      proc near

      mov  si,offset textt              ; Offset de la table des registres
      mov  bl,00100001b                       ; Mode texte 80*25, clignotant
      jmp  short vcprog                 ; Reprogramme le contrôleur vidéo

text      endp

;-- GRAFHI   : Active le mode graphique 640*200 points ----------------
;-- Entrée   : Aucune
;-- Sortie   : Aucune
;-- Registres: AX, SI, BH, DX et registre des indicateurs sont modifiés

grafhi    proc near

      mov  bl,00010010b                       ; Mode graphique 640*200 points
      jmp  short graphiq                ; Reprogramme le contrôleur vidéo

grafhi    endp

;-- GRAFLO   : Activer le mode graphique 320*200 points ---------------
;-- Entrée   : Aucune
;-- Sortie   : Aucune
;-- Registres: AX, SI, BH, DX et registre des indicateurs sont modifiés

graflo    proc near

      mov  bl,00100010b                       ; Mode graphique 320*200 points
graphiq:
      mov  si,offset grafit             ; Offset de la table des registres
```

```
;
;graflo    endp
;
;;-- VCPROG   : programme le contrôleur vidéo ----------------------------
;;-- Entrée   : SI = Adresse d'une table de registres
;;--            BL = Valeur pour registre de commande de l'écran
;;-- Sortie   : Aucune
;;-- Registres: AX, SI, BH, DX et registre des indicateurs sont modifiés
;
;vcprog    proc near
;
;     setmode bl                          ; Bit 3 = 0 : Ecran désactivé
;
;     mov  cx,12                               ; 12 registres à fixer
;     xor  bh,bh                          ; Commence par le registre 0
;
;vcp1: lodsb                         ; Lit une valeur de registre dans table
;     mov  ah,al                          ; Valeur de registre dans AH
;     mov  al,bh                          ; Numéro du registre dans AL
;     call setvc                    ; Transmet la valeur au contrôleur
;     inc  bh                             ; Appelle le registre suivant
;     loop vcp1                           ; Fixe les autres registres
;
;     or   bl,8                           ; Bit 3 = 1 : Ecran activé
;     setmode bl                                ; Fixe nouveau mode
;     ret                               ; Retour au programme appelant
;
;vcprog    endp
;
;;-- SETCOL   : fixe la couleur du cadre et du fond de l'écran ----------
;;-- Entrée   : AL = valeur de couleur
;;-- Sortie   : Aucune
;;-- Registres: AX et DX sont modifiés
;;-- Infos    : En mode texte les 4 bits inférieurs fixent la couleur du
;;--            cadre. En mode graphique les 4 bits inférieurs indiquent
;;--            les couleurs de cadre et de fond, le bit 5 sélectionne
;;--            la palette de couleurs
;
;setcol    proc near
;
;     mov  dx,REG_COUL       ; Adresse registre de sélection de couleur
;     out  dx,al                        ; Sort la valeur de la couleur
;     ret                               ; Retour au programme appelant
;
;setcol    endp
;
;;-- CDEF     : Délimite la forme du curseur -------------------------------
;;-- Entrée   : CL = ligne de trame de départ
;;--            CH = ligne de trame de fin
;;-- Sortie   : Aucune
;;-- Registres: AX et DX sont modifiés
;
;cdef      proc near
;
;     mov  al,CUR_START                   ; Registre : ligne de départ
;     mov  ah,cl                             ; Ligne de départ dans AH
;     call setvc                    ; Communique avec le contrôleur vidéo
;     mov  al,CUR_END                       ; Registre : ligne de fin
;     mov  ah,ch                               ; Ligne de fin dans AH
;     jmp  short setvc              ; Communique avec le contrôleur vidéo
;
;cdef      endp
;
;;-- PGE      : fixe la page d'écran affichée ------------------------------
;;-- Entrée   : BP = Numéro de la page d'écran (0 à 3)
;;-- Sortie   : Aucune
;;-- Registres: BX, AX, CX et DX sont modifiés
;;-- Infos    : dans les modes graphiques, la première page d'écran
;;--            porte le numéro 0, la seconde le numéro 2
;
;pge       proc near
;
;     mov  bx,bp                          ; Page d'écran dans BX
;     mov  cl,5                              ; Multiplie par 2048
;     ror  bx,cl
;     mov  al,PAGECOUR_HI           ; Registre : Octet fort adresse de page
;     mov  ah,bh                    ; Octet fort de la page d'écran dans AH
;     call setvc                      ; Communique avec le contrôleur vidéo
;     mov  al,PAGECOUR_LO           ; Registre : Octet faible de page écran
;     mov  ah,bl                    ; Octet faible de la page écran dans AH
;     jmp  short setvc                ; Communique avec le contrôleur vidéo
;
;pge       endp
;
;;-- SETCLIGN : Positionne le curseur clignotant de l'écran -------------
;;-- Entrée   : DI = Offset du curseur
;;-- Sortie   : Aucune
;;-- Registres: BX, AX et DX sont modifiés
;
;setclign  proc near
;
;     mov  bx,di                          ; Transfère offset dans BX
;     mov  al,14                    ; Registre 14 : MSB offset curseur
;     mov  ah,bh                             ; Octet fort de l'offset
;     call setvc                    ; Communique avec le contrôleur vidéo
;     mov  al,15                       ; Registre 15 : LSB offset curseur
;     mov  ah,bl                               ; Octet faible de l'offset
;
;     ;-- SETVC est appelée automatiquement ----------------------------
;
;setclign  endp
;
;;-- SETVC    : fixe un octet dans l'un des registres du contrôleur vidéo
;;-- Entrée   : AL = Numéro du registre
;;--            AH = Nouveau contenu du registre
;;-- Sortie   : Aucune
;;-- Registres: DX et AL sont modifiés
;
;setvc     proc near
;
;     mov  dx,ADRESS_6845                 ; Adresse du registre d'index
;     out  dx,al                        ; Envoye le numéro du registre
;     jmp  short $+2                    ; Petite pause d'entrée-sortie
;     inc  dx                            ; Adresse du registre d'index
;     mov  al,ah                                  ; Contenu dans AL
;     out  dx,al                             ; Fixe le nouveau contenu
;     ret                               ; Retour au programme appelant
;
;setvc     endp
;
;;-- GETVC    : lit un octet dans l'un des registres du contrôleur vidéo
;;-- Entrée   : AL = Numéro du registre
;;-- Sortie   : AL = Contenu du registre
;;-- Registres: DX et AL sont modifiés
;
;getvc     proc near
;
;     mov  dx,ADRESS_6845                 ; Adresse du registre d'index
;     out  dx,al                        ; Envoye le numéro du registre
;     inc  dx
;     jmp  short $+2
;     in   al,dx                        ; Transfère le contenu dans AL
;     ret                               ; Retour au programme appelant
;
;getvc     endp
;
;;-- SCROLLUP : fait défiler une fenêtre de N lignes vers le haut -------
;;-- Entrée   : BL = ligne supgauche
;;--            BH = colonne sup gauche
;;--            DL = ligne inf droite
;;--            DH = colonne inf droite
;;--            CL = Nombre de lignes de défilement
;;--            BP = Numéro de la page d'écran (0 à 3)
;;-- Sortie   : Aucune
;;-- Registres: seuls le registre des indicateurs est modifié
;;-- Infos    : les lignes de l'écran libérées sont vierges
;
;scrollup  proc near
;
;     cld                            ; Sens d'incr des chaînes de caractères
;     push ax                          ; Sauve tous les registres modifiés
;     push bx                                              ; sur la pile
;     push di                                ; Dans ce cas, l'ordre doit
;     push si                                             ; être respecté !
;
;     push bx                        ; Ces trois registres sont retirés de
;     push cx                              ; la pile avant même la fin de
;     push dx                                                ; la routine
;     sub  dl,bl                              ; Calculer le nombre de lignes
;     inc  dl
;     sub  dl,cl                     ; Retranche nb. de lignes de défilement
;     sub  dh,bh                              ; Calcule le nombre colonnes
;     inc  dh
;     call calo                      ; Convertit coin sup gauche en offset
;     mov  si,di                                ; Range l'adresse dans SI
;     add  bl,cl                          ; Première ligne dans la fenêtre
;     call calo                      ; Convertit première ligne en offset
;     xchg si,di                                        ; Echange SI et DI
;     cmp  attn,0                    ; Faut-il empêcher le tremblement ?
;     je   sup0                                          ; NON --> SUP0
;
;     waitret                             ; OUI, attend le rafraîchissement
;     setmode 00100101b                             ; Désactive l'écran
;
;sup0: push ds                                ; Sauve registres de segment
;     push es                                                ; sur la pile
;     mov  ax,VIO_SEG                ; Charge le segment de la mémoire écran
;     mov  ds,ax                                             ; en DS
;     mov  es,ax                                             ; et ES
;
;sup1: mov  ax,di                                      ; Range DI dans AX
;     mov  bx,si                                       ; Range SI dans BX
;     mov  cl,dh                     ; Nombre de colonnes dans le compteur
;     rep  movsw                                      ; Décale une ligne
;     mov  di,ax                                      ; Déduit DI de AX
;     mov  si,bx                                      ; Déduit SI de BX
;     add  di,160                                      ; Ligne suivante
;     add  si,160
```

```
        dec  dl                         ; Toutes les lignes traitées ?
        jne  sup1                       ; NON --> décale encore une ligne
;
        pop  es                         ; Reprend les registres de segment
        pop  ds                                 ; sur la pile
;
        cmp  attn,0                     ; Faut-il empêcher le tremblement ?
        je   sup2                               ; NON --> SUP2
;
        setmode 00101101b                       ; OUI, rétablit l'écran
;
sup2:   pop  dx                         ; Récupère coin inf. droit
        pop  cx                         ; Récupère nombre de lignes
        pop  bx                         ; Récupère coin sup. gauche
        mov  bl,dl                      ; Ligne inférieure dans BL
        sub  bl,cl                      ; Retranche le nombre de lignes
        inc  bl
        mov  ah,07h                             ; Couleur : noir sur blanc
        call clear                      ; Efface les lignes libérées
;
        pop  si                                 ; CX et DX ont déjà été
        pop  di                                         ; récupérés
        pop  bx
        pop  ax
;
        ret                             ; Retour au programme appelant
;
scrollup  endp
;
;-- SCROLLDN : fait défiler une fenêtre de N lignes vers le bas --------
;-- Entrée   : BL = ligne sup gauche
;--            BH = colonne sup gauche
;--            DL = ligne inf droite
;--            DH = colonne inf droite
;--            CL = Nombre de lignes de défilement
;--          : BP = Numéro de la page écran (0 à 3)
;-- Sortie   : Aucune
;-- Registres: seuls le registre des indicateurs est modifié
;-- Infos    : les lignes de l'écran libérées sont vierges
;
scrolldn  proc near
;
        cld                             ; Sens d'incr des chaines de caractères
;
        push ax                         ; Sauve tous les registres modifiés
        push bx                                 ; sur la pile
        push dl                         ; Dans ce cas l'ordre doit
        push sl                                 ; être respecté !
;
        push bx                         ; Ces trois registres sont retirés de
        push cx                                 ; la pile avant même la fin de
        push dx                                         ; la routine
;
        sub  dh,bh                              ; Calcule le nombre colonnes
        inc  dh
        mov  al,bl                      ; Range ligne sup gauche dans AL
        mov  bl,dl                              ; Ligne i.d. dans ligne i.g.
        call calo                       ; Convertit coin sup gauche en offset
        mov  sl,dl                              ; Range l'adresse dans SI
        sub  bl,cl                      ; Retranche nb de ligne de défilement
        call calo                       ; Convertit coin sup gauche en offset
        xchg sl,dl                              ; Echange SI et DI
        sub  dl,al                              ; Calcule le nombre de lignes
        inc  dl
        sub  dl,cl                      ; Retranche nb de lignes de défilement
;
        cmp  attn,0                     ; Faut-il empêcher le tremblement ?
        je   sdn0                               ; NON --> SDN0
;
        waitret                         ; OUI, attend le rafraichissement
        setmode 00100101b                       ; désactive l'écran
;
sdn0:   push ds                         ; Sauve les registres de segment
        push es                                 ; sur la pile
        mov  ax,VIO_SEG                 ; Charge le segment de la mémoire écran
        mov  ds,ax                                      ; en DS
        mov  es,ax                                      ; et en ES
;
sdn1:   mov  ax,dl                                      ; Range DI dans AX
        mov  bx,sl                                      ; Range SI dans BX
        mov  cl,dh                      ; Nombre de colonnes dans le compteur
        rep  movsw                                      ; Décale une ligne
        mov  dl,ax                                      ; Déduit DI de AX
        mov  sl,bx                                      ; Déduit SI de BX
        sub  dl,160                                     ; Ligne suivante
        sub  sl,160
        dec  dl                         ; Toutes les lignes traitées ?
        jne  sdn1                       ; NON --> décale encore une ligne
        pop  es                         ; Récupère les registres de segment
        pop  ds                                 ; sur la pile
;
        cmp  attn,0                     ; Faut-il empêcher le tremblement ?
        je   sdn2                               ; NON --> SDN2
;
        setmode 00101101b                       ; OUI, remet l'écran en service
;
sdn2:   pop  dx                                 ; Récupère coin inf. droit
        pop  cx                                 ; Récupère nombre de lignes
        pop  bx                                 ; Récupère coin sup. gauche
        mov  dl,bl                              ; Ligne supérieure dans DL
        add  dl,cl                      ; Additionne le nombre de lignes
        dec  dl
        mov  ah,07h                             ; Couleur : noir sur blanc
        call clear                              ; Efface les lignes libérées
;
        pop  si                                 ; CX et DX ont déjà été
        pop  di                                         ; récupérés
        pop  bx
        pop  ax
;
        ret                             ; Retour au programme appelant
;
scrolldn  endp
;
;-- CLS      : Efface l'écran dans son intégralité ---------------------
;-- Entrée   : BP = Numéro de la page d'écran (0 ou 1)
;-- Sortie   : Aucune
;-- Registres: seuls le registre des indicateurs est modifiés
;
cls     proc near
;
        mov  ah,07h                             ; Couleur : blanc sur noir
        xor  bx,bx                              ; Coin sup gauche = (0/0)
        mov  dx,4F18h                           ; Coin inf droit = (79/24)
;
        ;-- Exécute Clear ------------------------------------------------
;
cls     endp
;
;-- CLEAR    : remplit d'espaces une zone d'écran déterminée -----------
;-- Entrée   : AH = attribut/couleur
;--            BL = ligne sup gauche
;--            BH = colonne sup gauche
;--            DL = ligne inf droite
;--            DH = colonne inf droite
;--            BP = Numéro de la page d'écran (0 à 3)
;-- Sortie   : Aucune
;-- Registres: seul le registre des indicateurs sont modifiés
;
clear   proc near
;
        cld                             ; Sens d'incr des chaines de caractères
        push cx                         ; Sauve sur la pile tous les registres
        push dx                                 ; qui seront modifiés par la suite
        push si
        push di
        push es
        sub  dl,bl                              ; Calcule le nombre de lignes
        inc  dl
        sub  dh,bh                              ; Calcule le nombre de colonnes
        inc  dh
        call calo                       ; Offset du coin supérieur gauche
        mov  cx,VIO_SEG                 ; Segment de la mémoire d'écran
        mov  es,cx                                      ; dans ES
        xor  ch,ch                      ; Octet fort du compteur à 0
        mov  al," "                                     ; Espace
        cmp  attn,0                     ; Faut-il empêcher le tremblement ?
        je   clear1                             ; NON --> CLEAR1
;
        push dx                                 ; Sauve DX sur la pile
        waitret                         ; Attend le rafraichissement
        setmode 00100101b                       ; Désactive l'écran
        pop  dx                                 ; Récupère DX sur la pile
;
clear1:
        mov  si,di                                      ; Range DI dans SI
        mov  cl,dh                      ; Nombre de colonnes dans le compteur
        rep stosw                                       ; Sauve espace
        mov  di,si                                      ; Déduit DI de SI
        add  di,160                                     ; Ligne suivante
        dec  dl                         ; Toutes les lignes traitées ?
        jne  clear1                     ; NON --> efface encore une ligne
;
        cmp  attn,0                     ; Faut-il empêcher le tremblement ?
        je   clear2                             ; NON --> CLEAR2
;
        setmode 00101101b                       ; Remet en service l'écran
;
clear2:
        pop  es                         ; Récupère sur la pile les registres
        pop  di                                         ; sauvegardés
        pop  si
        pop  dx
        pop  cx
        ret                                     ; Retour au programme appelant
```

```
;
clear   endp
;
;-- PRINT    : Affiche une chaîne sur l'écran ------------------------
;-- Entrée   : AH = Attribut/couleur
;--            DI = Offset du premier caractère
;--            SI = Offset de la chaîne par rapport à DS
;--            BP = Numéro de la page d'écran (0 à 3)
;-- Sortie   : DI désigne la position après le dernier caractère affiché
;-- Registres: AL, DI et le registres des indicateurs sont modifiés
;-- Infos    : La chaîne doit se terminer par le caractère NUL.
;--            Les autres caractères de contrôle ne sont pas reconnus.
;
print   proc near
;
        cld                             ; Sens d'incr des chaînes de caractères
        push si                         ; Sauve SI, DX et ES sur la pile
        push es
        push cx
        push dx
        mov  dx,VIO_SEG                 ; Charge le segment de la mémoire écran
;
        mov  cl,attn                            ; Lit l'indicateur WAIT
        mov  es,dx                              ; en DX puis en ES
;
        jmp  short print3               ; Lit et traite un caractère
;
print1  label near
;
        or   cl,cl                      ; Faut-il empêcher le tremblement ?
        je   print2                             ; NON --> PRINT2
;
        push ax                         ; Mémorise le caractère et sa couleur
        mov  dx,30Ah                    ; Adresse du registre d'état de l'écran
hr1:    in   al,dx                              ; Lit le contenu
;
        test al,1                       ; Rafraîchissement horizontal ?
        jne  hr1                                ; NON --> attendre
        cli                             ; Inhibe les interruptions
hr2:    in   al,dx                              ; Lit le contenu
        test al,1                       ; Rafraîchissement horizontal ?
        je   hr2                                ; OUI --> attendre
        pop  ax                         ; récupère caractère et couleur
        sti                                     ; rétablit les interruptions
;
print2:
        stosw                           ; Attribut et couleur dans la mém écran
print3:
        lodsb                           ; Passe au caract. suivant dans chaîne
        or   al,al                              ; est-ce le caractère NUL ?
        jne  print1                             ; NON --> afficher
;
printe:
        pop  dx                                 ; récupère SI, CX, DX et  ES
        pop  cx                                         ; sur la pile
        pop  es
        pop  si
        ret                                     ; Rend la main à l'appelant
;
print   endp
;
;
;-- CALO     : convertit ligne et colonne en adresse d'offset ----------
;-- Entrée   : BL = ligne
;--            BH = colonne
;--            BP = Numéro de la page d'écran (0 à 3)
;-- Sortie   : DI = l'adresse d'offset
;-- Registres: DI et le registre des indicateurs sont modifiés
;
calo    proc near
;
        push ax                                 ; Sauve AX sur la pile
        push bx                                 ; Sauve BX sur la pile
;
        shl  bx,1                               ; Colonne et ligne fois 2
        mov  al,bh                                      ; Colonne dans AL
        xor  bh,bh                                      ; Octet fort
        mov  di,[lignes+bx]                     ; Lire offset de ligne
        xor  ah,ah                      ; Octet fort pour offset colonne
        add  di,ax                      ; Additionne offsets ligne et colonne
        mov  bx,bp                              ; Page d'écran dans BX
        mov  cl,4                               ; Multiplie par 4096
        ror  bx,cl
        add  di,bx                      ; Additionne début de page écran offset
        pop  bx                                 ; Retire BX de la pile
        pop  ax                                 ; Retire AX de la pile
        ret                             ; Retour au programme appelant
;
calo    endp
;
;-- CGR      : Efface intégralement l'écran graphique ------------------
;-- Entrée   : AL = 00(h) : Effacer tous les points
;--                 FF(h) : Dessiner tous les points
;-- Sortie   : Aucune
;-- Registres: AH, BX, CX, DI et registre des indicateurs sont modifiés
;-- Infos    : Cette fonction efface l'écran graphique dans les deux
;--            modes graphiques
;
cgr     proc near
;
        push es                                 ; Sauve ES sur la pile
        cbw                                             ; Etend AL à AH
        xor  di,di                              ; Offset mémoire d'écran
        mov  bx,VIO_SEG                         ; Segment mémoire d'écran
        mov  es,bx                                      ; Segment dans ES
        mov  cx,2000h                   ; Une page représente 8K de mots
        rep  stosw                                      ; Remplit la page
        pop  es                                 ; Récupèr eES sur  la pile
        ret                                     ; Retour au programme appelant
;
cgr     endp
;
;-- PIXLO    : Dessine un point en mode graphique 320*200 points -------
;-- Entrée   : BP = Numéro de la page d'écran (0 ou 1)
;--            BX = Colonne (0 à 319)
;--            DX = Ligne (0 à 199)
;--            AL = Couleur du point (0 à 3)
;-- Sortie   : Aucune
;-- Registres: AX, DI et le registre des indicateurs sont modifiés
;
pixlo   proc near
;
        push ax                                 ; Sauve AX sur la pile
        push bx                                 ; Range BX sur la pile
        push cx                                 ; Range CX sur la pile
        mov  cl,7
        mov  ah,bl                              ; Transfère colonne dans AH
        and  ah,11b                                     ; Colonne mod 4
        shl  ah,1                                       ; Colonne * 2
        sub  cl,ah                              ; 7 - 2 * (Colonne mod 4)
        mov  ah,11                                      ; Masque de bits
        shl  ax,cl                      ; Décale vers la position du point
        not  ah                                 ; Inverse AH pour masquage
        shr  bx,1                       ; Décale BX deux fois vers la droite
        shr  bx,1                               ; pour diviser par 4
        jmp  short spix                         ; Dessine le point
;
pixlo   endp
;
;-- PIXHI    : Dessine un point en mode graphique 640*200 points -------
;-- Entrée   : BP = Numéro de la page d'écran (0 ou 1)
;--            BX = Colonne (0 à 639)
;--            DX = Ligne (0 à 199)
;--            AL = Couleur du point (0 ou 1)
;-- Sortie   : Aucune
;-- Registres: AX, DI et le registre des indicateurs sont modifiés
;
pixhi   proc near
;
        push ax                                 ; Sauve AX sur la pile
        push bx                                         ; idem pour BX
        push cx                                                 ; et CX
;
        mov  cl,7
        mov  ah,bl                              ; Transfère la colonne dans AH
        and  ah,111b                                    ; Colonne mod 8
        sub  cl,ah                                      ; 7 - Colonne mod 8
        mov  ah,1                                       ; Masque de bits
        shl  ax,cl                      ; Décale vers la position du point
        not  ah                                 ; Inverse AH pour le masquage
        mov  cl,3                                       ; 3 décalages
        shr  bx,cl                                      ; Divise BX par 8
;
        ;-- Dessine le point ------------------------------------------
;
pixhi   endp
;
;-- SPIX     : Dessine un point sur l'écran graphique ------------------
;-- Entrée   : BX = Offset de colonne
;--            DX = Ligne (0 à 199)
;--            AH = Valeur pour masquer anciens bits
;--            AL = Nouvelle valeur de bit
;-- Sortie   : Aucune
;-- Registres: AX, DI et le registre des indicateurs sont modifiés
;
spix    proc near
;
        push es                                 ; Sauve ES sur la pile
        push dx                                 ; Sauve DX sur la pile
        push ax                                 ; Sauve AX sur la pile
;
        xor  di,di                              ; Offset mémoire d'écran
        mov  cx,VIO_SEG                         ; Segment mémoire d'écran
        mov  es,cx                                      ; Charge ES
        mov  ax,dx                              ; Amène ligne dans AX
        shr  ax,1                               ; Divise ligne par 2
```

```
        mov  cl,80                      ; Le facteur est 90
        mul  cl                     ; Multiplie ligne par 80
        and  dx,1                       ; Ligne mod 2
        mov  cl,3                       ; 3 décalages
        ror  dx,cl          ; Rotation vers la droite (* 2000(h))
        mov  di,ax                      ; 80 * int(Ligne/2)
        add  di,dx                ; + 2000(h) * (Ligne mod 4)
        add  di,bx              ; Additionne offset de colonne
        pop  ax                         ; Retire AX de la pile
        mov  bl,es:[di]                   ; Lit le point
        and  bl,ah                    ; Masque les bits
        or   bl,al                    ; Insère le point
        mov  es:[di],bl               ; Réécrit le point

        pop  dx                         ; Retire DX de la pile
        pop  es                         ; Retire ES de la pile
        pop  cx                         ; Retire CX de la pile
        pop  bx                         ; Retire BX de la pile
        pop  ax                         ; Retire AX de la pile
        ret                         ; Retour au programme appelant

spix    endp

_main   proc near
        jmp demo
_main   endp

;------------ Fin ------------------------------------------------
code    ends                                ; Fin du segment de CODE
        end  demo                 ; Début d'exécution du programme
```

4.8. Programmation des cartes EGA/VGA

Par leurs possibilités pour l'affichage graphique mais aussi pour l'affichage de texte, les cartes EGA/VGA surclassent largement ses prédécesseurs. Il existe bien sûr depuis longtemps des systèmes graphiques aux possibilités comparables dans d'autres secteurs informatiques (Stations de travail, application DAO etc.), mais c'est la première fois que ces systèmes entrent dans une catégorie de prix telle qu'ils feront bientôt partie de l'équipement standard indispensable d'un PC.

L'intérêt de cette nouvelle génération de cartes vidéo provient non seulement du fait qu'elles offrent des modes graphiques avec une résolution inconnue jusqu'ici, qu'elles permettent presque d'afficher sur l'écran autant de lignes de texte qu'on le souhaite, mais surtout de leur souplesse et de leur aptitude à émuler d'autres cartes vidéo.

Il n'est pas étonnant que ces possibilités étendues ne puissent être obtenues que grâce à une plus grande complexité des structures d'organisation. Concrètement, cela signifie que les caractéristiques d'une carte EGA ne peuvent plus être obtenues avec le contrôleur vidéo traditionnel dans le secteur des PC, le Motorola 6845.

Sur les cartes EGA/VGA, ce dernier est remplacé par quatre contrôleurs qui se partagent les tâches nécessaires liées à la création d'une image vidéo. Il s'agit du contrôleur vidéo, du Sequencer, du contrôleur graphique et du contrôleur d'attributs. Dans une carte VGA, il s'y ajoute en outre le DAC qui est capable de convertir un code de couleur digital en un signal analogique.

La grande complexité des cartes EGA/VGA complique d'autant la programmation directe des divers contrôleurs et registres. Elle s'accentue davantage quand on s'aperçoit que tous les constructeurs de cartes compatibles EGA ou VGA ne s'alignent pas en fait sur le standard IBM. Comme vous le saurez en lisant les paragraphes suivants, on ne parvient pas toujours à échapper à la programmation directe des registres lorsqu'on souhaite profiter pleinement des facultés offertes par ces cartes. Il est vrai que les fonctions du BIOS EGA et VGA, qui a été très augmenté par rapport au BIOS vidéo normal, nous soutiennent dans bien des circonstances, mais il existe toujours des situations où leurs services restent insuffisants.

Les paragraphes de cette section présentent les diverses performances des cartes EGA et VGA qui les distinguent de leurs prédécesseurs. Dans le domaine du texte, il faut signaler en particulier la faculté de ces cartes à utiliser des jeux de caractères quelconques comme décrit au paragraphe 4.8.2. Dans le domaine graphique, il faut noter les nouveaux modes graphiques à 16 couleurs et même un mode 256 couleurs avec la carte VGA. Dans ce contexte, nous vous fournirons des astuces permettant par exemple d'obtenir une résolution double à partir d'une carte VGA normale en mode 256 couleurs.

A propos de la carte VGA originale : Bien qu'IBM ait imposé des standards avec ses cartes EGA et VGA, il n'en reste pas moins que leurs spécificités originales sont oubliées depuis déjà fort longtemps. Avec les modes étendus, les constructeurs ont au contraire tenté d'adopter une norme standard avec des résolutions allant jusqu'à 1024*768 points, si bien que chaque constructeur propose son propre modèle. La section 4.9 essaye de contourner le problème en évoquant leurs ressemblances et comment en tirer profit.

Dans cette section, le thème Sprites est également à l'ordre du jour puisqu'il apparaît pour la première fois dans le monde PC avec les cartes EGA et VGA. Le paragraphe 4.8.9 montre comment l'utiliser pour programmer des animations.

Dans la plupart des paragraphes, nous parlerons de la programmation des registres et le rôle qu'ils jouent dans l'accomplissement de certaines tâches. Le dernier paragraphe énumère tous les registres des cartes EGA et VGA et décrit en détail leurs fonctions.

Ce chapitre contient également la partie descriptive des fonctions du BIOS EGA et VGA étendu incluse également en annexe. La plupart des fonctions citées dans les divers paragraphes sont certes décrites dans cette section, mais il est impossible de les évoquer en détail. Par conséquent, n'hésitez jamais à les consulter si vous n'êtes pas sûr des services rendus par le BIOS EGA/VGA étendu avant de vous lancer dans la programmation directe du registre électronique concerné.

4.8.1. A propos du moniteur

Lorsqu'on parle de moniteur avec les cartes EGA et VGA, tout un chacun s'attend à obtenir une multitude de couleurs avec une résolution de 800*600 points ou davantage selon les propos tenus par un vendeur de PC. Il est important de savoir que la carte vidéo n'assure pas à elle seule la création de l'image mais c'est surtout le moniteur qui en est le maître d'oeuvre. D'ailleurs, il faut réfléchir profondément si l'on souhaite bénéficier de 800*600 points avec 256 couleurs différentes si possible.

Le moniteur installé joue un rôle important dans le cadre des modes graphiques ainsi que pour le fonctionnement général d'une carte EGA/VGA. Cette section apporte des réponses aux questions suivantes :

- ✓ Comment la connexion d'un moniteur précis peut-elle influer sur une carte EGA ou VGA ?

✓ Comment déterminer le type d'une carte et le type du moniteur installé pendant l'exécution d'un programme ?

Eu égard à l'interrogation Multiscan ou non Multiscan, il est primordial de savoir si le moniteur relié à la carte est monochrome ou couleur. Les possibilités en ce domaine sont variées avec les cartes EGA et VGA.

Les moniteurs des cartes EGA

Les cartes EGA peuvent être connectées à un moniteur CGA, EGA ou Multisync ainsi qu'à un moniteur monochrome MDA simultanément. Selon le moniteur en présence, elles se comportent alors comme une carte MDA étendue ou une carte EGA ordinaire. La commutation ne se reproduit pas seulement dans l'image vidéo, mais aussi et surtout dans les registres internes de la carte EGA et la RAM vidéo. En mode monochrome, la RAM vidéo ne se trouve plus comme à l'accoutumée à l'adresse B800h, mais à B000h. Les octets d'attribut des caractères de la RAM vidéo sont en outre interprétés comme avec une carte MDA alors que l'état des registres d'index et de données du contrôleur CRT subit des changements pour qu'il corresponde au standard MDA.

Connectée à un moniteur monochrome, la carte EGA se comporte ainsi parfaitement comme une carte MDA. Nous n'insisterons donc pas sur cette configuration dans les paragraphes qui suivent. Dans ce livre, lorsqu'on parle de la carte EGA, il s'agit toujours d'une carte EGA reliée à un moniteur haute résolution. Cela est d'autant plus valable que la plupart des cartes EGA compatibles ne peuvent plus être connectées à un moniteur monochrome contrairement au modèle original d'IBM, mais fonctionnement plutôt avec des moniteurs EGA ou Multisync.

Le plus étonnant est que les cartes EGA d'IBM, livrées au début avec une RAM vidéo de 64 Ko pouvant être dotées par la suite d'un module RAM coûteux de 128, 192 et 256 Ko, ont complètement disparu du marché. Les cartes EGA modernes sont fournies avec 256 Ko de RAM tout comme les cartes VGA. Elles représentent d'ailleurs le modèle standard faisant l'objet des sections suivantes.

Les moniteurs des cartes VGA

Il en va autrement des cartes VGA. Elles peuvent être connectées à un moniteur monochrome quoiqu'il ne s'agisse pas d'un moniteur MDA quelconque. La carte VGA exige en effet un moniteur VGA analogique. Bien qu'il soit moins onéreux qu'un moniteur couleur VGA, il est néanmoins capable de traiter les hautes résolutions des cartes VGA.

Un tel moniteur n'est toutefois pas en mesure de couvrir l'éventail des 256 couleurs affichables simultanément, parce qu'il existe seulement 64 niveaux différents de gris.

Cette lacune ne devient apparente que lors de l'utilisation d'un grand nombre de couleurs variées dont les codes couleur présentent peu de différences dans la zone du vert.*

L'avantage du moniteur monochrome VGA par rapport à son homologue couleur est qu'une carte VGA peut être activée en mode monochrome avec ce moniteur. Dans ce cas, elle se comporte comme une carte MDA particulièrement performante. Tout comme une carte EGA, la RAM vidéo ne se trouve plus à l'adresse B800h, mais à B000h sans oublier que les adresses d'autres registres peuvent être adaptées au standard MDA.

La commutation de la carte s'effectue à l'aide des commandes DOS MODE MONO ou MODE CO80 ou tout simplement avec la fonction 00h de l'interruption vidéo du BIOS spécifiant un mode vidéo précis. A l'exclusion du mode vidéo 07h utilisable également sur les cartes MDA, il symbolise notamment le mode d'exploitation monochrome alors que les autres modes tendent à transformer la carte VGA en une véritable carte couleur.

Le tableau suivant mentionne les différents moniteurs utilisés par les cartes EGA et VGA pour afficher les modes vidéo.

Les modes vidéo des cartes VGA

Code	Mode	MONO	CGA	EGA/VGA
00h	40*25 caractères de texte, 16 couleurs		■	■
01h	40*25 caractères de texte, 16 couleurs		■	■
02h	80*25 caractères de texte, 16 couleurs		■	■
03h	80*25 caractères de texte, 16 couleurs		■	■
04h	320*200 points graphiques, 16 couleurs		■	■
05h	320*200 points graphiques, 16 couleurs		■	■
06h	640*200 points graphiques, 2 couleurs		■	■
07h	80*25 caractères, monochrome	■		
0Dh	320*200 points graphiques, 16 couleurs			■
0Eh	320*200 points graphiques, 16 couleurs			■
0Fh	640*350 points graphiques, monochrome	■		
10h	640*350 points graphiques, 16 couleurs +			■
11h	640*680 points graphiques, 2 couleurs			■*

* *Parmi les trois codes couleur RVB, les moniteurs analogiques monochromes ignorent tout simplement les deux codes couleur R(ouge) et B(leu). Ils ne permettent donc de décider que du code V(ert) à travers la couleur d'un point.*

■ Les modes vidéo des cartes VGA				
Code	Mode	MONO	CGA	EGA/VGA
12h	640*680 points graphiques, 16 couleurs			■*
13h	320*200 points graphiques, 256 couleurs			■*
■ * Carte VGA + Les cartes EGA de 64 Ko de RAM ne permettent de représenter que 4 couleurs				

Les facultés d'une carte VGA en mode monochrome étant devenues fortement limitées en raison des progrès techniques considérables, nous considérerons dans cette section que votre carte VGA est déjà en place pour assurer l'exécution des programmes d'exemple en mode couleur. Une considération similaire concerne la carte EGA où nous supposons la présence d'une RAM de 256 Ko et la connexion à un moniteur EGA ou Multisync.

Dans vos programmes, si vous tenez absolument à vérifier que la carte installée est une carte EGA ou VGA, reportez-vous alors à la section suivante où vous apprendrez à intégrer une routine appropriée dans vos programmes.

Identifier les cartes EGA et VGA

Dans vos programmes, si vous utilisez toutes sortes de techniques propres aux cartes EGA et VGA et déterminant éventuellement la présence d'un type de moniteur précis, vous devez inclure une routine appropriée au début de votre programme. Pendant l'exécution d'un programme, vous pourrez vous en servir pour faire la différence entre les cartes EGA/VGA et leurs prédécesseurs. Nous allons vous présenter une telle routine dans cette section.

Elle porte le nom IsEgaVga. Elle est écrite en Pascal et en C. Sa base forme l'appel de deux fonctions BIOS contenues uniquement dans le BIOS étendu des cartes EGA/VGA. Cette méthode permet d'ores et déjà de comparer les anciens modèles depuis MDA à CGA car ces appels de fonctions doivent obligatoirement échouer à cet endroit.

L'une des deux fonctions citées n'est disponible qu'avec une carte VGA et aide par conséquent à faire la différence entre les cartes EGA et VGA. Il s'agit de la fonction 1Ah munie de deux sous-fonctions. La sous-fonction qui nous intéresse est 00h puisqu'elle retourne des informations sur la carte vidéo active et passive.

Dans ce contexte, nous parlons de carte vidéo active et passive parce que dans de nombreux PC, on peut utiliser une carte supplémentaire en plus d'une carte VGA. Celle-ci doit alors être une carte MDA ou une carte Hercules. Cette faculté reste toutefois inconnue dans la plupart des modèles PS/2 où ni une carte MDA ni une carte Hercules n'est disponible pour le bus MCA.

Si vous appelez maintenant la fonction 1Ah avec les numéros de fonction et sous-fonction dans les registres AH et AL, le contenu du registre AL vous indique immédiatement après l'appel si la fonction est reconnue par le BIOS. Vous pouvez en conclure qu'une carte VGA et un BIOS VGA approprié sont implantés. Le BIOS VGA confirme la situation car le BIOS normal est retourné immédiatement comme résultat de l'appel d'une fonction inconnue sans pour autant changer le contenu des registres.

Dans le registre AL, le BIOS normal retourne la valeur inchangée 00h alors que le BIOS VGA charge exprès le numéro de fonction 1Ah dans ce registre pour rendre compte de la bonne exécution de l'appel. Le code du registre BL indiquant la carte vidéo active et le code du registre BH la carte vidéo passive peut être interprété de la manière suivante :

Code	Signification
00h	Pas de carte vidéo
01h	Carte MDA sur un moniteur MDA
02h	Carte CGA sur un moniteur CGA
03h	Réservé
04h	Carte EGA sur un moniteur couleur EGA
05h	Carte EGA sur un moniteur MDA
06h	Réservé
07h	Carte VGA sur un moniteur monochrome analogique
08h	Carte VGA sur un moniteur couleur analogique
09h	Réservé
0Ah	MCGA sur une carte CGA
0Bh	MCGA sur un moniteur monochrome analogique
0Ch	MCGA sur un moniteur couleur analogique

Si l'appel de la fonction 1Ah échoue, tous les rêves fondés sur les possibilités merveilleuses d'une carte VGA sont à l'eau. Il faut en conclure en fait sur la présence d'une carte EGA qui sera prouvée dans l'étape suivante. Dans ce cas, la sous-fonction 10h de la fonction 12h soutenue uniquement par les cartes EGA s'avère nécessaire. Contrairement à beaucoup de fonctions BIOS, le numéro de sous-fonction n'est pas à transmettre dans le registre AL mais BL. Après l'appel, si le numéro de sous-fonction 10h se trouve encore présent, cela prouve l'absence d'une carte EGA.

Dans un autre cas de figure, le contenu du registre BH indique si la carte EGA est reliée à un moniteur couleur MDA ou EGA. La valeur 0 confirme le dernier cas et la valeur 1 symbolise l'existence d'un moniteur MDA. Le registre BL informe par ailleurs sur l'organisation de la RAM de la carte selon le tableau suivant :

Code	RAM de la carte EGA
00h	64 Ko
01h	128 Ko
02h	192 Ko
03h	256 Ko

Voici deux programmes d'exemple ISEVP.PAS et ISEVC.C qui contiennent une version de la fonction IsEgaVga et démontrent en même temps leur fonctionnement dans le cadre d'un programme. Vous retrouvez cette fonction dans d'autres programmes fournis dans ce livre mais sous une forme modifiée.

Listing : ISEVP.PAS

```
{*********************************************************************
*                          I S E V P . P A S                         *
**-------------------------------------------------------------------**
*  Fonction: teste la présence d'une carte EGA ou VGA.               *
**-------------------------------------------------------------------**
*  Auteur          : MICHAEL TISCHER                                 *
*  Développé le    : 05.08.1990                                      *
*  Dernière MAJ    : 14.01.1991                                      *
*********************************************************************}

Program IsEgaVgaP;

uses DOS;                                       { Pour les Interruptions }

{-- Déclarations de types ---------------------------------------------}

type CARTE = ( EGA_MONO, EGA_COLOR, VGA_MONO, VGA_COLOR, NINI );

{*********************************************************************
*  IsEgaVga : Teste la présence d'une carte EGA ou VGA               *
**-------------------------------------------------------------------**
*  Entrée : néant                                                    *
*  Sortie : Type de carte selon le code CARTE                        *
*********************************************************************}

function IsEgaVga : CARTE;

var Regs : Registers;               { Registres pour gérer l'Interruption }

begin
  Regs.AX := $1a00;               { La fonction 1Ah n'existe que pour VGA }
  Intr( $10, Regs );
  if ( Regs.AL = $1a ) then          { La fonction est-elle disponible ? }
    case Regs.BL of                                  { Oui, exploite le code }
        4 : IsEgaVga := EGA_COLOR;
        5 : IsEgaVga := EGA_MONO;
        7 : IsEgaVga := VGA_MONO;
        8 : IsEgaVga := VGA_COLOR;
    else IsEgaVga := NINI;
  end
  else                                    { Pas de VGA, mais peut-être EGA }
  begin
    Regs.ah := $12;                                  { Invoque l'option 10h }
    Regs.bl := $10;                                   { de la fonction 12h }
    intr($10, Regs);                                  { Interruption vidéo }
    if ( Regs.bl <> $10 ) then                    { Est-ce une carte EGA ? }
      begin                                 { Oui, recherche l'écran associé }
        if Regs.BH = 0 then IsEgaVga := EGA_COLOR
                       else IsEgaVga := EGA_MONO;
      end
    else IsEgaVga := NINI;
  end;
end;

{*********************************************************************
*                          PROGRAMME PRINCIPAL                       *
*********************************************************************}

begin
  writeln( 'ISEVP  -  (c) 1990 by MICHAEL TISCHER'#13#10 );
  case IsEgaVga of
    NINI : writeln( 'La carte vidéo active n''est ni une carte ' +
                    'EGA ni une carte VGA !');

    EGA_MONO  : writeln( 'La carte active est une carte EGA branchée'+
                         ' sur un écran MDA.');

    EGA_COLOR : writeln( 'La carte active est une carte EGA branchée'+
                         ' sur un écran EGA ou Multiscan .' );

    VGA_MONO  : writeln( 'La carte active est une carte VGA branchée'+
                         ' sur un écran monochrome analogique.');

    VGA_COLOR : writeln( 'La carte active est une carte VGA branchée'+
                         ' sur un écran VGA ou Multiscan .' );
  end;
end.
```

Listing : ISEVC.C

```
/*****************************************************************
*                          I S E V C . C                         *
******************************************************************
* Fonction        : Teste la présence d'une carte EGA ou VGA     *
*                                                                *
******************************************************************
* Auteur          : MICHAEL TISCHER                              *
* Développé le    : 6.08.1990                                    *
* Dernière MAJ    : 14.02.1992                                   *
******************************************************************
* (MICROSOFT C)                                                  *
* Compilation     : CL /AS isevc.c                               *
******************************************************************
* (BORLAND TURBO C)                                              *
* Compilation     : par l'EDI                                    *
******************************************************************
* Appel           : isevc                                        *
*****************************************************************/

#include <dos.h>                                /* Fichiers d'en-tête */
#include <stdarg.h>
#include <stdio.h>

/*-- Constantes ----------------------------------------------------*/

#define EGA_MONO   0                        /* EGA avec moniteur MDA */
#define EGA_COLOR  1                        /* EGA avec moniteur EGA */
#define VGA_MONO   2        /* VGA avec moniteur monochrome analogique */
#define VGA_COLOR  3                        /* VGA avec moniteur VGA */
#define NINI       4                              /* NI EGA NI VGA */

/*-- Déclarations de types -----------------------------------------*/

typedef unsigned char    BYTE;

/*****************************************************************
* IsEgaVga : teste la présence d'une carte EGA ou VGA            *
******************************************************************
* Entrée   : néant                                               *
* Sortie   : l'une des constantes EGA_MONO, EGA_COLOR etc.       *
*****************************************************************/

BYTE IsEgaVga( void )
{
  union REGS   Regs;                    /* Registres pour l'interruption */

  Regs.x.ax = 0x1a00;           /* La fonction 1Ah n'existe que pour VGA */
  int86( 0x10, &Regs, &Regs );
  if( Regs.h.al == 0x1a )         /* La fonction est-elle disponible ? */
    switch ( Regs.h.bl )                        /* Oui, exploite le code */
    {
      case 4  : return EGA_COLOR;
      case 5  : return EGA_MONO;
      case 7  : return VGA_MONO;
      case 8  : return VGA_COLOR;
      default : return NINI;
    }
  else                                    /* Non, serait-ce une EGA ? */
  {
    Regs.h.ah = 0x12;                          /* Appelle l'option 10h */
    Regs.h.bl = 0x10;                           /* de la fonction 12h */
    int86(0x10, &Regs, &Regs );                 /* Interruption vidéo */
    if( Regs.h.bl != 0x10 )                                   /* EGA? */
      return Regs.h.bh == 0 ? EGA_COLOR : EGA_MONO;           /* oui */
    else                                                      /* Non */
      return NINI;
  }
}

/*****************************************************************
*                        PROGRAMME PRINCIPAL                     *
*****************************************************************/

void main( void )
{
  printf( "ISEVC - (c) 1990, 92 by MICHAEL TISCHER\n\n" );
  switch( IsEgaVga() )
  {
    case NINI     :
      printf( "La carte vidéo active n'est ni une carte EGA"\
              " ni une carte VGA !");
      break;

    case EGA_MONO :
      printf( "La carte active est une carte EGA branchée"\
              " sur un écran MDA." );
      break;

    case EGA_COLOR :
      printf( "La carte active est une carte EGA branchée"\
              " sur un écran EGA ou Multiscan." );
      break;

  case VGA_MONO  :
      printf( "La carte active est une carte VGA branchée sur"\
              " un écran monochrome analogique.");
      break;

  case VGA_COLOR :
      printf( "La carte active est une carte VGA branchée sur"\
              " un écran VGA ou Multiscan." );
  }
  printf( "\n\n" );
}
```

4.8.1. Sélection et programmation de jeux de caractères

Comme nous l'avons indiqué au début du chapitre, les cartes EGA et VGA ne sont pas tributaires des divers jeux de caractères stockés dans une structure ROM de la carte, contrairement à leurs prédécesseurs. Elles disposent en revanche d'un générateur de caractères particulièrement performant. Il tire les informations concernant l'aspect des caractères à partir d'une zone précise de la RAM vidéo dans laquelle il dépose le modèle de bits des caractères selon un schéma immuable. Cette section donne un aperçu sur la construction et le fonctionnement des tables de caractères et toutes les techniques dont on peut tirer profit. Bref, voici les thèmes présentés :

- ✓ Chargement et définition de jeux de caractères avec le BIOS
- ✓ Commutation entre l'affichage en 9 points et 8 points
- ✓ Création de logos par regroupement de caractères
- ✓ Structure et état des tables de caractères

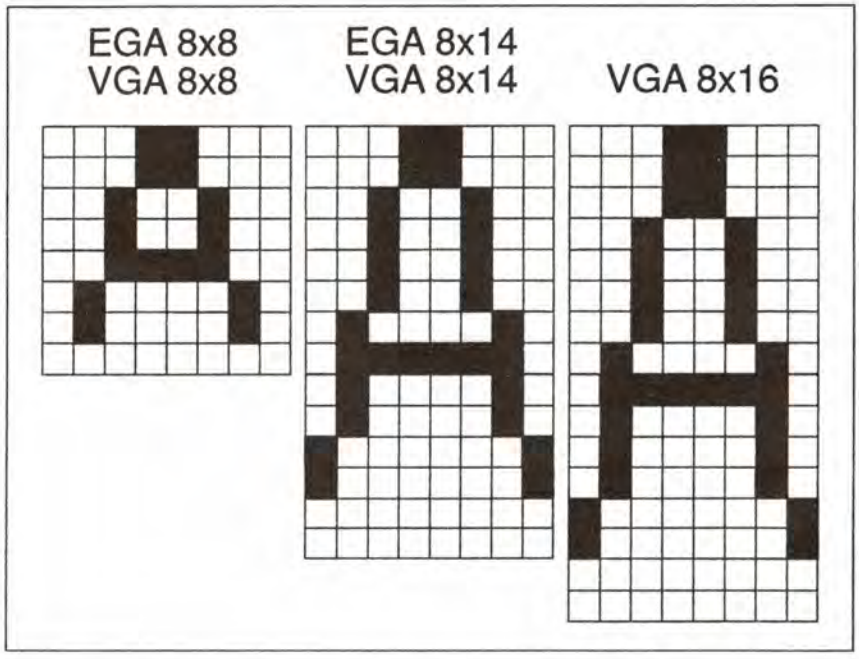
- ✓ Commutation entre plusieurs jeux de caractères ou Affichage simultané de 512 caractères variés
- ✓ Une fenêtre graphique en mode texte
- ✓ Jeux de caractères pour le mode graphique

Charger et définir des jeux de caractères avec le BIOS

Le BIOS étendu des cartes EGA et VGA contient des fonctions permettant d'agir sur les diverses tables de caractères. Il existe en tout une douzaine de sous-fonctions à appeler à travers la fonction 11h de l'interruption vidéo du BIOS. Comme à l'accoutumée, le numéro de fonction est à transmettre dans le registre AH et le numéro de sous-fonction dans le registre AL.

Ces fonctions servent à charger des tables de caractères prédéfinies ou personnelles, à les afficher ou commuter entre elles. Les tables prédéfinies sont des jeux de caractères stockés dans la ROM de la carte EGA ou VGA. De là, ils peuvent être copiés dans la zone de la RAM vidéo d'où le générateur de caractères obtient les informations nécessaires sur l'apparence des divers caractères.

Les cartes EGA et VGA comportent des différences de ce point de vue. Deux jeux de caractères seulement sont disponibles dans la ROM avec une carte EGA alors que la carte VGA dispose d'un troisième jeu pouvant être activé par le BIOS. La différence entre les jeux de caractères ne porte pas pour autant sur le type de la police comme c'est le cas avec une imprimante. Ici, la différence concerne également la taille des caractères, ou mieux la matrice, qui est à la base même des caractères.



Construction des caractères sous forme d'une matrice

Normalement, la carte EGA utilise par exemple la matrice 8*14 points décrite ci-dessus. Elle contient un second jeu dans la ROM dont les caractères doivent se serrer à l'intérieur d'un champ étroit de 8*8 points. La taille d'un point écran restant inchangée quel que soit le jeu implanté, il va sans dire que les caractères de ce jeu occupent nettement moins de place, mais sont à peine visibles en raison de la faible résolution. Quel intérêt y-a-il donc à utiliser ce jeu de caractères ?

La réponse à cette question se trouve sur le revers de la médaille. La taille infime des caractères aide en effet à afficher un grand nombre de caractères parce que la résolution globale de l'écran reste naturellement inchangée. Ainsi, le minuscule jeu de 8*8 permet d'afficher 43 lignes de texte au lieu des 25 lignes habituelles si bien que l'utilisateur obtient simultanément une quantité d'informations intéressante.

La résolution de 25 lignes sur 43 n'a pas été sélectionnée par hasard. Elle résulte en fait de la résolution verticale d'une carte EGA qui s'élève toujours à 350 points en mode texte. Comme le montre la figure suivante, la mise en place d'un jeu 8*8 devrait en fait provoquer l'affichage de 43,75 lignes sur l'écran. Mais cela importe peu puisqu'on se limite ici à 43 lignes.

```
Résolution EGA verticale :        350 points     350 points
Hauteur de la matrice :           14 points      8 points

Nombre des lignes de texte :      25 lignes      43,75 lignes
```

Ici, la carte EGA dispose fondamentalement de 640 points. Divisés par 8 correspondant à la largeur de caractères utilisée dans les deux jeux, on obtient les fameuses 80 colonnes par ligne de texte.

Il en va autrement avec la carte VGA tant du point de vue horizontal que vertical. En mode texte, la résolution verticale ne s'élève pas ici à 350 mais 480 points. En mode texte normal de 80*25 caractères, le jeu EGA avec ses 8*14 points devient inutile puisqu'un jeu spécial VGA avec 816 points entre en action. Mais les deux jeux EGA restent disponibles comme auparavant. Avec une carte VGA, ils résultent avec une résolution écran de 28 lignes sur 50.

La résolution de la carte VGA est supérieure du point de vue horizontal puisque 720 points viennent s'afficher par ligne d'écran au lieu de 640. Mais malgré tout, une ligne ne contient que 80 caractères parce que les caractères occupent 9 points horizontalement et non 8. Nous reviendrons sur le neuvième point ultérieurement.

Revenons aux fonctions permettant de charger les jeux de caractères et spécifier ainsi automatiquement le nombre de lignes écran. Elles portent les numéros 11h, 12h et 14h. Mis à part les deux numéros de fonctions dans le registre AX, elles attendent un autre argument dans le registre BL. Il reproduit le numéro de la table de caractères dans laquelle le jeu spécifié est à charger puis activer. Si vous ne souhaitez pas travailler simultanément avec plusieurs tables, vous devez indiquer ici la valeur 0 pour la première table. Sinon, vous pouvez choisir entre les valeurs 0 et 3 avec les cartes EGA et les valeurs 0 et 7 avec les cartes VGA. Pour de plus amples informations sur les tables de caractères, lisez les sections suivantes.

Sous-fonction	Matrice	Lignes EGA	Lignes VGA
11h	8*14	25	28
12h	8*8	43	50
14h	8*16	X	25

Ces fonctions permettent de charger un jeu de caractères précis et programmer simultanément les divers registres de la carte vidéo de manière à afficher ce jeu et un nombre correspondant de lignes écran. Mais les jeux désignés peuvent être aussi tout simplement chargés dans une table de caractères. S'il ne s'agit pas en l'occurrence de la table en cours, l'affichage écran ne subit aucune influence puisque ni le jeu ni le nombre de lignes nécessaires ne sera défini. Leur appel est identique à celui des trois fonctions déjà décrites en ce qui concerne l'affectation des registres.

Sous-fonction	Matrice	EGA	VGA
1h	8*14	■	■
2h	8*8	■	■
4h	8*16		■

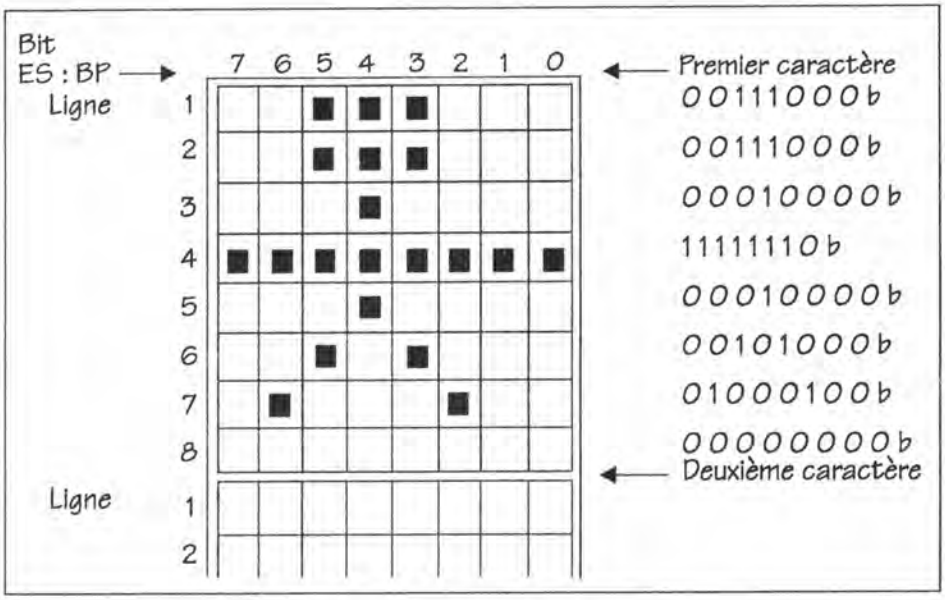
Le BIOS permet de charger dans la RAM vidéo des tables de caractères déjà stockées dans la ROM, mais il dispose également de fonctions destinées spécialement aux jeux définis personnellement. Ainsi, vous pouvez redéfinir les différents caractères ou sélectionner carrément une autre police comme c'est le cas avec une imprimante.

Le clou de l'opération est que la hauteur des caractères peut être choisie librement entre 1 et 32 points où le nombre de lignes affichées sur l'écran s'ajuste automatiquement en conséquence. Votre imagination est toutefois limitée ici pour des raisons optiques. En effet, les caractères nécessitant moins de six lignes de points restent à peine lisibles. Par ailleurs, avec plus de 16 lignes de points, les différences entre les caractères deviennent plus faciles à corriger puisque le nombre de lignes affichées sur l'écran diminue d'autant. En règle générale, il convient de sélectionner les caractères dans l'intervalle prédéfini par BIOS, c'est-à-dire entre 8 et 16 lignes de points, ce qui n'oblige pas l'utilisateur à s'habituer à une résolution tout à fait étrangère.

Dans ce contexte, le BIOS vous donne cependant carte blanche. Lors de l'appel de la fonction 10h destinée à charger un jeu personnalisé, il faut transmettre la hauteur des points de caractères dans le registre BH. Vous n'êtes pas obligé de spécifier la largeur puisqu'elle reste toujours constante avec 8 points sur une carte EGA et 9 points sur une carte VGA.

Aucune contrainte ne vous est imposée quant au choix de la table dans laquelle vous devez charger le jeu de caractères. Avant d'appeler la fonction, chargez tout simplement le numéro de la table dans le registre BL. Dans le registre CX, la fonction attend le nombre de caractères à charger, car il suffit parfois de charger seulement quelques caractères dans une table existante et non la totalité d'une table. Pour que cela soit reconnu, vous devez indiquer le numéro (code ASCII) du premier caractère attendu dans le registre DX. Il se situe entre 0 et 255.

Le BIOS lit les informations concernant les motifs de points des caractères dans un buffer défini et initialisé préalablement par l'utilisateur. La fonction attend son adresse sous la forme d'un pointeur FAR dans la paire de registres ES:BP.



Structure du buffer pour l'appel de la sous-fonction 00h de la fonction 11h

D'après la figure précédente, le buffer ainsi obtenu doit contenir une entrée pour chaque caractère défini dont l'ampleur en octets doit correspondre à la hauteur des caractères. Si les caractères sont par exemple défini dans une matrice 8*12, chaque entrée se compose alors de 12 octets. La longueur totale du buffer résulte de la multiplication par 12 et du nombre de caractères à définir car les entrées sont consécutives dans le buffer. La première entrée à l'adresse d'offset 0000h lit l'information pour le premier caractère à définir, viennent ensuite les entrées relatives aux autres caractères en suivant un ordre croissant.

A l'intérieur des différentes entrées, chaque octet représente le motif de points pour une ligne de points du caractère en suivant évidemment un ordre croissant depuis la première jusqu'à la dernière ligne de points. A l'intérieur d'un tel octet, les divers bits reproduisent l'état des points d'une ligne de ce genre depuis la gauche vers la droite. Si un bit est réglé, le point écran correspondant est affiché dans la couleur de premier plan définie pour le caractère. Si le bit est effacé, le point apparaît dans la couleur d'arrière-plan appropriée.

La sous-fonction 00h fonctionne comme la sous-fonction 10h. Elle ne charge les définitions de caractères que dans la table de caractères sans l'activer ni ajuster le nombre de lignes de texte. Nous rencontrons à nouveau le dualisme évoqué dans les rapports entre les sous-fonctions 11h, 12h et 14h d'une part et les sous-fonctions 01h, 02h et 04 d'autre part.

Pour la définition des caractères et jeux de caractères, n'oubliez jamais que vos définitions ne sont valables que sur l'écran. Elles n'ont aucun effet sur l'imprimante car elle n'a

aucune aptitude à prendre connaissance de la modification d'aspect subie par les caractères. Ne vous étonnez donc pas si la hardcopy d'un écran ne ressemble nullement à l'image renvoyée par l'écran.

Commutation entre l'affichage 9 points et 8 points

Pour illustrer la programmation des jeux de caractères, nous allons prendre un exemple permettant d'afficher des logos sur l'écran composés à partir de plusieurs caractères. Un problème survient avec les cartes VGA ayant un rapport avec l'étirement horizontal des divers caractères. Sur une carte EGA, la largeur d'un caractère qui est de 8 points devient 9 points sur une carte VGA. Cette différence résulte de la diversité des résolutions horizontales qui s'élève à 720 points sur une carte VGA au lieu de 640 points.

D'où peut-on extraire le neuvième point sachant que les jeux définis dans la ROM ainsi que les jeux chargés individuellement à l'aide du BIOS ne contiennent que 8 points ? La réponse n'est pas difficile à trouver puisque le neuvième point reste généralement vide à une exception près : les caractères dotés des codes ASCII entre C0h et DFh. Cette zone contient les caractères de cadre parmi lesquels certains servent à tracer une ligne horizontale continue. En ce qui concerne ces caractères, le huitième point est tout simplement copié dans le neuvième pour créer un passage vers les caractères suivants.

L'utilisation d'un neuvième point avec une carte VGA procure aux caractères une résolution élevée en apparence même si seulement 8 points sont copiés dans la ROM avec des cartes EGA ou VGA. Etant donné qu'une petite place doit exister entre les divers caractères, on peut uniquement utiliser les sept premiers points avec des cartes EGA pour que le huitième point resté vide libère de la place pour le caractère suivant. En revanche, le huitième point peut entièrement être utilisé avec les cartes VGA parce que le neuvième point inséré avec adresse crée de la place pour le voisin.

Cette méthode d'action de la carte VGA ne comporte normalement aucun risque tant qu'on se contente de définir des caractères distincts n'ayant aucun lien avec les caractères avoisinants. Mais si on se met par exemple à construire une mosaïque à partir de différents caractères - pour créer des logos - on n'obtiendra pas un tracé fluide entre les éléments de la mosaïque puisqu'on a aucun contrôle sur le neuvième point. Dans un tel cas, le neuvième point doit donc être interdit et il faut activer l'affichage en 8 points tel qu'il a été défini avec la carte EGA.

Cette commutation fait entrer en action plusieurs registres de la carte VGA. Ces derniers ne devant malheureusement pas être réclamés à travers le BIOS, il devient par conséquent impossible de les programmer. Le premier registre concerné s'appelle "Miscellaneous Output Register" pouvant être lu via l'adresse de port 3CCh et inscrit via l'adresse de port 3C2h. Il contient deux bits (les bits 2 et 3) permettant de régler l'horloge de la carte VGA et par voie de conséquence la résolution horizontale.

Normalement, la carte VGA est exploitée en mode texte avec 28,322 MHz et affiche 720 points par ligne sur l'écran. Avec une fréquence de 25,175 MHz, le nombre de points se réduit à 640. Cela correspond à la résolution souhaitée car avec une largeur de caractère de 8 points, on obtient exactement 80 caractères par ligne d'écran. Pour qu'on puisse d'abord lire le registre nommé, modifier en conséquence les bits 2 et 3 et réinscrire la nouvelle valeur, il suffit de commuter la carte VGA sur la résolution horizontale souhaitée. Ce n'est là que la moitié du rendement puisque la carte VGA travaille ensuite avec 9 points par caractère et élimine très vite l'apparition de neige sur l'écran si on ne la commute pas sur 8 points.

Cela se produit également à l'étape suivante par simple modification du contenu du registre "Clocking Mode". Ce registre fait partie du contrôleur Sequencer et ne peut pas être adressé aussi directement que le registre Miscellaneous Output. Avant et après l'accès aux registre du contrôleur Sequencer, il faut notamment effectuer un reset de ce contrôleur au moyen de son registre Reset. Ce n'est qu'à cette condition que le registre Sequencer ainsi que le registre Clocking Mode dont le bit 0 vaut pour la largeur de caractère peuvent être modifiés.

En guise de dernière étape, il faut modifier le registre Horizontal Pel Panning qui a déjà été présenté dans le cadre du Smooth Scrolling. En d'autres termes, après avoir commuté vers l'affichage 8 points, ce registre doit être chargé avec la valeur 0 pour que l'image vidéo créée se décale d'un point vers la gauche par rapport à l'affichage normal. Si cela ne se produit pas, l'utilisateur apercevra en permanence un vascillement peu appréciable dans la première colonne de points. Ici, il n'est pas indispensable d'accéder directement au registre nommé parce qu'il existe une fonction BIOS destinée à cet effet.

La procédure qui provoque la commutation de la largeur de caractères depuis 9 points vers 8 points en trois étapes est naturellement réciproque. Ainsi, on peut retourner dans l'affichage 9 points selon la même méthode. A cet effet, il suffit d'activer une résolution horizontale de 720 points dans le registre Miscellaneous Output et l'affichage 9 points dans le registre Clocking Mode. Mais il ne faut oublier de régler le registre Horizontal Pel Panning sur 8 pour que la première colonne de points soit rendue à nouveau visible.

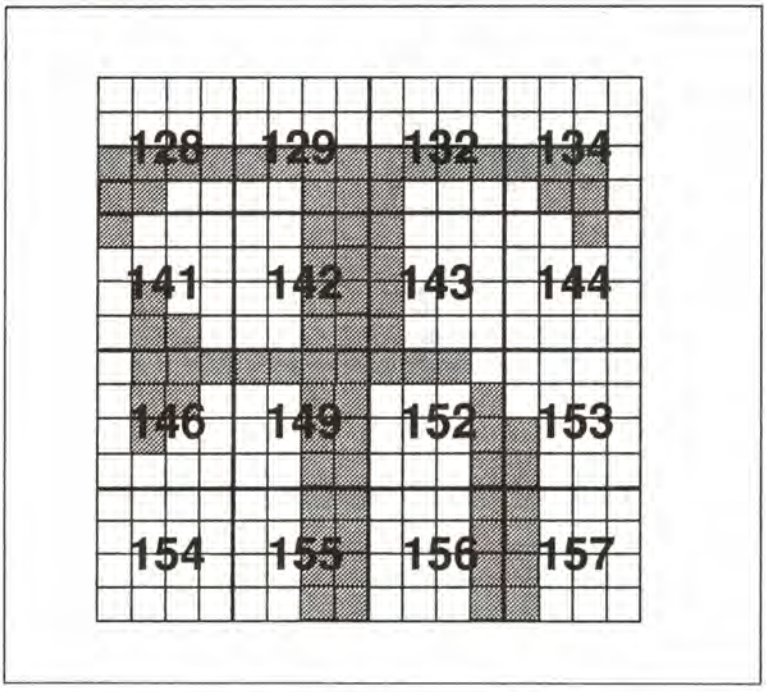
Si vous vous intéressez à la structure précise des registres cités, reportez-vous à la fin du chapitre où vous trouverez une description détaillée de tous les registres EGA et VGA. Les programmes d'exemple contiennent en outre une routine accomplissant la commutation selon les règles qui viennent d'être décrites.

Création de logos par regroupement de caractères

Revenons aux logos pour lesquels la largeur de caractères doit être ramenée à 8 points avec une carte VGA. Un tel logo rend en général un meilleur effet sur l'écran au lieu d'un message de copyright quelconque que l'on rencontre dans pratiquement tous les programmes. Mais les caractères prédéfinis dans le jeu ASCII ne permettent que très

rarement de construire un tel logo. Cela vaut naturellement pour les cartes EGA et VGA bien qu'elles offrent la possibilité de définir séparément l'aspect des divers caractères.

Un seul caractère suffit dans de très rares cas à construire un logo parce qu'une carte VGA offre 128 points (8*16) alors qu'une carte EGA ne propose que 112 points au dessinateur du logo. Soulignons en outre qu'un seul caractère affiché sur l'écran offre un spectacle plutôt déroutant. Pourquoi ne pas alors construire un logo avec des caractères multiples telle une mosaïque à l'intérieur d'un rectangle ?



Création du logo avec des caractères multiples

Cela augmente non seulement la résolution du logo, mais il est plus agréable à regarder puisqu'il occupe plus de place sur l'écran. Mais il ne faut pas non plus exagérer la taille du logo au risque de dépasser l'offre de caractères disponible dans le jeu ASCII. Il faut tout au moins que toutes les lettres et tous les nombres soient visibles sur l'écran malgré la présence du logo afin qu'un dialogue puisse s'établir avec l'utilisateur. Il ne s'agit pas ici de définir un jeu entièrement nouveau. Il suffit de redéfinir quelques caractères issus d'un jeu déjà défini pour obtenir l'effet souhaité.

Pour définir le logo, il serait judicieux d'utiliser tous les caractères du jeu ASCII n'intervenant pas dans le programme. Généralement, il s'agit des caractères dont les codes ASCII sont inférieurs à 32, les caractères en langue étrangère et ceux dont les codes ASCII sont supérieurs à 224. Selon les caractères effectivement utilisés dans votre programme, il reste toutefois une centaine de caractères disponibles pour le logo. Avec

un tel bagage, vous pouvez créer aisément un logo avec un étirement de 10*10 caractères et une résolution supérieure à 12 000 points.

Décimal	Hexadécimal	Caractère	Décimal	Hexadécimal	Caractère	Décimal	Hexadécimal	Caractère	Décimal	Hexadécimal	Caractère
0	00		32	20		64	40	@	96	60	`
1	01	☺	33	21	!	65	41	A	97	61	a
2	02	☻	34	22	"	66	42	B	98	62	b
3	03	♥	35	23	#	67	43	C	99	63	c
4	04	♦	36	24	$	68	44	D	100	64	d
5	05	♣	37	25	%	69	45	E	101	65	e
6	06	♠	38	26	&	70	46	F	102	66	f
7	07	•	39	27	'	71	47	G	103	67	g
8	08	◘	40	28	(	72	48	H	104	68	h
9	09	○	41	29	)	73	49	I	105	69	i
10	0A	◙	42	2A	*	74	4A	J	106	6A	j
11	0B	♂	43	2B	+	75	4B	K	107	6B	k
12	0C	♀	44	2C	,	76	4C	L	108	6C	l
13	0D	♪	45	2D	-	77	4D	M	109	6D	m
14	0E	♫	46	2E	.	78	4E	N	110	6E	n
15	0F	☼	47	2F	/	79	4F	O	111	6F	o
16	10	►	48	30	0	80	50	P	112	70	p
17	11	◄	49	31	1	81	51	Q	113	71	q
18	12	↕	50	32	2	82	52	R	114	72	r
19	13	‼	51	33	3	83	53	S	115	73	s
20	14	¶	52	34	4	84	54	T	116	74	t
21	15	§	53	35	5	85	55	U	117	75	u
22	16	▬	54	36	6	86	56	V	118	76	v
23	17	↨	55	37	7	87	57	W	119	77	w
24	18	↑	56	38	8	88	58	X	120	78	x
25	19	↓	57	39	9	89	59	Y	121	79	y
26	1A	→	58	3A	:	90	5A	Z	122	7A	z
27	1B	←	59	3B	;	91	5B	[	123	7B	{
28	1C	∟	60	3C	<	92	5C	\	124	7C	¦
29	1D	↔	61	3D	=	93	5D	]	125	7D	}
30	1E	▲	62	3E	>	94	5E	^	126	7E	~
31	1F	▼	63	3F	?	95	5F	_	127	7F	⌂

Décimal	Hexadécimal	Caractère	Décimal	Hexadécimal	Caractère	Décimal	Hexadécimal	Caractère	Décimal	Hexadécimal	Caractère
128	80	Ç	160	A0	á	192	C0	└	224	E0	α
129	81	ü	161	A1	í	193	C1	┴	225	E1	ß
130	82	é	162	A2	ó	194	C2	┬	226	E2	Γ
131	83	â	163	A3	ú	195	C3	├	227	E3	π
132	84	ä	164	A4	ñ	196	C4	─	228	E4	Σ
133	85	à	165	A5	Ñ	197	C5	┼	229	E5	σ
134	86	å	166	A6	ª	198	C6	╞	230	E6	µ
135	87	ç	167	A7	º	199	C7	╟	231	E7	τ
136	88	ê	168	A8	¿	200	C8	╚	232	E8	Φ
137	89	ë	169	A9	⌐	201	C9	╔	233	E9	Θ
138	8A	è	170	AA	¬	202	CA	╩	234	EA	Ω
139	8B	ï	171	AB	½	203	CB	╦	235	EB	δ
140	8C	î	172	AC	¼	204	CC	╠	236	EC	∞
141	8D	ì	173	AD	¡	205	CD	═	237	ED	φ
142	8E	Ä	174	AE	«	206	CE	╬	238	EE	ε
143	8F	Å	175	AF	»	207	CF	╧	239	EF	∩
144	90	É	176	B0	░	208	D0	╨	240	F0	≡
145	91	æ	177	B1	▒	209	D1	╤	241	F1	±
146	92	Æ	178	B2	▓	210	D2	╥	242	F2	≥
147	93	ô	179	B3	│	211	D3	╙	243	F3	≤
148	94	ö	180	B4	┤	212	D4	╘	244	F4	⌠
149	95	ò	181	B5	╡	213	D5	╒	245	F5	⌡
150	96	û	182	B6	╢	214	D6	╓	246	F6	÷
151	97	ù	183	B7	╖	215	D7	╫	247	F7	≈
152	98	ÿ	184	B8	╕	216	D8	╪	248	F8	°
153	99	Ö	185	B9	╣	217	D9	┘	249	F9	∙
154	9A	Ü	186	BA	║	218	DA	┌	250	FA	·
155	9B	¢	187	BB	╗	219	DB	█	251	FB	√
156	9C	£	188	BC	╝	220	DC	▄	252	FC	η
157	9D	¥	189	BD	╜	221	DD	▌	253	FD	²
158	9E	₧	190	BE	╛	222	DE	▐	254	FE	■
159	9F	ƒ	191	BF	┐	223	DF	▀	255	FF	

Pour vous aider dans votre création de logos personnalisés, nous vous soumettons le programme LOGO écrit en Pascal et C. Les programmes portent les noms LOGOP.PAS et LOGOC.C.

La pièce maîtresse de ces programmes est une procédure ou fonction appelée BuildLogo prenant en main la construction et la sortie du logo. En guise d'informations, elle n'exige que la position du logo sur l'écran, sa hauteur en lignes de points, sa couleur et un tableau chaîne permettant de déterminer l'apparence du logo.

Dans ce tableau, chaque chaîne représente une ligne de points du logo où chaque caractère de la chaîne correspond à un point de la colonne de points appropriée. Si le caractère est un espace, le point correspondant dans le logo reste vide, sinon il est occupé. Cette méthode de codage nécessite une place mémoire importante, mais le logo peut être aisément édité dans le programme source sans qu'il soit nécessaire de développer un éditeur spécial à cet effet. En outre, la largeur du logo peut être obtenue directement à partir du tableau chaîne et la procédure BuildLogo ne doit pas être transmise comme paramètre.

Selon la taille du logo spécifié et la carte vidéo installée (EGA ou VGA), BuildLogo calcule automatiquement le nombre de caractères nécessaires et définit le motif de points des différents caractères du logo à l'aide de la sous-fonction 00h déjà décrite de la fonction 10h de l'interruption vidéo du BIOS. Si le logo ne remplit pas complètement le rectangle par des caractères, alors il sera placé au centre.

Pour la définition du logo, on utilise les caractères accentués situés entre les codes 128 et 166 dans le jeu ASCII. Il reste en tout 38 caractères à votre disposition. Comme BuildLogo reste ouverte à toute manipulation, il est possible d'extraire davantage de caractères du jeu ASCII pour la réalisation du logo.

BuildLogo est aussi une procédure qui sélectionne la largeur de 8 points lorsqu'elle détecte une carte VGA. La commutation s'effectue avec la procédure ou fonction SetCharWidth qui accomplit les différentes étapes de commutation de la largeur comme nous l'avons décrit dans les sections précédentes.

En ce qui concerne la hauteur d'un caractère unique, BuildLogo considère qu'un caractère se compose de 14 lignes de points sur une carte EGA et 16 lignes de points sur une carte VGA. Il provient alors d'un jeu normal aboutissant dans une résolution écran de 25 lignes de texte sur 80 colonnes. Si vous souhaitez utiliser BuildLogo dans un programme activant un jeu plus petit et donc élevant le nombre de lignes écran, vous devez adapter la variable CharHauteur dans les deux programmes LOGO. Avec un logo inchangé, il faudra naturellement redéfinir un nombre de caractères plus important parce que le nombre de lignes de points disponibles par caractère est inférieur.

Si vous n'avez plus besoin du logo, effacez-le tout simplement avec la procédure ResetLogo. Le programme réinstalle l'ancien jeu de caractères et réactive l'affichage 9 points en présence d'une carte VGA.

Pour illustrer le fonctionnement de BuildLogo, nous avons défini un petit logo dans les deux programmes. Il est affiché dans la marge inférieure de l'écran. Immédiatement au-dessus, vous apercevez la liste des caractères du jeu ASCII. Ils apparaissent en couleur pour les mettre en évidence mais peuvent être redéfinis avec BuildLogo. Les caractères redéfinis se distinguent ainsi nettement et constituent une petite partie de la mosaïque qui représente en fait le logo.

Le programme Pascal LOGOP.PAS ne comporte pratiquement aucune routine en Assembleur contrairement au programme C LOGOC.C qui soutient le module Assembleur LOGOCA.ASM. Il s'avère nécessaire parce que les fonctions BIOS appelées pour définir les caractères attendent des informations dans le registre BP sans que ce registre puisse être appelé à l'aide de la fonction d'interruption normale int86(). La routine defchar du module Assembleur se charge donc de définir les caractères.

Listing : LOGOP.PAS

```
{*********************************************************************
*                          L O G O P . P A S                         *
**--------------------------------------------------------------------**
*  Fonction      : Montre comment définir des jeux de caractères      *
*                  personnalisés avec une carte VGA ou EGA en donnant *
*                  comme exemple une routine représentant un logo en  *
*                  mode texte.                                        *
**--------------------------------------------------------------------**
*  Auteur        : MICHAEL TISCHER                                    *
*  Développé le  : 05.08.1990                                         *
*  Dernière MAJ  : 14.01.1991                                         *
*********************************************************************}

Program Logop;

Uses DOS, CRT;

{-- Constantes ------------------------------------------------------}

Const EGAVGA_SEQUENCER = $3C4;   { Port adresses/données du séquenceur }
      EGAVGA_MONCTR    = $3D4;           { Adresse contrôleur d'écran }
      EGAVGA_GRAPHCTR  = $3CE;{Port adr./donnée du contrôleur graphique}
      EV_STATC         = $3DA;     { Registre d'état couleur EGA/VGA }
      EV_STATM         = $3BA;       { Registre d'état mono EGA/VGA }
      EV_ATTR          = $3C0;      { Contrôleur d'attribut EGA/VGA }

{-- Déclarations de types -------------------------------------------}

type CARTE = ( EGA, VGA, NINI);

procedure CLI; inline( $FA );                { Inhibe les interruptions }
procedure STI; inline( $FB );               { rétablit les interruptions}

{*********************************************************************
*  SetCharWidth : Fixe la largeur des caractères à 8 ou 9 pixels     *
*                 pour les cartes VGA.                               *
**--------------------------------------------------------------------**
*  Entrée       : LARGEUR = Largeur des caractères (8 ou 9)          *
*********************************************************************}

procedure SetCharWidth( largeur : byte );

var Regs : Registers;          { Registres pour gérer les interruptions }
    x    : byte;                                  { Variable de travail }

begin
  if ( largeur = 8 ) then Regs.BX := $0001      { BH = Alignem. horiz. }
                     else Regs.BX := $0800;{ BL=valeur pour reg du séq }

  x := port[ $3CC ] and not(4+8);     { Résolution horizontale de 720 à }
  if ( largeur = 9 ) then                      { 640 pixels ou vice-versa }
    x := x or 4;
  port[ $3C2 ] := x;

  CLI;                          { Programme le séquenceur en conséquence }
  portw[ EGAVGA_SEQUENCER ] := $0100;
  portw[ EGAVGA_SEQUENCER ] := $01 + Regs.BL shl 8;
  portw[ EGAVGA_SEQUENCER ] := $0300;
  STI;

  Regs.AX := $1000;                                    { Ajuste l'écran }
  Regs.BL := $13;
  intr( $10, Regs );
end;

{*********************************************************************
*  IsEgaVga : Teste la présence d'une carte EGA ou VGA.              *
**--------------------------------------------------------------------**
*  Entrée : néant                                                     *
*  Sortie : EGA, VGA ou NINI                                          *
*********************************************************************}

function IsEgaVga : CARTE;

var Regs : Registers;          { Registres pour gérer les interruptions }

begin
  Regs.AX := $1a00; { La fonction 1Ah n'existe que pour les cartes VGA }
  Intr( $10, Regs );
  if ( Regs.AL = $1a ) then         { La fonction est-elle disponible ? }
    IsEgaVga := VGA
  else
    begin
      Regs.ah := $12;                              { Appelle l'option 10h }
      Regs.bl := $10;                               { de la fonction 12h }
      intr($10, Regs);                       { Déclenche l'interruption vidéo }
      if ( Regs.bl <> $10 ) then IsEgaVga := EGA
                            else IsEgaVga := NINI;
    end;
end;

{*********************************************************************
*  BuildLogo : Dessine un logo composé de différents caractères      *
*              redéfinis qui ne sont pas utilisés en français        *
**--------------------------------------------------------------------**
*  Entrée :  COLONNE    = Colonne où débute le logo (1-80)            *
*            LIGNE      = Ligne où débute le logo (1-25)              *
*            PROFONDEUR = Nombre de lignes de trame du logo           *
*            COULEUR    = Couleur d'affichage du logo                 *
*            BUF        = Pointeur sur un tableau de pointeurs        *
*                         qui référencent le motif du logo            *
*  Info    : - La procédure test montre comment -                     *
*              réaliser le buffer à transmettre .                     *
*            - Le logo est centré dans son bloc de caractères         *
*********************************************************************}

procedure BuildLogo(colonne, ligne, profondeur, couleur: byte; var buf);

type BYTEAR = array[0..10000] of byte;           { Tableau d'octets  }
     BARPTR = ^BYTEAR;                          { pour le buffer du logo }

const MAX_CHAR = 32;          { maximum de caractères redéfinissables }

const UseChars : array[1..MAX_CHAR] of byte =   { Caractères redéfinis }
                 ( 128, 129, 132, 134, 141, 142, 143, 144, 146, 149,
                   152, 153, 154, 155, 156, 157, 158, 159, 160, 161,
                   162, 163, 164, 165, 166, 167, 168, 171, 172, 173,
                   174, 175 );

var Regs      : Registers;    { Registres pour gérer les interruptions }
    cvideo    : CARTE;                          { Type de la carte vidéo }
    chardef   : array[0..15] of byte;  { Motif binaire d'un caractère }
    charhauteur,             { Nombre de lignes de trame par caractère }
    i, j, k, l,                                 { Compteurs d'itérations }
    masque,                          { Masque pour une ligne de trame }
    largeur,                                  { Largeur de la chaîne }
    index,                 { Indice pour parcourir le tableau UseChars }
    dx,                  { Largeur du bloc du logo (colonnes de texte) }
    dy,                { Profondeur du bloc de logo (lignes de texte ) }
    gauche,                                   { limite gauche en pixels }
    droite,                                   { limite droite en pixels }
    haut,                                      { limite sup en pixels }
    bas     : byte;                            { limite inf en pixels }
    bptr    : barptr;                { pour adresser le buffer du logo }

{ Fonction imbriquée IsSet: Détermine si un pixel du logo est dessiné }

function IsSet( ligne, colonne : byte ) : boolean;

begin
  if ( ligne < haut ) or ( ligne > bas ) or          { Pixel en dehors }
     ( colonne < gauche ) or ( colonne > droite ) then  { de la zone ? }
    IsSet := false                                    { Oui, n'est pas dessiné }
  else                                  { Non, regarde dans le buffer du logo }
    IsSet := bptr^[ (ligne-haut)*(largeur+1) +1+(colonne-gauche) ] <> 32;
end;

{-- Procédure principale -------------------------------------------}

begin
  cvideo := IsEgaVga;                          { Détermine la carte vidéo }
  case cvideo of
    NINI :
      begin
        writeln( 'Erreur: pas de carte EGA ou VGA installée !' );
        exit;
      end;

    EGA  :
      charhauteur := 14;        { EGA: 14 lignes de trame par caractère }

    VGA  :
      begin                                                      { VGA }
        SetCharWidth( 8 );          { Un caractère a 8 pixels en largeur }
        charhauteur := 16;           { 16 lignes de trame par caractère }
      end;
  end;

  bptr := @buf;                              { Pointe sur le buffer du logo }
  largeur := bptr^[0];        { Longueur de la chaîne = largeur du logo }
  dx := ( largeur + 7 ) div 8;                    { Nombre de caractères }
  dy := ( profondeur + charhauteur - 1 ) div charhauteur;
  if ( dx*dy > MAX_CHAR ) then
    writeln( 'Erreur: Logo trop grand dans BuildLogo ! ' )
  else
    begin
```

```
      haut   := ( dy*charhauteur-profondeur ) div 2;{ Calcule les lim. }
      bas  := profondeur + haut - 1;
      gauche  := ( dx*8-largeur ) div 2;
      droite := largeur + gauche - 1;

      TextColor( couleur and 15 );          { Fixe la couleur d'affichage }
      TextBackGround( couleur shr 4 );
      index := 1;                  { Premier caractère de la table UseChar }
      for i := 0 to dy-1 do                { Parcourt les lignes de texte }
        begin
          GotoXY( colonne, ligne + i );
          for j := 1 to dx do           { Parcourt les colonnes de texte }
            begin
              write( chr( UseChars[ index ] ) ); { Affiche les caract. }

              {-- Calcule le nouveau motif pour le caractère affiché --}

              for k := 0 to charhauteur-1 do  { Parcourt les de trames }
                begin
                  masque := 0;               { Pour l'instant masque nul }
                  for l := 0 to 7 do{ Chaque caract. 8 pixels de large }
                    begin
                      masque := masque shl 1;{Décale masque vers gauche}
                      if IsSet( i*charhauteur+k, (j-1)*8+l ) then
                        masque := masque or 1;      { Dessine le pixel }
                    end;
                  chardef[ k ] := masque;       { Motif binaire dans le }
                end;                            { buffer des caracteres }
              Regs.AX := $1100;          { Fixe le ,ouveau motif binaire }
              Regs.BH := charhauteur;          { à l'aide de l'option $00 }
              Regs.BL := 0;      { de la fonction $10 de l'interruption }
              Regs.CX := 1;                              { vidéo du BIOS }
              Regs.DX := UseChars[ index ];
              Regs.ES := seg( chardef );
              Regs.BP := ofs( chardef );
              intr( $10, Regs );

              inc( index );             { Caractère suivant dans UseChar }
            end;
        end;
    end;
end;

{************************************************************************
*  ResetLogo : Recharge le jeu de caractères d'origine                  *
**----------------------------------------------------------------------**
*  Entrée : néant                                                       *
************************************************************************}

procedure ResetLogo;

var Regs      : Registers;       { registres pour gérer l'interruption }

begin
  case IsEgaVga of
    EGA  : begin
             Regs.AX := $1101;          { Charge à nouveau le jeu 8 x 14 }
             Regs.BL := 0;
             intr( $10, Regs );
           end;

    VGA  : begin
             SetCharWidth( 9 );{ Représente les caractères sur 9 pixel }
             Regs.AX := $1104;                    { recharge le jeu 8*16 }
             Regs.BL := 0;
             intr( $10, Regs );
           end;
    end;
end;

{************************************************************************
*  Test : exemple de fonctionnement de la procédure BuildLogo           *
**----------------------------------------------------------------------**
*  Entrée : néant                                                       *
************************************************************************}

procedure Test;

const MyLogo : array[1..32] of string[38] =
                ( '                    **                ',
                  '                   ****               ',
                  '                   ****               ',
                  '                    **                ',
                  '                                      ',
                  '                                      ',
                  '                                      ',
                  '                                      ',
                  '                   ****               ',
                  '                   ****               ',
                  '######################################',
                  '######################################',
                  '###                ****            ###',
                  '##                 ****             ##',
                  '#                  ****              #',
                  '                   ****               ',
                  '                   ****               ',
                  '      ########    ****   ########     ',
                  '      ****  **** **** **** ****  ****  ',
                  '      ****     ********     ****      ',
                  '      ****      ******      ****      ',
                  '      ****       ****       ****      ',
                  '      ****       ****       ****      ',
                  '      ****       ****       ****      ',
                  '      ****       ****       ****      ',
                  '      ****       ****       ****      ',
                  '      ****       ****       ****      ',
                  '      ****       ****       ****      ',
                  '      ****       ****       ****      ',
                  '      ****       ****       ****      ',
                  '      ****       ****       ****      ',
                  '      ****       ****       ****      ' );

var ch : char;
    i  : byte;

const NouvDef : set of char = [#128, #129, #132, #134, #141..#144, #146,
                               #149, #152..#168, #171..#173];

begin
  TextBackGround( BLACK );
  ClrScr;
  GotoXY( 1, 1 );
  for i := 0 to 255 do          { Affiche le jeu complet des caractères }
    begin
      if ( chr(i) in NouvDef ) then TextColor( WHITE )
                               else TextColor( YELLOW );

      GotoXY( i mod 13 * 6 + 2, i div 13 + 1 );
      write( i:3 , ':' );
      if ( i <> 13 ) and ( i <> 10 ) and ( i <> 7 ) then
       write( chr( i ) );
    end;

  GotoXY( 23, 22 );
  write( 'LOGOP  -  (c) 1990 by MICHAEL TISCHER' );
  BuildLogo( 61, 21, 32, CYAN shl 4 + WHITE, MyLogo );{ Dessine le logo}
  ch := ReadKey;                              { Attend une frappe de touche }
  ResetLogo;                                              {Efface le logo }
  ClrScr;
  GotoXY( 1, 1 );
end;

{************************************************************************
*--                          PROGRAMME PRINCIPAL                       --*
************************************************************************}

begin
  Test;
end.
```

Listing : LOGOC.C

```
/*****************************************************************
*                          L O G O C . C                         *
*****************************************************************
* Fonction        : Montre comment définir des jeux de caractères *
*                   personnalisés avec une carte EGA ou VGA en    *
*                   donnant comme exemple une routine représentant*
*                   un logo en mode texte                         *
*****************************************************************
* Auteur          : MICHAEL TISCHER                               *
* Développé le    : 06.08.1990                                    *
* Dernière MAJ    : 14.01.1992                                    *
*****************************************************************
* (MICROSOFT C)                                                   *
* Compilation     : CL /AS /Zp logoc.c logoca                      *
*****************************************************************
* (BORLAND TURBO C)                                               *
* Compilation     : créer un projet avec le contenu suivant       *
*                   logoc.c                                       *
*                   logoca.obj                                    *
*****************************************************************
* Appel           : logoc                                         *
*****************************************************************/

#include <dos.h>                                /* Fichier d'en-tête */
#include <stdarg.h>
#include <stdio.h>
#include <string.h>
#include <conio.h>

#ifdef __TURBOC__                               /* Travail avec Turbo C? */
  #define CLI()           disable()
  #define STI()           enable()
  #define outpw( p, w )   outport( p, w )
  #ifndef inp
    #define outp( p, b )  outportb( p, b )
    #define inp( p )      inportb( p )
  #endif
#else                                   /* Non c'est QuickC 2.0 ou MSC */
  #include <conio.h>
  #define MK_FP(seg,ofs)  ((void far *)\
                          (((unsigned long)(seg) << 16) | (ofs)))
  #define CLI()           _disable()
  #define STI()           _enable()
#endif

#define EGA       0                             /* Types de cartes */
#define VGA       1
#define NINI      2
#define MAX_CHAR  32          /* maximum de caractères redéfinissables */

#define EGAVGA_SEQUENCER 0x3C4    /* Port adresses/données séquenceur */
#define EGAVGA_MONCTR    0x3D4          /* Adr. contrôleur d'écran */
#define EGAVGA_GRAPHCTR  0x3CE     /*Port adresses/données ctrl graph */

/*-- Déclarations de types ---------------------------------------*/

typedef unsigned char     BYTE;

                                        /* Routine en assembleur */
extern void       defchar( BYTE ascii, BYTE table, BYTE height,
                  BYTE nombre, void far * buf );

/*****************************************************************
* SetCursor : Positionne le curseur clignotant                   *
*****************************************************************
* Entrée    : COLONNE = nouvelle colonne du curseur (0-79)       *
*             LIGNE   = nouvelle ligne du curseur (0-24)         *
* Sortie    : néant                                              *
*****************************************************************/

void SetCursor( BYTE colonne, BYTE ligne )
{
  union REGS  regs;                     /* Registres d'interruption */

  regs.h.ah = 2;                  /* Numéro de la fonction "SetCursor" */
  regs.h.bh = 0;                                /* Page d'écran 0 */
  regs.h.dh = ligne;                            /* Indique la ligne */
  regs.h.dl = colonne;                          /* et la colonne */
  int86(0x10, &regs, &regs);          /* Appelle l'interruption vidéo */
}

/*****************************************************************
* PrintfAt : Affiche une chaîne en n'importe quel point de l'écran*
*****************************************************************
* Entrées  : COLONNE = Position d'affichage                      *
*            LIGNE                                               *
*            COULEUR  = attribut des caractères                  *
*            STRING = Pointe sur la chaîne                       *
* Sortie   : néant                                               *
* Info     : Cette fonction en doit être appelée qu'après vérification*
*            préalable de la présence d'une carte EGA ou VGA     *
*****************************************************************/

void PrintfAt(BYTE colonne, BYTE ligne, BYTE couleur, char *string,... )
{
  va_list     parameter;        /* Liste de paramètres pour macro _VA */
  char        affichage[255],         /* buffer pour la chaîne formatée*/
              *affptr;
  BYTE far    *vptr;                  /* Pointe sur la mémoire d'écran */

  va_start( parameter, string );          /* Conversion des paramètres */
  vsprintf( affichage, string, parameter );             /* Formatage */

  vptr = (BYTE far *) MK_FP( 0xB800, colonne * 2 + ligne * 160 );

  for( affptr = affichage; *affptr ; )          /* Parcourt la chaîne */
  {
    *vptr++ = *(affptr++);        /* Caractères dans la mémoire d'écran */
    *vptr++ = couleur;                    /* et de même les attributs */
  }
}

/*****************************************************************
*  ClrScr : Efface l'écran                                       *
*****************************************************************
* Entrée : COULEUR  = attribut des caractères                    *
* Sortie : néant                                                 *
*****************************************************************/

void ClrScr( BYTE couleur )
{
  BYTE far  *vptr;                      /* Pointe sur la mémoire d'écran */
  int       count = 2000;               /* Nombre de caractères à effacer*/

  vptr = (BYTE far *) MK_FP( 0xB800, 0 ); /* Ptr sur la mémoire écran */

  for( ; count--; )                     /* Parcourt la mémoire d'écran */
  {
    *vptr++ = ' ';                /* Ecrit le caractère et son attribut */
    *vptr++ = couleur;                      /* dans la mémoire d'écran */
  }
}

/*****************************************************************
*  SetCharWidth : Fixe la largeur des caractères pour cartes VGA *
*                 à 8 ou 9 pixels                                *
*****************************************************************
* Entrée       : Largeur = largeur du caractère (8 ou 9)         *
*****************************************************************/

void SetCharWidth( BYTE largeur )
{
  union REGS      Regs;         /* Registres pour gérer l'interruption */
  unsigned char   x;                          /* Variable de travail */

  Regs.x.bx = ( largeur == 8 ) ? 0x0001 : 0x0800;

  x = inp( 0x3CC ) & (255-12);      /* Passe de la résolution de 720 à */
  if( largeur == 9 )                      /* 640 pixels ou vice-versa */
    x |= 4;
  (void) outp( 0x3C2, x);

  CLI();                        /* Programme le séquenceur en conséquence */
  outpw( EGAVGA_SEQUENCER, 0x0100 );
  outpw( EGAVGA_SEQUENCER, 0x01 + ( Regs.h.bl << 8 ) );
  outpw( EGAVGA_SEQUENCER, 0x0300 );
  STI();

  Regs.x.ax = 0x1000;                   /* Ajuste l'écran horizontalement */
  Regs.h.bl = 0x13;
  int86( 0x10, &Regs, &Regs );
}

/*****************************************************************
*  IsEgaVga : teste la présence d'une carte EGA ou VGA           *
*****************************************************************
* Entrée : néant                                                 *
* Sortie : l'une des constantes EGA_MONO, EGA_COLOR etc.         *
*****************************************************************/

BYTE IsEgaVga( void )
{
  union REGS      Regs;             /* Registres pour l'interruption */
```

```
  Regs.x.ax = 0x1a00;          /* La fonction 1Ah n'existe que pour VGA */
  int86( 0x10, &Regs, &Regs );
  if( Regs.h.al == 0x1a )      /* La fonction est-elle disponible ? */
    return VGA;
  else                                 /* Non, serait-ce une EGA ? */
  {
    Regs.h.ah = 0x12;                        /* Appelle l'option 10h */
    Regs.h.bl = 0x10;                          /* de la fonction 12h */
    int86( 0x10, &Regs, &Regs );               /* Interruption vidéo */
    return ( Regs.h.bl != 0x10 ) ? EGA : NINI;
  }
}

/*****************************************************************
*  BuildLogo : Dessine un logo composé de différents caractères  *
*              qui sont peu utilisés français                    *
*                                                                *
*----------------------------------------------------------------*
*  Entrée    : COLONNE   = Colonne où débute le logo (1-80)      *
*              LIGNE     = Ligne où débute le logo (1-25)        *
*              PROFONDEUR = Nombre de lignes de trame du logo    *
*              COULEUR   = Couleur d'affichage du logo           *
*              BUFP      = pointeur sur une tableau de pointeurs qui *
*                          référencent des chaînes de caractères *
*                          représentant le motif du logo         *
*  Info      : - La fonction test montre comment réaliser le buffer à *
*                transmettre                                     *
*              - Le logo est centré dans son bloc de caractères  *
*****************************************************************/

void BuildLogo( BYTE colonne, BYTE ligne, BYTE profondeur,
                BYTE couleur, char **bufp )
{
  static BYTE     UseChars[MAX_CHAR] =       /* caractères redéfinis */
          {
            128, 129, 132, 134, 141, 142, 143, 144, 146, 149,
            152, 153, 154, 155, 156, 157, 158, 159, 160, 161,
            162, 163, 164, 165, 166, 167, 168, 171, 172, 173,
            174, 175
            };

  BYTE       cvideo;                          /* Type de carte vidéo */
  BYTE       chardef[16],                /* Motif binaire d'un caractère */
             charhauteur,   /* Nombre de lignes de trame par caractère */
             i, j, k, l,                 /* Variables pour les itérations */
             masque,                  /* Masque pour une ligne de trame */
             largeur,                          /* Largeur de la chaîne */
             index,       /* Indice pour parcourir le tableau UseChars */
             dx,        /* Largeur du bloc du Logo (colonnes de texte) */
             dy,       /* Profondeur du bloc du logo (lignes de texte) */
             colonnec,                          /* colonne courante */
         lignec,                              /* ligne courante */
             gauche,                      /* Marge gauche en pixels */
             droite,                      /* Marge droite en pixels */
             haut,                          /* Marge sup en pixels */
             bas;                           /* Marge inf en pixels */

  cvideo = IsEgaVga();                      /* Quelle carte vidéo ? */
  switch( cvideo )
  {
    case NINI :
      printf( "Erreur : Pas de carte EGA ou VGA installée !\n" );
      return;

    case EGA  :
      charhauteur = 14;       /* EGA: 14 lignes de trame par caractère */
      break;

    case VGA  :
      SetCharWidth( 8 );                /* 8 pixels de largeur en VGA */
      charhauteur = 16;           /* 16 lignes de trame par caractère */
      break;
  }

  largeur = strlen( *bufp );                        /* largeur du logo */
  dx = ( largeur + 7 ) / 8;                     /* Nombre de caractères */
  dy = ( profondeur + charhauteur - 1 ) / charhauteur;
  if( dx*dy > MAX_CHAR )
    printf( "Erreur : Logo trop grand dans BuildLogo !\n" );
  else
  {
    haut   = ( dy*charhauteur-profondeur ) / 2;              /* Marges */
    bas    = profondeur + haut - 1;
    gauche = ( dx*8-largeur ) / 2;
    droite = largeur + gauche - 1;

    for( index = 0, i = 0; i < dy; ++ i)
    {                                          /* Parcourt lignes texte */
      for(j = 0; j < dx; ++j, ++index )    /* Parcourt colonnes texte */
      {
        PrintfAt( colonne+j, ligne+i, couleur,  /* Affiche caractères */
                  "%c", UseChars[ index ] );

/*-- Calcule le nouveau motif pour le caractère ----------------------*/

        for( k = 0; k < charhauteur; ++ k )  /* Parcourt lignes trame */
        {
      masque = 0;                              /* Pour l'instant masque nul */
          for( l = 0; l <= 7; ++l )              /* 8 pixels de largeur */
      {
            masque <<= 1;             /* Décale le masque vers la gauche */
            lignec = i * charhauteur + k;
            colonnec = j * 8 + l;

            if( lignec>=haut     && lignec<=bas  &&  /* Pixel au-delà */
                colonnec>=gauche && colonnec<=droite)/* de la marge ? */
              if( *(*(bufp+lignec-haut)+colonnec-gauche) != ' ' )
                masque |= 1;                 /* Dessine un point du logo */
          }
          chardef[ k ] = masque;   /* motif dans le buffer caractères */
        }
      defchar( UseChars[ index ], 0, charhauteur, 1, chardef );
      }
    }
  }
}

/*****************************************************************
*  ResetLogo : recharge le jeu de caractères d'origine           *
*----------------------------------------------------------------*
*  Entrée    : néant                                             *
*****************************************************************/

void ResetLogo( void )
{
  union REGS    Regs;           /* Registres pour gérer l'interruption */

  switch( IsEgaVga() )
  {
    case EGA :
      Regs.x.ax = 0x1101;               /* Charge à nouveau le jeu 8*14 */
      Regs.h.bl = 0;
      int86( 0x10, &Regs, &Regs );
      break;

    case VGA :
      SetCharWidth( 9 );    /* Représente les caractères sur 9 pixels */
      Regs.x.ax = 0x1104;                       /* recharge le jeu 8*16 */
      Regs.h.bl = 0;
      int86( 0x10, &Regs, &Regs );
    }
}

/*****************************************************************
*  Test    : exemple de fonctionnement de la procédure BuildLogo *
*----------------------------------------------------------------*
*  Entrée : néant                                                *
*****************************************************************/

void Test( void )
{
  static char *MyLogo[32] =
              { "                  **                  ",
                "                 ****                 ",
                "                 ****                 ",
                "                  **                  ",
                "                                      ",
                "                                      ",
                "                                      ",
                "                                      ",
                "                 ****                 ",
                "                 ****                 ",
                "****************************************",
                "****************************************",
                "***               ****               ***",
                "**                ****                **",
                "*                 ****                 *",
                "                  ****                ",
                "                  ****                ",
                "      ********    ****   ********      ",
                "      ****  **** **** **** ****  ****  ",
                "      ****    ********     ****  ****  ",
                "      ****     ******      ****  ****  ",
                "      ****      ****       ****  ****  ",
                "      ****      ****       ****  ****  ",
                "      ****      ****       ****  ****  ",
                "      ****      ****       ****  ****  ",
                "      ****      ****       ****  ****  ",
                "      ****      ****       ****  ****  ",
                "      ****      ****       ****  ****  ",
                "      ****      ****       ****  ****  ",
                "      ****      ****       ****  ****  ",
                "      ****      ****       ****  **** " };

  int       i, j;
```

```
BYTE        couleur;

static BYTE NouvDef[MAX_CHAR] =             /* Caractères redéfinis */
        {
        128, 129, 132, 134, 141, 142, 143, 144, 146, 149,
        152, 153, 154, 155, 156, 157, 158, 159, 160, 161,
        162, 163, 164, 165, 166, 167, 168, 171, 172, 173,
        174, 175
        };

ClrScr( 0 );
for( i = 0; i < 256; ++i ) /* Affiche le jeu complet des caractères */
{
  for( j = 0; j < MAX_CHAR; ++ j )          /* nouvelle définition ? */
    if( NouvDef[ j ] == i )     /* Le caractère en fait-il partie ? */
      break;                              /* Oui, sortir de la boucle */
  couleur = ( j < MAX_CHAR ) ? 15 : 14;

  PrintfAt( (i % 13) * 6 + 1, i / 13, couleur, "%3d:%c", i, i );
}
```

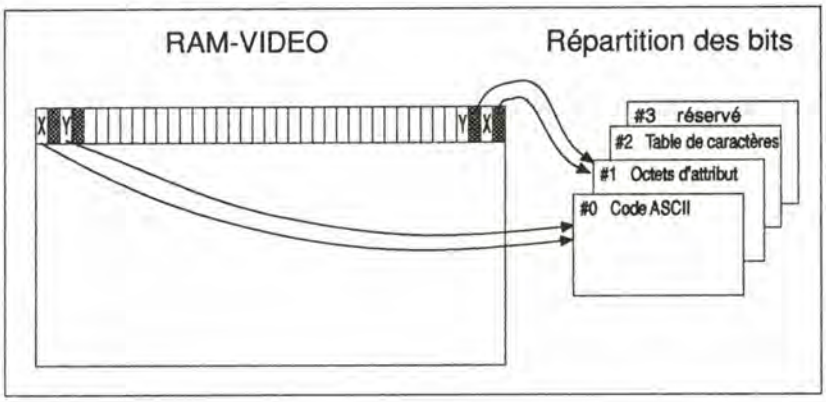
```
PrintfAt( 18, 22, 14, "LOGOC  -  (c) 1990, 92 by MICHAEL TISCHER" );
BuildLogo( 60, 21, 30, 0x3F, MyLogo );          /* dessine le logo */
getch();
ResetLogo();                                    /* Efface le logo */
ClrScr( 15 );
SetCursor( 0, 0 );
}

/*****************************************************************
*                          PROGRAMME PRINCIPAL                   *
*****************************************************************/

void main( void )
{
  Test();
}
```

Listing : LOGOCA.ASM

```
;*****************************************************************;
;*                        L O G O C A . A S M                    *;
;*---------------------------------------------------------------*;
;*   Fonction       : définit un jeu de caractères personnalisés *;
;*                    pour les cartes EGA et VGA                 *;
;*---------------------------------------------------------------*;
;*   Auteur         : MICHAEL TISCHER                            *;
;*   Développé le   : 7.08.1990                                  *;
;*   Dernière MAJ   : 14.02.1992                                 *;
;*---------------------------------------------------------------*;
;*   Assemblage     : TASM -mx logoca     ou MASM -mx logoca;    *;
;*****************************************************************;

        DOSSEG                          ; Ordre des segments habituel
        .MODEL SMALL, C                 ; Prépare l'inclusion du code
                                        ; dans un programme C en modèle SMALL

        PUBLIC defchar

;-- Code ---------------------------------------------------------

        .CODE

;-- DEFCHAR : fixe les motifs des caractères pour cartes EGA et VGA
;--
;-- Déclaration en C : void defchar( BYTE ascii, BYTE table, BYTE height
;--                                  BYTE nombre, void far * buf );
;-- Entrées          : ASCII  = Numéro du 1er caractère à définir
;--                    TABL   = Numéro de la table des caractères
;--                    HEIGHT = Hauteur des caractères (lignes de trame)
;--                    NOMBRE = Nombre de caractères
;--                    BUF    = Pointeur Far sur le buffer
;-- Sortie           : néant

defchar   proc ascii:byte,  tabl:byte, height:byte, \
               nombre:byte, buf:dword

        mov  ax,1100h           ; Appelle l'option 00h de la fonction 11h
        mov  bh,height                  ; Charge les paramètres appropriés
        mov  bl,tabl                            ; dans les registres
        mov  dl,ascii
        xor  dh,dh
        mov  cl,nombre
        mov  ch,dh

        push bp                 ; Mémorise BP pour l'adressage de la pile
        les  bp,buf
        int  10h                        ; Appelle l'interruption vidéo BIOS
        pop  bp                                 ; Récupère l'ancien BP

        ret                                     ; Retourne à l'appelant

defchar   endp

;-- Fin ----------------------------------------------------------
        end
```

Structure et état des diverses tables de caractères

Tant que vous vous contentez d'adresser les tables de caractères à l'aide du BIOS, vous n'avez aucun souci à faire quant à la structure et l'état de ces tables. Mais si vous décidez d'avoir directement recours aux diverses tables de caractères, alors la situation change du tout au tout. Il convient dans ce cas de connaître l'état et la structure des tables pour effectuer un accès direct à toutes les définitions de caractères ou quelques-unes seulement. Beaucoup d'opérations impossibles à réaliser avec le BIOS ou du moins avec grand-peine deviennent alors accessibles. Où se situent donc ces tables et comment sont-elles construites ?

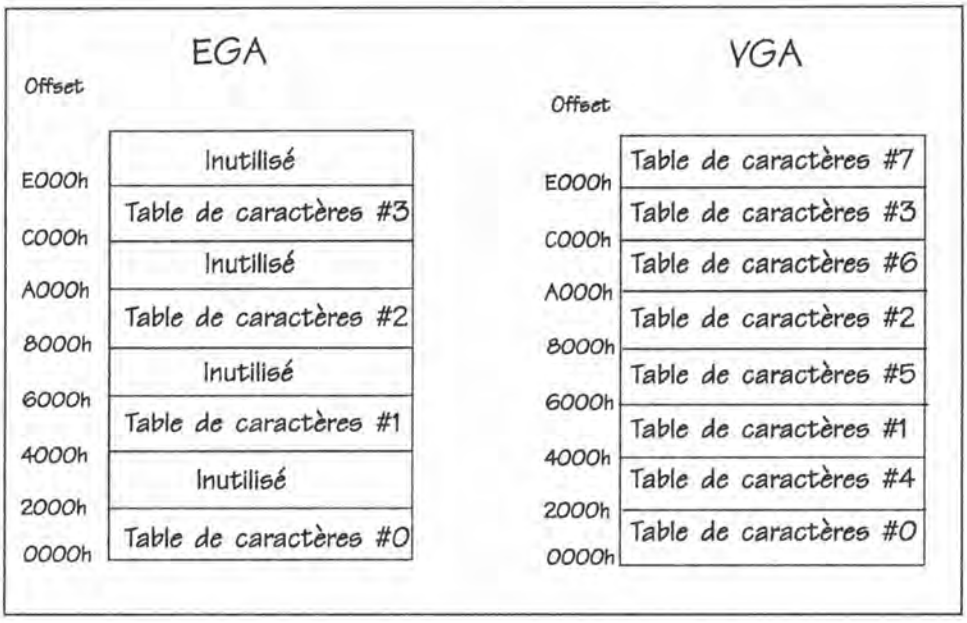


Utilisation des plans de bits en mode texte

Pour répondre à ces questions, nous devons d'abord ouvrir une parenthèse dans l'organisation de la RAM vidéo avec des cartes EGA et VGA. Dans le paragraphe 4.8.5, vous apprendrez que les cartes EGA et VGA subdivisent leur RAM vidéo en quatre zones identiques désignées par plans de bits. Leurs tâches varient selon qu'il s'agit du mode graphique ou du mode texte. Toujours est-il que les quatre plans de bits occupent 64 Ko dans une carte vidéo entièrement comblée c'est-à-dire équipée de 256 Ko.

En mode texte, les deux premiers plans de bits (plan de bits #0 et plan de bits #1) sont utilisés pour recevoir les codes de caractères et octets d'attributs complètement à l'insu du programmeur. Les codes ASCII des caractères sont stockés dans le plan de bits #0 parce que tous les accès à la RAM vidéo à partir de B800h, faisant référence à une adresse d'offset paire, sont déviés automatiquement vers ce plan de bits. En revanche, tous les accès à des adresses impaires où sont stockés les octets d'attributs aboutissent dans le plan de bits #1.

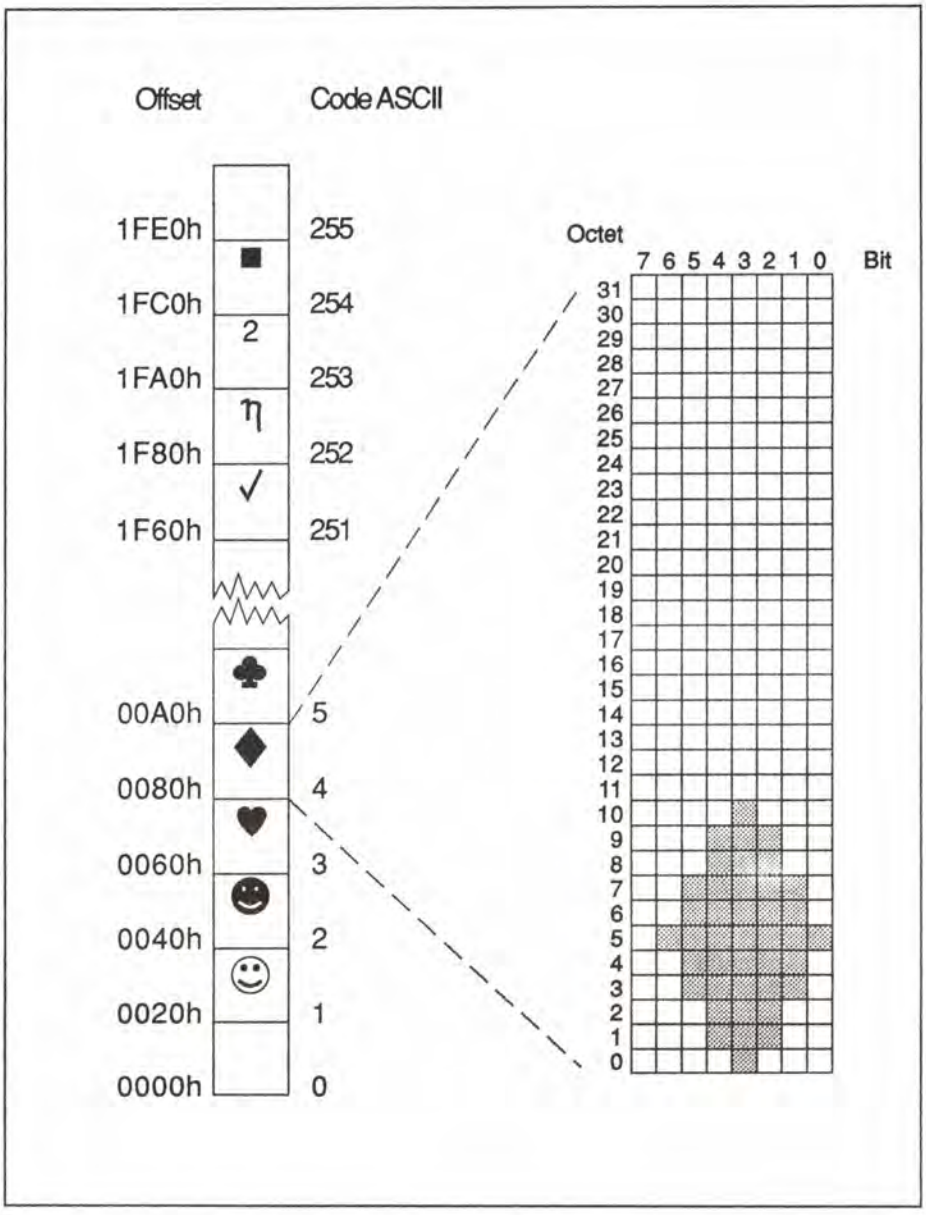
Mais les autres plans de bits ne restent pas pour autant inutilisés en mode texte. En fait, les tables de caractères dans les cartes EGA et VGA sont stockées dans le troisième plan de bits qui sert exclusivement à cet usage. Comme ce plan de bits occupe 64 Ko d'une carte vidéo équipée de 256 Ko, chaque table de caractères n'utilise que 8 Ko si bien que le plan de bits #2 peut recevoir simultanément 8 tables différentes. Comme le montre la figure suivante, seules les cartes VGA en font usage. Les cartes EGA se contentent seulement de quatre tables de caractères si bien qu'il reste en tout 32 Ko inutilisés dans ce plan de bits.



Etat des tables de caractères à l'intérieur du plan de bits #2

Chaque table a besoin de 8 Ko parce qu'elle doit réserver 32 octets pour chacun des 256 caractères. Vous vous rappelez que 32 correspond à la hauteur maximale des caractères dans les cartes EGA et VGA. Cela explique pourquoi on prévoit toujours l'éventualité maximale dans les tables de caractères. Si une table doit accepter un jeu dont la hauteur est inférieure à 32 lignes de points, un nombre correspondant d'octets reste tout simplement inutilisé à la fin de chaque entrée.

Les octets restants sont codés selon la même méthode que celle décrite dans le contexte des fonctions BIOS chargées de la définition des caractères. Chaque octet reçoit le motif de points d'une ligne de points du caractère où les différents bits réflètent l'état des divers points de cette ligne. Ce mode d'organisation permet aisément de calculer l'adresse de départ d'un caractère ou d'une ligne de points lorsque vous connaissez l'adresse de départ de la table concernée. Multipliez les codes ASCII du caractère par 32, ajoutez-y le numéro de la ligne souhaitée puis l'adresse de départ de la table. Vous obtenez ainsi l'adresse d'offset de la ligne de points souhaitée.



Structure d'une table de caractères

Si l'accès à une table de caractères paraît si facile à effectuer c'est parce que le plan de bits #2 n'a pas encore été adressé. Pour accéder à un plan de bits avec des cartes EGA et VGA, il faut d'abord l'intégrer dans la mémoire pour pouvoir ensuite le réclamer au moyen de l'adresse de segment A000h. Le BIOS n'offre malheureusement aucune

fonction assurant cette tâche et il ne reste alors plus qu'à se tourner vers la programmation directe des registres EGA et VGA.

Concrètement parlant, il s'agit des registres du contrôleur Sequencer et du contrôleur graphique dont la programmation libère l'accès au plan de bits #2. Les registres 2 et 4 ont la responsabilité d'organiser l'accès aux divers plans de bits pour le contrôleur Sequencer. Ces registres s'appellent registre Map Mask et registre Memory Mode. Le premier détermine le plan de bits accessible actuellement. Un bit correspond à chaque fois à un plan de bits. Dans notre cas, il convient uniquement de régler le bit du plan de bits #2, les autres sont à effacer. Il ne faut pas non plus que l'accès au plan de bits #2 fasse éprouver de la pitié pour les autres plans de bits.

Pour empêcher l'accès simultané à tous les plans de bits et n'adresser en fait que les plans de bits marqués dans le registre Map Mask, il faut charger le registre Memory Mode avec la valeur 7. En mode texte, il contient normalement la valeur 3 qui agit en liaison avec le registre Map Mask. Ainsi, lors d'un accès à la RAM vidéo en B800h, tous les accès aux adresses de mémoire paires (codes ASCII) sont déviés vers le plan de bits #0, tous les accès aux adresses de mémoire impaires (codes d'attributs) vers le plan de bits #1. La division des adresses de mémoire sera bientôt interdite.

Tout comme pour la commutation depuis l'affichage 9 points vers l'affichage 8 points, il faut effectuer un reset lors de l'accès aux registres du Sequencer via le registre Reset. Avant d'accéder aux registres Map Mask et Memory Mode, il faut d'abord inscrire la valeur 1 puis la valeur 3 dans le registre Reset.

L'accès aux registres du contrôleur graphique ne nécessite aucun reset. Ici, les registres 4, 5 et 6 à manipuler sont prêts pour adresser directement le plan de bits #2. Il s'agit des registres Read Map Select, Graphics Mode et Miscellaneous.

Dans le registre Read Map Select, il faut inscrire tout d'abord le numéro du plan de bits à adresser, soit la valeur 2. Il faut vérifier ici que le plan de bits #2 reste disponible pour les accès en lecture dans la RAM vidéo. Dans le registre Graphics Mode, il faut spécifier la valeur 0. Il s'agit uniquement d'effacer le bit 4 puisque les autres bits n'interviennent qu'en mode graphique. La suppression de ce bit confirme que la division entre adresses de mémoire paires et impaires ne doit pas avoir lieu.

Cela se réalise à l'aide du registre Miscellaneous où les bits 2 et 3 sont utiles en plus du bit 1 qui est prévu à cet effet. Il spécifient l'endroit occupé par la RAM vidéo, les plans de bits doivent donc être intégrés dans la zone d'adresse du processeur. En mode texte, la RAM vidéo apparaît à l'adresse de segment B800h et occupe 32 Ko. Le plan de bits #2 est accessible à l'adresse de segment A000h et peut être adressé sur l'ensemble des 64 Ko du plan de bits.

Les paramètres décrits annulant les prédéfinitions effectuées pour le mode texte, un accès au plan de bits #2 devient possible à la fin des opérations menées sur les registres, mais aucun accès à la RAM vidéo normalement en B800h n'est autorisé. Au cours de l'accès, les sorties de caractères s'accumulent sur le plan de bits #2 et ne retombent pas

dans la RAM vidéo. L'image vidéo reste par ailleurs inchangée sur le moniteur parce que la carte vidéo peut accéder directement à la RAM vidéo sur le plan interne.

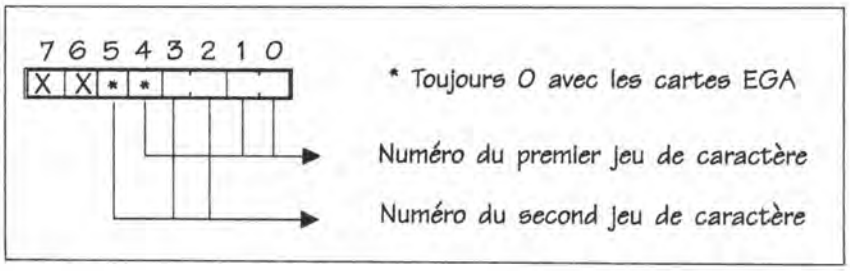
Pour que votre programme ou le BIOS puisse à nouveau accéder à la RAM vidéo, les modifications effectuées dans les divers registres doivent être annulées au profit des définitions antérieures. Le tableau suivant montre les valeurs à entrer pour l'accès au plan de bits #2 ou à la RAM vidéo en B800h.

Registres	Plan de bits	RAM vidéo
Registre Map Mask (Contrôleur Sequencer)	04h	03h
Registre Memory Mode (Contrôleur Sequencer)	07h	03h
Registre Read Map Select (Contrôleur graphique)	02h	00h
Registre Graphics Mode (Contrôleur graphique)	00h	10h
Registre Miscellaneous (Contrôleur graphique)	04h	0Eh

Les programmes d'exemple MIKADO décrits ci-après contiennent deux routines permettant de commuter entre l'accès à la RAM vidéo et le plan de bits #2. Vous aurez également de plus amples informations sur les registres intervenant dans cette commutation à la fin de ce chapitre lors de la description des registres EGA et VGA.

Commutation entre des jeux multiples ou affichage simultané de 512 caractères variés

Nous avons souligné plus haut que le plan de bits #2 peut recevoir quatre jeux de caractères avec des cartes EGA et huit jeux avec des cartes VGA. Par ailleurs, lors de la manipulation des jeux de caractères, le BIOS ne reste pas uniquement fixé sur le premier jeu. Il reste maintenant à savoir comment s'effectue la commutation entre les divers jeux et l'apparition d'un jeu sur l'écran. La réponse est fournie par le registre Character Map Select qui fait partie intégrante du contrôleur Sequencer et peut être adressé au moyen du numéro de registre 4. Il est exclusivement responsable de la sélection de la table de caractères en cours comme le montre la figure suivante.



D'après l'affectation du registre Character Map Select, on constate immédiatement qu'on a affaire à deux jeux de caractères. Essayons d'éclairer la situation.

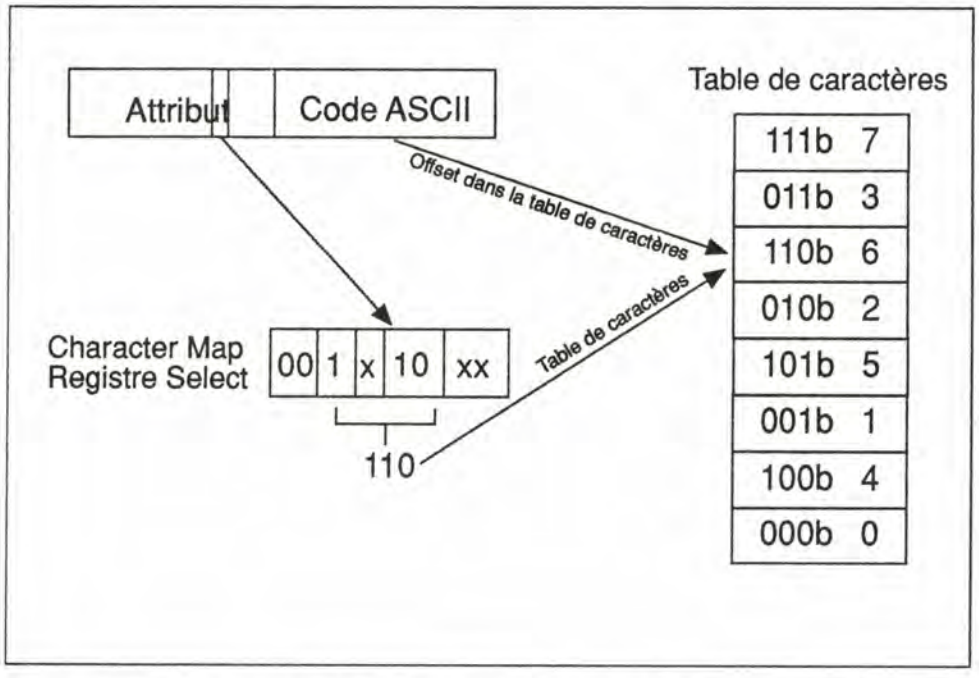
L'organisation des différents bits indiquant les numéros des deux jeux semble également déroutante. Au lieu d'utiliser notamment trois bits consécutifs pour coder un numéro, les deux numéros se composent respectivement d'un groupe de 2 bits et un bit supplémentaire séparé des deux et représentant le bit de poids fort du troisième groupe. Cette organisation inhabituelle résulte de l'histoire même des cartes EGA et VGA. En fait, les cartes EGA nécessitent seulement deux bits pour coder un numéro de jeu de caractères. Sachant que quatre jeux de caractères sont disponibles dans ce cas, il ne faut donc retenir que les numéros entre 0 et 3.

Il en va tout autrement avec les cartes VGA qui permettent l'accès à huit jeux différents. Pour coder les numéros des jeux 0 à 7, on utilise ici trois bits. Afin de garantir la plus grande compatibilité de registres entre les cartes EGA et VGA, on a tout simplement conservé l'affectation prédéfinie EGA et ajouté deux bits distincts aux trois qui existaient déjà. Cela ne pose aucun problème puisque la position de bits correspondante n'a pas encore servi sur une carte EGA.

Pourquoi ce registre contient donc deux numéros de registre ? La figure précédente mentionne l'existence d'un "premier" et d'un "second" jeu de caractères. Cela n'évoque nullement une hiérarchie entre les deux jeux. Contrairement à toutes les cartes qui leur ont précédé, les cartes EGA et VGA sont en mesure d'afficher simultanément 512 caractères (et non 256) sur l'écran. Bien que cette faculté soit peu connue, elle est très intéressante puisqu'elle pose immédiatement le problème de la sélection entre les deux jeux. Le code ASCII d'un caractère dans la RAM vidéo ne reproduit en fait qu'un nombre entre 0 et 256 si bien qu'il est inutile de rechercher ici le critère de décision.

A première vue, il semble qu'aucune place n'est accordée à ce type d'information dans l'octet d'attribut d'un caractère. En fait, les deux Nibbles de cet octet sont utilisés pour sélectionner une couleur de premier plan et de fond pour le caractère concerné parmi les 16 disponibles. Et voici la clé de l'énigme consistant à utiliser 512 caractères différents. C'est le bit de poids fort de la couleur de premier plan - soit le bit 3 dans l'octet d'attribut - qui permet de sélectionner le premier ou le second jeu de caractères.

Si ce bit contient la valeur 0 (la couleur de premier plan sélectionnée est alors inférieure à 8), la carte vidéo extrait l'image représentant le caractère à partir du premier jeu. Si le bit #3 est réglé, le second jeu est mis à contribution. Dans ce cas, la couleur de premier plan utilisée est supérieure ou égale à huit.



Sélection des jeux de caractères à travers le registre Character Map Select compte tenu de l'octet d'attribut d'un caractère

La séparation entre le premier et le second jeu de caractères s'effectue habituellement de base même si elle reste généralement invisible aux yeux de l'utilisateur. Cela s'explique tout simplement par le fait que le numéro indiqué pour le premier et le second jeu de caractères dans le registre Character Map Select est identique. Mais au cas où les deux numéros ne sont pas les mêmes, la différence apparaît nettement sur l'écran.

Il convient donc d'accéder à ce registre lorsqu'il faut afficher deux jeux différents sur l'écran ou sélectionner un autre jeu. On peut ainsi commuter rapidement d'un jeu à l'autre et obtenir des effets optiques intéressants. Mais il faut éviter de programmer directement le registre Character Map Select puisque le BIOS fournit spécialement à cet effet une sous-fonction de la fonction 11h dans le BIOS vidéo.

Il s'agit de la sous-fonction 03h. En dehors des deux numéros de fonctions dans les registres AH et AL, elle attend un autre paramètre dans le registre BL. Il représente la valeur à charger dans le registre Character Map Select et spécifie ainsi les jeux affichés. La section suivante illustre l'utilisation de cette fonction dans le cadre d'un exemple où une fenêtre graphique est configurée dans l'écran texte à l'aide du "second jeu de caractères".

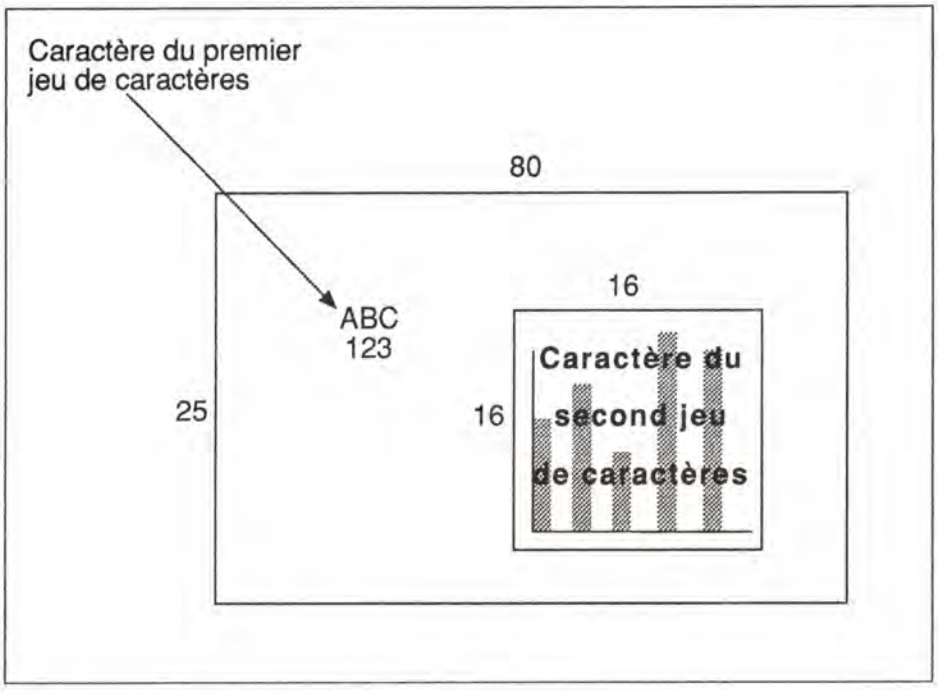
L'utilisation de deux jeux de caractères crée toutefois une situation ennuyeuse où les caractères du second jeu restent surlignés en permanence par rapport à ceux du premier jeu. Le fait est que le bit 3 est mis comme couleur de premier plan. Pour remédier à ce

problème, il suffit d'harmoniser le contenu des registres de palette portant le numéro 8 avec celui portant un numéro entre 0 et 7. Une telle opération ne pose aucun problème puisque le BIOS n'est pas dépourvu de fonctions appropriées à cette situation. Le nombre des couleurs disponibles pour le premier plan se limite pourtant à 8 car il est rare qu'un programme utilise réellement huit couleurs différentes et de toute manière vous pouvez sélectionner à votre convenance les couleurs disponibles en programmant les registres de palette.

Une fenêtre graphique en mode texte

Si vous n'avez pas besoin d'afficher 512 caractères sur l'écran parce que vous utilisez des symboles mathématiques ou autres, alors le registre Character Map Select vous laisse une alternative intéressante : une fenêtre graphique en mode texte. Elle permet d'afficher de petits graphiques ou dessins en mode texte sans être obligé d'activer le mode graphique dont la programmation est plus complexe.

La technique qui se cache derrière une telle fenêtre graphique ressemble à celle de la création d'un logo. Ici aussi, il s'agit de regrouper des caractères - provenant cette fois du second jeu et non du premier - en un bloc et former une sorte de mosaïque. Dans cette procédure, chaque caractère du second jeu est représenté exactement une fois dans la fenêtre graphique. Cette opération étant importante, il faut que la fenêtre graphique soit assez grande pour recevoir 256 caractères ou moins. Un tel bloc peut par exemple représenté par un carré de 16 fois 16, mais n'importe quelle figure est évidemment autorisée.



Structure d'une fenêtre graphique en mode texte

Pour la sortie graphique dans la fenêtre, le second jeu est considéré comme un grand tableau constitués de différents points représentés dans la fenêtre graphique sur les différentes coordonnées de points. Si un point doit être mis ou effacé dans la fenêtre graphique, le caractère dans la zone duquel se trouve le point est d'abord calculé. On obtient ainsi la position du point par rapport au coin supérieur gauche du caractère et le bit correspondant à l'intérieur du motif de points du caractère dans la table de caractères qu'il soit mis ou effacé. Voici un exemple concret :

Dans les cartes VGA, les caractères se composent de 16 lignes de points offrant de la place à huit points différents. Si on définit le coin inférieur gauche comme point d'origine de la fenêtre graphique, alors tous les points de coordonnée X sont inférieurs à 8 et les points de coordonnée Y inférieurs à 16 dans la zone d'influence du caractère texte situé dans le coin inférieur gauche de la fenêtre graphique. D'après la figure précédente, il s'agit toujours du caractère avec le code ASCII 0.

Le numéro du caractère à manipuler est ainsi spécifié à l'intérieur du jeu. L'étape suivante consiste à calculer les lignes de points à modifier dans le caractère. Il convient ici de faire attention à la structure des caractères dans la table. Les différentes lignes de points des caractères sont stockées de haut en bas. Un point avec la coordonnée Y 15 se trouve donc dans la ligne de points supérieure avec le code ASCII 0 alors que des points de coordonnée Y 0 occupent la ligne de points inférieure. Cette règle permet de calculer rapidement les lignes de points à adresser.

Il ne reste plus qu'à déterminer la position des bits du point graphique à réclamer dans la ligne de points concernée. En partant du bord gauche du caractère, les points occupent les positions 7, 6, 5 et ainsi de suite jusuqu'à 0. Le point demandé peut être mis ou effacé grâce aux trois informations concernant le numéro de caractère, la ligne de points et le numéro de bit. Il suffit tout simplement de placer ces informations sur la structure de la table de caractères dans la RAM vidéo.

Il paraît quelque peu contradictoire que la définition ou la suppression d'un point nécessite de redéfinir complètement le caractère, ce qui n'est pas bienvenu lors de l'utilisation des fonctions BIOS pour la définition de caractères. Ces fonctions ne sont pas en outre rapides et on s'en aperçoit très vite lorsqu'on édite successivement plusieurs points (pour tracer une ligne par exemple). Cette raison conduit les deux programmes à accéder directement au plan de bits #2 pour obtenir une vitesse tout à fait passable lors de la manipulation de la fenêtre graphique.

Avec les cartes VGA, on se confronte à nouveau au problème déjà évoqué dans le cadre du programme LOGO : le neuvième point. En mode graphique et avec des cartes VGA, le huitième point d'un graphique n'étant pas automatiquement reproduit sur le neuvième ou restant tout simplement vide, l'affichage des caractères doit se limiter à huit points comme c'était le cas avec LOGO. Les étapes nécessaires à cette opération ont été déjà décrites dans les paragraphes précédents.

Les deux programmes présentés ici s'appellent MIKADO et sont fournis en Pascal (MIKADOP.PAS) et en C (MIKADOC.C). La pièce maîtresse de ces programmes est représentée par une routine InitGraphArea permettant de configurer une fenêtre graphique. Il existe une autre routine servant à placer ou effacer séparément les différents points. Elle porte le nom SetPixel et sert de base pour la procédure ou fonction Line qui dessine une ligne dans la fenêtre graphique selon le fameux algorithme Bresenham.

La procédure InitGraphArea transmet sur l'écran l'état de la fenêtre graphique ainsi que ses dimensions et la couleur des caractères. Il est possible de sélectionner également le numéro du jeu à utiliser pour construire la fenêtre graphique. Grâce aux indications obtenues sur les dimensions de la fenêtre et la résolution des caractères, on peut calculer les coordonnées maximales X et Y et entrer xmax et ymax dans les variables globales.

L'accès au plan de bits #2 s'effectue selon le principe déjà décrit à l'aide d'une routine appelée GetFontAccess. En guise de complément, la procédure ou fonction Release-FontAccess ouvre à nouveau la voie à la RAM vidéo B800h. Dans le programme MIKADO, nous n'appelons pas souvent les deux routines parce que leur exécution s'accompagne toujours momentanément de l'apparition de neige sur l'écran.

La procédure SelectMaps est également importante dans le programme MIKADO. Elle a pour tâche d'intégrer les jeux souhaités dans le registre Character Map Select. Elle se sert à cet effet de la fonction BIOS décrite antérieurement. Cela évite d'accéder directement à l'électronique de la carte vidéo ce qui répond parfaitement au caractère contractuel du programme eu égard à l'affectation variée des registres des cartes VGA des divers constructeurs.

Dans les deux programmes MIKADO, deux procédures servent à programmer les registres de palette. A l'aide du BIOS, ces procédures redéfinissent à chaque fois un élément de la palette de couleurs en 16 parties ou retournent le contenu. Elles portent les noms SetPalCol, SetPalAry, GetPalCol et GetPalAry. Elles ont été définies à dessein pour que les diverses couleurs deviennent disponibles pour le texte et la fenêtre graphique.

Revenons à la couleur. Elle permet de représenter telle ou telle chose dans telle ou telle fenêtre graphique parce chaque caractère et non chaque point est doté d'une couleur individuelle. Dans une carte EGA, 14*8 points interviennent dans cette opération et dans une carte VGA, 16*8 points. Il faut donc essayer d'harmoniser les couleurs de premier plan et de fond affectées aux points de la fenêtre graphique.

Nous avons intentionnellement omis de respecter ce conseil dans les programmes MIKADO où les différents caractères sont dotés d'un code de couleur continu. Les programmes portent d'ailleurs bien leurs noms puisque des lignes sont tracées dans la fenêtre graphique, elles croisent généralement toutes sortes de caractères et créent ainsi des reflets de couleurs changeants. Ces lignes ressemblent à des mikados ou baguettes en bois utilisées dans le jeu japonais de même nom.

Les deux programmes MIKADO mettent en évidence l'ordre à suivre pour l'appel des diverses routines : vous devez d'abord vous assurer de l'existence d'une carte EGA ou VGA avec la fonction IsEgaVga. La fenêtre graphique ne peut être ouverte en appelant InitGraphArea qu'à la suite de ce test. Avant d'avoir accès à cette fenêtre, il faut en outre appeler GetFontAccess. Ce n'est qu'à cette condition que vous pouvez tracer les points ou lignes à l'aide de SetPixel ou Line.

Si vous devez entre-temps sortir un texte sur l'écran, vous devez appeler d'abord ReleaseFontAccess puis GetFontAccess si vous souhaitez continuer la sortie graphique. L'appel de la procédure ClearGraphArea permet d'effacer la fenêtre graphique. Cette procédure libère automatiquement l'accès à la RAM vidéo et réinstalle un jeu de caractères standard (portant le numéro 0).

Le fait que les caractères du jeu ASCII restent malgré tout disponibles dans la fenêtre graphique à travers le premier jeu, le programme MIKADO vous le démontre en remplissant l'écran autour de la fenêtre graphique par les caractères correspondants.

Tout ce dont vous avez besoin pour utiliser une fenêtre graphique en mode texte vous est fourni par les deux programmes MIKADO. Après avoir consulté les listings, il ne vous sera certainement pas difficile d'utiliser les routines pour vos besoins personnels. Il est certes fascinant d'apercevoir brusquement une fenêtre dotée d'un graphique au beau milieu d'un écran texte. Il ne vous reste plus qu'à donner libre cours à votre imagination.

Listing : MIKADOP.PAS

```
{*****************************************************************
*                          M I K A D O P                         *
*****************************************************************
* Fonction : Montre comment mettre en service le mode 512 caractères *
*            des cartes EGA et VGA. La routine de démonstration      *
*            installe une fenêtre graphique en mode texte.           *
*****************************************************************
* Auteur       : MICHAEL TISCHER                                    *
* Développé le :  2.04.1990                                         *
* Dernière MAJ : 14.01.1991                                         *
*****************************************************************}

program MikadoP;

uses DOS, CRT;      { Inclut les unités DOS et CRT }

{-- Constantes ---------------------------------------------------}

const EGAVGA_SEQUENCER = $3C4; { Port d'adresses/données du séquenceur }
      EGAVGA_MONCTR    = $3D4;                 { Contrôleur d'écran }
      EGAVGA_GRAPHCTR  = $3CE;(Port d'adr./données du contrôleur graph.)
      CHAR_LARGEUR     = 8;
      CHAR_BYTES       = 32;
      MIKADOS          = 5; { Nombre de mikados visibles en même temps }

{-- Déclarations de types ----------------------------------------}

type VEL      = record         { Décrit un couple caractère - attribut }
                  case boolean of                  { en mémoire d'écran }
                    true  : ( Caractere, Attribut : byte );
                    false : ( Contenu             : word );
                end;
     VPTR     = ^VEL;              { Pointe sur un caractère/attribut }
     VELARRAY = array [1..25,1..80] of VEL; {Reproduit la mémoire écran}
     VELARPTR = ^VELARRAY;           { Pointe sur la mémoire d'écran }

     FONT = array[0..255,0..CHAR_BYTES-1] of byte;{Reproduit une police}
     FPTR = ^font;                          { Pointe sur une police }

     PALARY = array[ 1..16] of BYTE:    { Jeu de registres de palettes }

{-- Variables globales -------------------------------------------}

const vioptr : VELARPTR = ptr( $B800, $0000 ); { Pointe sur mém. écran }

var CharHauteur,
    lenx    : byte;        { Largeur en pixels de la fenêtre graphique }
    xmax,          { Coordonnées max en pixels de la fenêtre graphique }
    ymax    : integer;
    fontptr : fptr;                { Pointe sur la police de caractères }

procedure CLI; inline( $FA );
procedure STI; inline( $FB );

{*****************************************************************
* IsEgaVga : Teste la présence d'une carte EGA/VGA.              *
*            Mémorise en même temps le nombre de lignes de trame *
*            par caractère dans la variable CharHauteur          *
*****************************************************************
* Entrée : néant                                                 *
* Sortie : TRUE, si carte EGA ou VGA , sinon FALSE               *
*****************************************************************}

function IsEgaVga : boolean;

var Regs : Registers;             { Registres pour gérer l'interruption }

begin
  Regs.AX := $1a00;                { La fonction 1Ah n'existe qu'en VGA }
  Intr( $10, Regs );
  if ( Regs.AL = $1a ) then      { Cette fonction est-elle disponible ? }
    begin
      IsEgaVGa := TRUE;
      CharHauteur := 16;                              { 16 lignes de trame }
    end
  else
    begin
      Regs.ah := $12;             { AppelLe l'option 10h de la fonction }
      Regs.bl := $10;
      intr($10, Regs);          { Appelle l'interruption vidéo du BIOS }
      IsEgaVga := ( Regs.bl <> $10 );
      CharHauteur := 14;                              { 14 lignes de trame }
    end;
end;

{*****************************************************************
* SetCharWidth : Fixe à 8 ou 9 pixels la largeur des caractères des *
*                cartes VGA.                                        *
*****************************************************************
* Entrée: LARGEUT : Largeur des caractères ( 8 ou 9 )            *
*****************************************************************}

procedure SetCharWidth( largeur : byte );

var Regs : Registers;             { Registres pour gérer l'interruption }
    x    : byte;                                  { Variable de travail }

begin
  if ( largeur = 8 ) then Regs.BX := $0001    { BH = alignement horiz. }
                     else Regs.BX := $0800;   { BL = Valeur pour
                                              le registre du séquenceur }

  x := port[ $3CC ] and not(4+8);     { Passe de la résolution de 720 à }
  if ( largeur = 9 ) then                    { 640 pixels ou vice-versa }
    x := x or 4;
  port[ $3C2 ] := x;

  CLI;                             { Programme le séquenceur en conséquence }
  portw[ EGAVGA_SEQUENCER ] := $0100;
  portw[ EGAVGA_SEQUENCER ] := $01 + Regs.BL shl 8;
  portw[ EGAVGA_SEQUENCER ] := $0300;
  STI;

  Regs.AX := $1000;                          { Ajuste l'alignement de l'écran }
  Regs.BL := $13;
  intr( $10, Regs );
end;

{*****************************************************************
*  SelectMaps : Sélectionne les jeux de caractères accessibles par le *
*               bit 3 de l'attribut de caractères                 *
*****************************************************************
* Entrées : MAP0 = Numéro du premier jeu de caractères (bit 3 = 0 )  *
*           MAP1 = Numéro du deuxième jeu de caractères (bit 3 = 1 ) *
* Info    : - Avec une carte EGA on peut choisir les jeux 0 à 3,     *
*             avec une carte VGA les jeux 0 à 7                     *
*****************************************************************}

procedure SelectMaps( map0, map1 : byte );

var Regs : Registers;          { Registres pour gérer les interruptions }

begin
  Regs.AX := $1103;(Prog. le registre de sélection du jeu de caractères)
  Regs.BL := ( ( map0 and 3 ) + ( map0 and 4 ) shl 2 ) +
             ( ( map1 and 3 ) shl 2 + ( map1 and 4 ) shl 3 );
  Intr( $10, Regs ); { Appelle l'option 03 de la fonction $11 }
end;

{*****************************************************************
*  GetFontAccess : Permet d'accéder directement à la deuxième zone *
*                  de mémoire où sont stockés les jeux de caractères *
*                  par l'adresse A000:0000                          *
*****************************************************************
* Entrée  : néant                                                *
* Info    : - A l'issue de cette procédure, il n'est plus possible *
*             d'accéder à la mémoire d'écran par B800:0000       *
*****************************************************************}

procedure GetFontAccess;

const SeqRegs : array[1..4] of word = ( $0100, $0402, $0704, $0300 );
      GCRegs  : array[1..3] of word = ( $0204, $0005, $0406 );

var i : byte;                                  { Compteur d'itérations }

begin
  CLI;
  for i := 1 to 4 do             { Charge les registres du séquenceur }
    portw[ EGAVGA_SEQUENCER ] := SeqRegs[ i ];

  for i := 1 to 3 do    { Charge les registres du contrôleur graphique }
    portw[ EGAVGA_GRAPHCTR ] := GCRegs[ i ];
  STI;
end;

{*****************************************************************
*  ReleaseFontAccess : Libère l'accès à la mémoire d'écran par    *
*                      B800:0000 empêchent en même temps l'accès aux *
*                      jeux de caractère situés en page mémoire N°2 *
*****************************************************************
* Entrée : néant                                                 *
*****************************************************************}

procedure ReleaseFontAccess;
```

```
const SeqRegs : array[1..4] of word = ( $0100, $0302, $0304, $0300 );
      GCRegs  : array[1..3] of word = ( $0004, $1005, $0E06 );

var i : byte;                                   { Compteur d'itérations }

begin
  for i := 1 to 4 do   { Charge les différents registres du séquenceur }
    portw[ EGAVGA_SEQUENCER ] := SeqRegs[ i ];
  for i := 1 to 3 do     { Charge les registres du contrôleur graphique}
    portw[ EGAVGA_GRAPHCTR ] := GCRegs[ i ];
end;

{*********************************************************************
* ClearGraphArea : Efface la zone graphique en y mettant à 0 les     *
*                  motifs des caractères.                            *
**--------------------------------------------------------------------**
* Entrée : néant                                                     *
*********************************************************************}

procedure ClearGraphArea;

var caracteres,                                 { Caractères à parcourir }
    ligne     : byte;       { Lignes à l'intérieur de chaque caractère }

begin
  for caracteres := 0 to 255 do                 { Parcourt les caractères }
    for ligne := 0 to CharHauteur-1 do            { Parcourt les lignes }
      fontptr^[ caracteres, ligne ] := 0;             { et les met à 0 }
end;

{*********************************************************************
* InitGraphArea : Prépare une zone d'écran à recevoir un affichage   *
*                 graphique.                                         *
**--------------------------------------------------------------------**
* Entrée : X      = Colonne où débute la zone (1-80)                 *
*          Y      = Ligne où débute la zone (1-25)                   *
*          XLEN   = Largeur de la zone en caractères                 *
*          YLEN   = Profondeur de la zone en caractères              *
*          MAP    = Numéro du jeu de caractères graphiques           *
*          COULEUR = Couleur de la zone graphique (0 à 7 ou $FF)     *
* Info   : - Si la couleur indiquée est $FF , elle est variable      *
*            ce qui donne une effet de "mikado"                      *
*********************************************************************}

procedure InitGraphArea( x, y, xlen, ylen, map, couleur : byte );

var colonne, ligne : integer;                   { Compteur d'itérations }
    codec          : byte;                        { Code de caractère }

begin
  if ( xlen * ylen > 256 ) then                    { Zone trop grande ? }
    writeln( 'Erreur: La zone graphique ne doit pas englober ' +
             ' plus de 256 caracteres !' )
  else
    begin
      if ( CharHauteur = 16 ) then                              { VGA ? }
        SetCharWidth( 8 );   { Oui, on change la largeur des caractères }
      SelectMaps( 0, map );          { Sélectionne le jeu de caractères }
      xmax := xlen*CHAR_LARGEUR;              { Coordonnées max en pixels }
      ymax := ylen*CharHauteur;
      lenx := xlen;
      fontptr := ptr( $A000, map * $4000 ); { Pointe sur mém graphique }
      GetFontAccess;          { Autorise l'accès aux jeux de caractères }
      ClearGraphArea;                                  { Efface la zone }
      ReleaseFontAccess;        { rétablit l'accès à la mémoire d'écran }

      {-- Remplit la zone graphique avec des caractères ---------------}

      codec := 0;
      for ligne := ylen-1 downto 0 do          { Lignes de bas en haut }
        for colonne := 0 to xlen-1 do     { Colonnes de gauche à droite }
          begin              { Fixe le code et l'attribut du caractère }
            vioptr^[ligne*y,colonne*x].Caractere := codec;
            if ( couleur = $ff ) then
              vioptr^[ligne*y,colonne*x].Attribut := codec mod 6 + 1 + 8
            else
              vioptr^[ligne*y,colonne*x].Attribut := couleur or $08;
            inc( codec );                              { Caractère suivant }
          end;
    end;
end;

{*********************************************************************
* CloseGraphArea : Clôture l'accès à la zone graphique               *
**--------------------------------------------------------------------**
* Entrée : néant                                                     *
*********************************************************************}

procedure CloseGraphArea;

begin
  ReleaseFontAccess;              { Libère l'accès à la mémoire d'écran }
  SelectMaps( 0, 0 );                 { Choisit le jeu de caractères 0 }
  if ( CharHauteur = 16 ) then                                { VGA ? }
    SetCharWidth( 9 );              { Oui, fixe la largeur des caractères }
end;

{*********************************************************************
* SetPixel: Dessine ou efface un pixel dans la fenêtre graphique     *
**--------------------------------------------------------------------**
* Entrée : X,Y   = Coordonnées du pixel (0-...)                      *
*          ON    = TRUE pour dessiner et FALSE pour effacer          *
*********************************************************************}

procedure SetPixel( x, y : integer; on : boolean );

var charnum,                                    { Numéro du caractère }
    line     : byte;                { Line à l'intérieur du caractère }

begin
  if ( x < xmax ) and ( y < ymax ) then            { Coordonnées o.k.? }
    begin              { Oui calcule le numéro du caractère et la ligne }
      charnum := ((x div CHAR_LARGEUR) + (y div CharHauteur * lenx));
      line    := CharHauteur - ( y mod CharHauteur ) - 1;
      if on then                                  { Dessiner ou effacer ? }
        fontptr^[charnum, line] := fontptr^[charnum, line] or
            1 shl (CHAR_LARGEUR - 1 - ( x mod CHAR_LARGEUR ) )
      else
        fontptr^[charnum, line] := fontptr^[charnum, line] and
          not( 1 shl (CHAR_LARGEUR - 1 - ( x mod CHAR_LARGEUR ) ) );
    end;
end;

{*********************************************************************
* Line : Trace un segment dans la fenêtre graphique en appliquant    *
*        l'algorithme de Bresenham                                   *
**--------------------------------------------------------------------**
* Entrée : X1, Y1 = Coordonnées de l'origine (0 - ...)               *
*          X2, Y2 = Coordonnées de l'extrémité terminale             *
*          ON     = TRUE pour dessiner et FALSE pour effacer         *
*********************************************************************}

procedure Line( x1, y1, x2, y2 : integer; on : boolean );

var d, dx, dy,
    aincr, bincr,
    xincr, yincr,
    x, y                : integer;

{-- Procédure accessoire pour échanger deux variables entières --------}

procedure SwapInt( var i1, i2: integer );

var dummy : integer;

begin
  dummy := i2;
  i2    := i1;
  i1    := dummy;
end;

{-- Procédure principale ----------------------------------------------}

begin
  if ( abs(x2-x1) < abs(y2-y1) ) then{ Sens : par l'axe des X ou des Y }
    begin                                                { par l'axe des Y }
      if ( y1 > y2 ) then                          { y1 supérieur à y2 ? }
        begin
          SwapInt( x1, x2 );                     { Oui, échange X1 et X2, }
          SwapInt( y1, y2 );                                 { Y1 et Y2 }
        end;

      if ( x2 > x1 ) then xincr := 1          { Fixe le pas horizontal }
                     else xincr := -1;

      dy := y2 - y1;
      dx := abs( x2-x1 );
      d  := 2 * dx - dy;
      aincr := 2 * (dx - dy);
      bincr := 2 * dx;
      x := x1;
      y := y1;

      SetPixel( x, y, on );                  { Dessine le premier point }
      for y:=y1+1 to y2 do                     { Parcourt l'axe des Y }
        begin
          if ( d >= 0 ) then
            begin
              inc( x, xincr );
              inc( d, aincr );
            end
          else
            inc( d, bincr );
          SetPixel( x, y, on );
        end;
    end
  else                                                   { par l'axe des X }
    begin
```

```
    if ( x1 > x2 ) then                    ( x1 plus grand que x2 ? )
      begin
        SwapInt( x1, x2 );                 ( Oui, échange X1 et X2 )
        SwapInt( y1, y2 );                          ( Y1 et Y2 )
      end;

    if ( y2 > y1 ) then yincr := 1         ( Fixe le pas vertical )
                   else yincr := -1;

    dx := x2 - x1;
    dy := abs( y2-y1 );
    d  := 2 * dy - dx;
    aincr := 2 * (dy - dx);
    bincr := 2 * dy;
    x := x1;
    y := y1;

    SetPixel( x, y, on );                  ( Dessine le premier point )
    for x:=x1+1 to x2 do                     ( Parcourt l'axe des X )
      begin
        if ( d >= 0 ) then
          begin
            inc( y, yincr );
            inc( d, aincr );
          end
        else
          inc( d, bincr );
        SetPixel( x, y, on );
      end;
  end;
end;

{*********************************************************************
*  SetPalCol : définit une couleur dans une des 16 palettes          *
*              ou la couleur du cadre d'écran (Overscan-Color)       *
**-------------------------------------------------------------------**
*  Entrée : RegNr = Numéro du registre de palette (0 à 15) ou 16     *
*                   pour la couleur du cadre d'écran                 *
*           Col   = Code de couleur entre 0 et 15                   *
*********************************************************************}

procedure SetPalCol( RegNr : byte; Col : byte );

var Regs   : Registers;       { Registres pour gérer les interruptions }

begin
  Regs.AX := $1000;              { Option $00 de la fonction vidéo $10 }
  Regs.BH := Col;                                   { Code couleur }
  Regs.BL := RegNr;       { Numéro du registre du contrôleur d'attribut }
  Intr( $10, Regs );             { Appelle l'interruption vidéo du BIOS }
end;

{*********************************************************************
*  SetPalAry : Installe une nouvelle palette de 16 couleurs          *
*              sans changer la couleur du cadre d'écran.             *
**-------------------------------------------------------------------**
*  Entrée : NewCol = Table de palettes du type PALARY                *
*********************************************************************}

procedure SetPalAry( NewCol : PALARY );

var i : byte;   { Compteur d'itérations }

begin
  for i := 1 to 16 do  { Parcourt les 16 éléments de la table }
    SetPalCol( i-1, NewCol[ i ] ); { Fixe une couleur }
end;

{*********************************************************************
*  GetPalCol: Détermine le contenu d'un registre de palette          *
**-------------------------------------------------------------------**
*  Entrée : RegNr = Numéro du registre de palette (0 à 15) ou 16     *
*                   pour la couleur du cadre d'écran                 *
*  Sortie : Code de couleur                                         *
*  Info   : Avec les cartes EGA il n'est pas possible de lire le     *
*           contenu des registres de palette. On suppose que dans ce *
*           cas les registres de palette sont en disposition standard.*
*           la fonction retourne alors le numéro correspondant       *
*********************************************************************}

function GetPalCol( RegNr : byte ) : byte;

var Regs   : Registers;              { Registres pour l'interruption }

begin
  if ( CharHauteur = 14 ) then                        { Carte EGA ? }
    GetPalCol := RegNr{ Oui impossible de lire les registres de palette }
  else                                                { Non , VGA }
    begin
      Regs.AX := $1007;          { Option $07 de la fonction vidéo $10 }
      Regs.BL := RegNr;  { Numéro du registre du contrôleur d'attribut }
      Intr( $10, Regs );      { Déclenche l'interruption vidéo du BIOS }
      GetPalCol := Regs.BH;       { C'est ici que se trouve le contenu }
    end;                                          { du registre de palette }
end;

{*********************************************************************
*  GetPalAry : Lit les contenus des 16 registres de palette et les   *
*              transfère dans une table                              *
**-------------------------------------------------------------------**
*  Entrée : ColAry = Table de palettes de type PALARY, qui va recevoir *
*                    qui va recevoir les codes des couleurs          *
*********************************************************************}

procedure GetPalAry( var ColAry : PALARY );

var i : byte;                                  { Compteur d'itérations }

begin    for i := 1 to 16 do     { Parcourt les 16 éléments du tableau }
   ColAry[ i ] := GetPalCol( i-1 );   { Lit une couleur à chaque fois }
end;

{*********************************************************************
*  Mikado : Démonstration du maniement des routines présentées       *
*           dans ce programme                                        *
**-------------------------------------------------------------------**
*  Entrée : néant                                                   *
*********************************************************************}

procedure Mikado;

type Linie = record                          { Coordonnées d'un segment }
               x1, y1,
               x2, y2  : integer;
             end;

const NewCols : PALARY =
( {--------- Couleurs des caractères de texte ordinaires ----------}
          BLACK,                                          { noir }
          BLUE,                                           { bleu }
          GREEN,                                          { vert }
          RED,                                            { rouge }
          CYAN,                                           { cyan }
          MAGENTA,                                        { magenta }
          YELLOW,                                         { brun }
          WHITE,                                          { blanc }
                {------------ Couleurs des graphiques ------------}
          LIGHTBLUE,                                      { bleu clair }
          LIGHTGREEN,                                     { vert clair }
          LIGHTRED,                                       { rouge clair }
          LIGHTCYAN,                                      { cyan clair }
          LIGHTMAGENTA,                                   { magenta clair }
          BLUE,                                           { bleu }
          YELLOW,                                         { jaune }
          WHITE );                                        { blanc }

var i,                                       { Compteur d'itérations }
    first,                             { Indice du mikado le plus récent }
    last   : integer;                  { Indice du mikado le plus ancien }
    clear  : boolean;                        { Efface les mikados }
    lar    : array[1..MIKADOS] of linie;     { Table des mikados }
    OldCols: PALARY;                  { Table des anciennes couleurs }

begin
  GetPalAry( OldCols );              { Détermine les couleurs présentes }
  SetPalAry( NewCols );                { Installe une nouvelle palette }
  TextColor( 7 );
  TextBackGround( 1 );
  ClrScr;                                          { Efface l'écran }
  GotoXY(1,1);                           { Et le remplit de caractères }
  for i:=1 to 25*80-1 do                                { du jeu standard }
   write( chr(32 + i mod 224) );

  {-- Initialise la zone graphique et fait tomber les mikados ---------}

  GotoXY(27,6);
  TextColor( 7 );
  TextBackGround( 3 );
  write('      M I K A D O      ');
  GotoXY(27,6);
  InitGraphArea( 27, 7, 25, 10, 1, $FF );
  GetFontAccess;                  { Assure l'accès aux jeux de caractères }

  clear := false;                                { Pour effacer les mikados }
  first := 1;                              { Commence au début de la table }
  last := 1;
  repeat                                              { Boucle des mikados }
    if first = MIKADOS+1 then first := 1;                { Wrap-Around ? }
    lar[first].x1 := random( xmax-1 );                   { Crée un mikado }
    lar[first].x2 := random( xmax-1 );
    lar[first].y1 := random( ymax-1 );
    lar[first].y2 := random( ymax-1 );
    line( lar[first].x1, lar[first].y1,                  { et le dessine }
          lar[first].x2, lar[first].y2, true );
    inc( first );                                    { Mikado suivant }
    if first = MIKADOS+1 then clear := true;       { faut-il effacer ? }
```

```
    if clear then                                   ( maintenant ? )
      begin                                                 ( Oui )
        line( lar[last].x1, lar[last].y1,
              lar[last].x2, lar[last].y2, false );
        fnc( last );     ( Mikado suivant )
        if last = MIKADOS+1 then last := 1;
      end;
  until keypressed;              ( Répète jusqu'à détection d'une frappe )

  (-- Termine le programme ------------------------------------------)

  CloseGraphArea;
  SetPalAry( OldCols );                    ( Rétablit l'ancienne palette )
  GotoXY(1, 25 );
  TextColor( 7 );
  TextBackGround( 0 );
  ClrEol;
  writeln( 'Le jeu de caractères standard est à nouveau en place.');
end;

{-----------------------------------------------------------------------}
{---------------------- PROGRAMME PRINCIPAL ----------------------------}
{-----------------------------------------------------------------------}

begin
  if IsEgaVga then                        ( A-t-on une carte EGA ou VGA ? )
    Mikado                                ( Oui, c'est parti pour la démo )
  else                    ( Non, impossible de faire tourner le programme )
    writeln( 'Attention: aucune carte EGA ou VGA n''est installée !' );
end.
```

Listing : MIKADOC.C

```
/************************************************************************
*                          M I K A D O C . C                            *
**----------------------------------------------------------------------**
*  Fonction        : Montre comment mettre en service le mode 512       *
*                    caractères des cartes EGA et VGA.                  *
*                    La routine de démonstration installe une fenêtre   *
*                    graphique en mode texte                            *
**----------------------------------------------------------------------**
*  Auteur          : MICHAEL TISCHER                                    *
*  Développé le    : 2.04.1990                                          *
*  Dernière MAJ    : 14.02.1992                                         *
************************************************************************/

/*-- Constantes et fichiers d'en-tête ----------------------------------*/

#include <dos.h>
#include <stdarg.h>
#include <stdlib.h>
#include <stdio.h>
#include <conio.h>

#ifdef __TURBOC__                          /* Compilation par Turbo C? */
  #define CLI()           disable()
  #define STI()           enable()
  #define outpw( p, w )   outport( p, w )
  #ifndef inp
    #define outp( p, b )  outportb( p, b )
    #define inp( p )      inportb( p )
  #endif
#else                                      /* Non, par QuickC 2.0 ou MSC */
  #include <conio.h>
  #define random(x)       (rand() % ( x + 1 ))
  #define MK_FP(seg,ofs) ((void far *)\
                         (((unsigned long)(seg) << 16) | (ofs)))
  #define CLI()           _disable()
  #define STI()           _enable()
#endif

#define EGAVGA_SEQUENCER 0x3C4          /* Port adr/data du séquenceur */
#define EGAVGA_MONCTR    0x3D4                  /* Contrôleur vidéo */
#define EGAVGA_GRAPHCTR  0x3CE         /* Ports adr/data du ctrlr graph */
#define CHAR_LARGEUR     8
#define CHAR_BYTES       32
#define MIKADOS          5           /* Mikados visibles simultanément */

#define TRUE             (0 == 0)
#define FALSE            (0 == 1)

#define BLACK            0x00                  /* Attributs de couleur */
#define BLUE             0x01
#define GREEN            0x02
#define CYAN             0x03
#define RED              0x04
#define MAGENTA          0x05
#define BROWN            0x06
#define LIGHTGREY        0x07
#define GREY             0x01
#define LIGHTBLUE        0x09
#define LIGHTGREEN       0x0A
#define LIGHTCYAN        0x0B
#define LIGHTRED         0x0C
#define LIGHTMAGENTA     0x0D
#define YELLOW           0x0E
#define WHITE            0x0F

/*-- Déclarations de types ---------------------------------------------*/

typedef unsigned char    BYTE;
typedef BYTE             BOOL;

typedef BYTE             PALARY[16];  /* Jeu de registres de palettes */

/*-- Variables globales ------------------------------------------------*/

BYTE far *vioptr = (BYTE far *)0xB8000000L,       /* Mémoire graphique */
     far *fontptr;                    /* Pointe sur la police graphique */

BYTE CharHauteur,
     lenx;           /* Largeur en caractères de la fenêtre graphique */
int  xmax,       /* Coordonnées max en pixels de la fenêtre graphique */
     ymax;

/************************************************************************
*  IsEgaVga : Teste la présence d'une carte EGA ou VGA.                 *
**----------------------------------------------------------------------**
*  Entrée   : néant                                                     *
*  Sortie   : EGA, VGA ou NINI                                          *
************************************************************************/

BYTE IsEgaVga( void )
{
  union REGS Regs;          /* Registres pour gérer les interruptions */

  Regs.x.ax = 0x1a00;          /* La fonction 1Ah n'existe qu'en VGA */
  int86( 0x10, &Regs, &Regs );
  if( Regs.h.al == 0x1a )    /* Cette fonction est-elle disponible ? */
  {                                          /* Oui, on a une carte VGA */
    CharHauteur = 16;               /* Hauteur des caractères en VGA */
    return 1;
  }
  else
  {
    CharHauteur = 14;               /* Hauteur des caractères en EGA */
    Regs.h.ah = 0x12;                  /* Appelle l'option 10h de la */
    Regs.h.bl = 0x10;                                  /* fonction 12h */
    int86(0x10, &Regs, &Regs );     /* Déclenche l'interruption vidéo */
    return Regs.h.bl != 0x10;
  }
}

/************************************************************************
*  SetCursor   : Positionne le curseur clignotant                       *
**----------------------------------------------------------------------**
*  Entrées     : COLONNE = nouvelle colonne du curseur (0-79)           *
*                LIGNE = nouvelle ligne du curseur (0-24)               *
*  Sortie      : néant                                                  *
************************************************************************/

void SetCursor( BYTE colonne, BYTE ligne )
{
  union REGS  regs;        /*  Registres pour gérer les interruptions */

  regs.h.ah = 2;                 /* Numéro de la fonction "Set Cursor" */
  regs.h.bh = 0;                                  /* Accède à la page 0 */
  regs.h.dh = ligne;                                /* Fixe la ligne */
  regs.h.dl = colonne;                              /* et la colonne */
  int86(0x10, &regs, &regs);        /* Déclenche l'interruption vidéo */
}

/************************************************************************
*  PrintfAt : Affiche une chaîne formatée en n'importe quel point de    *
*             l'écran                                                   *
```

```
/*                                                                    *
 **--------------------------------------------------------------------**
 *  Entrées :   COLONNE = Position d'affichage                        *
 *              LIGNE                                                 *
 *              COULEUR = attribut des caractères                     *
 *              STRING  = Pointe sur la chaîne                        *
 *  Sortie  : néant                                                   *
 *  Info    : Cette fonction ne doit être invoquée qu'après           *
 *            vérification préalable de la présence d'une             *
 *            carte EGA ou VGA                                        *
 *--------------------------------------------------------------------*/

void PrintfAt( BYTE colonne, BYTE ligne, BYTE couleur, char *string,...)
{
  va_list   parametre;        /* Liste de paramètres pour macro _VA */
  char      Affichage[255],   /* Buffer pour la chaîne formatée */
            *affptr;
  BYTE far  *vptr;            /* Pointeur sur la mémoire d'écran */

  va_start( parametre, string );          /*Conversion des paramètres */
  vsprintf( Affichage, string, parametre );          /* Formatage */

  vptr = (BYTE far *) MK_FP( 0xB800, colonne * 2 + ligne * 160 );

  for( affptr = Affichage; *affptr ; )          /* Parcourt la chaîne */
  {
    *vptr++ = *(affptr++);            /* Ecrit dans la mémoire d'écran */
    *vptr++ = couleur;                      /* et de même les attributs */
  }
}

/*********************************************************************
 *  ClrScr : Efface l'écran                                          *
 **-------------------------------------------------------------------**
 *  Entrée : COULEUR  = attribut des caractères                      *
 *  Sortie : néant                                                   *
 *********************************************************************/

void ClrScr( BYTE couleur )
{
  BYTE far  *vptr;                    /* Pointe sur la mémoire d'écran */
  int       count = 2000;             /* Nombre de caractères à effacer */

  vptr = (BYTE far *) MK_FP( 0xB800, 0 );          /* Fixe le pointeur */

  for( ; count--; )                       /* Parcourt la mémoire d'écran */
  {
    *vptr++ = ' ';            /* Ecrit les caractères en mémoire d'écran */
    *vptr++ = couleur;                      /* et de même les attributs */
  }
}

/*********************************************************************
 *  SetCharWidth : Fixe la largeur des caractères des cartes VGA     *
 *                 à 8 ou 9 pixels                                   *
 **-------------------------------------------------------------------**
 *  Entrée : Largeur = Largeur des caractères (8 ou 9)               *
 *********************************************************************/

void SetCharWidth( BYTE largeur )
{
  union REGS Regs;            /* Registres pour gérer les interruptions */
  unsigned char x;                          /* Variable de travail */
  Regs.x.bx = ( largeur == 8 ) ? 0x0001 : 0x0800;

  x = inp( 0x3CC ) & (255-12);      /* Passe de la résolution de 720 à */
  if ( largeur == 9 )                       /* 640 pixels ou vice-versa */
    x |= 4;
  (void)outp( 0x3C2, x);

  CLI();                          /* Programme le séquenceur en conséquence */
  outpw( EGAVGA_SEQUENCER, 0x0100 );
  outpw( EGAVGA_SEQUENCER, 0x01 + ( Regs.h.bl << 8 ) );
  outpw( EGAVGA_SEQUENCER, 0x0300 );
  STI();

  Regs.x.ax = 0x1000;                   /* Ajuste l'écran horizontalement */
  Regs.h.bl = 0x13;
  int86( 0x10, &Regs, &Regs );
}

/*********************************************************************
 *  SelectMaps : Sélectionne les jeux de caractères accessibles par  *
 *               le bit 3 de l'attribut de caractère                 *
 **-------------------------------------------------------------------**
 *  Entrée : MAP0 = Numéro du premier jeu de caractères  (Bit 3 = 0 ) *
 *           MAP1 = Numéro du deuxième jeu de caractères (Bit 3 = 1 ) *
 *  Info   : - avec une carte EGA on peut choisir les jeux 0 à 3,     *
 *           - avec une carte VGA les jeux 0 à 7                      *
 *********************************************************************/

void SelectMaps( BYTE map0, BYTE map1)
{
  union REGS  Regs;       /* Registres pour gérer les interruptions */

  Regs.x.ax = 0x1103;           /* Registre de sélection des caractères */
  Regs.h.bl = ( map0 & 3 ) + ( ( map0 & 4 ) << 2 ) +
              ( ( map1 & 3 ) << 2 ) + ( ( map1 & 4 ) << 3 );
  int86( 0x10, &Regs, &Regs );    /* Déclenche l'interruption du BIOS */
}

/*********************************************************************
 *  GetFontAccess : Permet d'accéder directement à la deuxième zone  *
 *                  de mémoire où sont stockés les jeux de caractères*
 *                  par l'adresse A000:0000                          *
 **-------------------------------------------------------------------**
 *  Entrée : néant                                                   *
 *  Info   : A l'issue de cette procédure il n'est plus possible     *
 *           d'accéder à la mémoire d'écran par B800:0000            *
 *********************************************************************/

void GetFontAccess( void )
{
  static unsigned  SeqRegs[4] = { 0x0100, 0x0402, 0x0704, 0x0300 },
                   GCRegs[3]  = { 0x0204, 0x0005, 0x0006 };
  BYTE             i;                          /* Compteur d'itérations */

  CLI();                                 /* Pas d'interruption maintenant */

  for ( i=0; i<4; ++i )           /* Charge les registres du séquenceur */
    outpw( EGAVGA_SEQUENCER, SeqRegs[ i ] );

  for ( i=0; i<3; ++i )           /* Charge les registres du ctrlr graph */
    outpw( EGAVGA_GRAPHCTR, GCRegs[ i ] );

  STI();                                 /* rétablit les interruptions */
}

/*********************************************************************
 *  ReleaseFontAccess : Rétablit l'accès à la mémoire d'écran par    *
 *                      B800:0000 mais empêche en même temps l'accès *
 *                      aux jeux de caractères                       *
 *                      situés en page mémoire N° 2.                 *
 **-------------------------------------------------------------------**
 *  Entrée            : néant                                        *
 *********************************************************************/

void ReleaseFontAccess( void )
{
  static unsigned   SeqRegs[4] = { 0x0100, 0x0302, 0x0304, 0x0300 },
                    GCRegs[3]  = { 0x0004, 0x1005, 0x0E06 };
  BYTE              i;                          /* Compteur d'itérations */

  CLI();                                 /* Pas d'interruptions maintenant */

  for( i=0; i<4; ++i )            /* Charge les registres du séquenceur */
    outpw( EGAVGA_SEQUENCER, SeqRegs[ i ] );

  for ( i=0; i<3; ++i )           /* Charge les registres du Ctrl Graph */
    outpw( EGAVGA_GRAPHCTR, GCRegs[ i ] );

  STI();                                 /*Rétablit les interruptions */
}

/*********************************************************************
 *  ClearGraphArea : Efface la zone graphique en y mettant à 0 les   *
 *                   motifs des caractères.                          *
 **-------------------------------------------------------------------**
 *  Entrée           : néant                                         *
 *********************************************************************/

void ClearGraphArea( void )
{
  int  caract,                            /* Caractères à parcourir */
       ligne;                  /* Ligne à l'intérieur de chaque caractère */

  for( caract = 0; caract < 256; ++caract )          /* Caractères */
    for( ligne = 0; ligne < CharHauteur; ++ ligne )      /* Lignes */
      *(fontptr + caract * CHAR_BYTES + ligne) = 0;      /* Mise à 0 */
}

/*********************************************************************
 *  InitGraphArea  : Prépare une zone d'écran à recevoir un affichage*
 *                   graphique                                       *
 **-------------------------------------------------------------------**
 *  Entrée         : X      = Colonne où débute la zone (1-80)       *
 *                   Y      = Ligne où débute la zone   (1-25)       *
 *                   XLEN   = Largeur de la zone en caractères       *
 *                   YLEN   = Hauteur de la zone en caractères       *
 *                   MAP    = Numéro du jeu de caractères graphiques *
 *                   COULEUR = Couleur de la zone graphique          *
 *                             (0 à 7 ou 0xFF)                       *
 *  Info    : - Si la couleur indiquée est 0xFF, elle est variable   *
 *              ce qui donne un effet de "mikado"                    *
 *********************************************************************/
```

```
/
void InitGraphArea( BYTE x, BYTE y, BYTE xlen, BYTE ylen, BYTE map,
                    BYTE couleur )
{
  unsigned  offset;                       /* Offset en mémoir d'écran */
  int       colonne, ligne;               /* Compteur d'itérations */
  BYTE      codec;                        /* Code de caractère */

  if( xlen > 256 )                        /* Zone trop grande ? */
    printf( "Erreur : La zone graphique ne doit pas englober"\
            " plus de 256  caractères !\n" );
  else
  {
    if( CharHauteur == 16 )                                 /* VGA? */
      SetCharWidth( 8 );     /* Oui, change la largeur des caractères */
    SelectMaps( 0, map );          /* Sélectionne le jeu de caractères */
    xmax = xlen*CHAR_LARGEUR;             /* Coordonnées max en pixels */
    ymax = ylen*CharHauteur;
    lenx = xlen;
    fontptr = MK_FP( 0xA000, map * 0x4000 );      /* Mémoire graphique */
    GetFontAccess();        /* Autorise l'accès aux jeux de caractères */
    ClearGraphArea();                             /* Efface la zone */
    ReleaseFontAccess();     /* Rétablit l'accès à la mémoire d'écran */

    /*-- remplit la zone graphique avec des caractères ---------------*/

    codec = 0;
    for( ligne = ylen-1; ligne >= 0; --ligne )              /* Lignes */
      for ( colonne = 0; colonne < xlen; ++colonne )       /* Colonnes */
      {                          /* Fixe le code et l'attribut du caractère */
        offset = ((ligne*y) * 80 + colonne*x) << 1; /* Ofst mém écran */
        *(vioptr+offset) = codec;              /* Ecrit le caractère */
        *(vioptr+offset+1) = ( couleur == 0xff )
                                 ? ( codec % 6 ) + 1 + 8
                                 : couleur | 0x08;
        ++codec;                          /* Passe au caractère suivant */
      }
  }
}

/*************************************************************************
*  CloseGraphArea : Clôture l'accès à la zone graphique                  *
**-----------------------------------------------------------------------**
*  Entrée         : néant                                                *
*************************************************************************/

void CloseGraphArea( void )
{
  ReleaseFontAccess();          /* Libère l'accès à la mémoire d'écran */
  SelectMaps( 0, 0 );               /* Choisit le jeu de caractères 0 */
  if( CharHauteur == 16 )                                   /* VGA? */
    SetCharWidth( 9 );          /* Oui, fixe la largeur des caractères */
}

/*************************************************************************
*  SetPixel: Dessine ou efface un pixel dans la fenêtre graphique        *
**-----------------------------------------------------------------------**
*  Entrées : X,Y   = Coordonnées du pixel  (0-...)                       *
*            ON    = TRUE pour dessiner et FALSE pour effacer            *
*************************************************************************/

void SetPixel( int x, int y, BOOL on )
{
  BYTE charnum,                           /* Numéro du caractère */
       linenr,                    /* Ligne à l'intérieur du caractère */
       far *bptr;

  if( ( x < xmax ) && ( y < ymax ) )              /* Coordonnées o.k.? */
  {                    /* Oui calcule le numéro du caractère et la ligne */
    charnum = ((x / CHAR_LARGEUR) + (y / CharHauteur * lenx));
    linenr  = CharHauteur - ( y % CharHauteur ) - 1;
    bptr = fontptr + charnum * CHAR_BYTES + linenr;
    if( on )                                  /* Dessiner ou effacer ? */
      *bptr= *bptr | ( 1 << (CHAR_LARGEUR - 1 - ( x % CHAR_LARGEUR )));
    else
      *bptr= *bptr & !( 1 << (CHAR_LARGEUR - 1 - ( x % CHAR_LARGEUR )));
  }
}

/*************************************************************************
*  Line : Trace un segment dans la fenêtre graphique en appliquant       *
*         l'algorithme de Bresenham                                     *
**-----------------------------------------------------------------------**
*  Entrées : X1, Y1 = Coordonnées de l'origine                           *
*            X2, Y2 = Coordonnées de l'extrémité terminale               *
*            ON     = TRUE pour dessiner et FALSE pour effacer           *
*************************************************************************/

/*-- Fonction accessoire pour échanger deux variables entières -------*/

void SwapInt( int *i1, int *i2 )
{
  int   dummy;

  dummy = *i2;
  *i2   = *i1;
  *i1   = dummy;
}

/*-- Procédure principale ----------------------------------------------*/

void Line( int x1, int y1, int x2, int y2, BOOL on )
{
  int d, dx, dy,
      aincr, bincr,
      xincr, yincr,
      x, y;

  if( abs(x2-x1) < abs(y2-y1) )    /* Sens du parcours : axe X ou Y ? */
  {                                                        /* Par Y */
    if( y1 > y2 )                          /* y1 plus grand que y2? */
    {
      SwapInt( &x1, &x2 );                 /* Oui échange X1 et X2, */
      SwapInt( &y1, &y2 );                                 /* Y1 et Y2 */
    }

    xincr = ( x2 > x1 ) ?  1 : -1;          /* Fixe le pas horizontal */

    dy = y2 - y1;
    dx = abs( x2-x1 );
    d  = 2 * dx - dy;
    aincr = 2 * (dx - dy);
    bincr = 2 * dx;
    x = x1;
    y = y1;

    SetPixel( x, y, on );                  /* dessine le premier pixel */
    for( y=y1+1; y<= y2; ++y )                /* Parcourt l'axe des Y */
    {
      if( d >= 0 )
      {
        x += xincr;
        d += aincr;
      }
      else
        d += bincr;
      SetPixel( x, y, on );
    }
  }
  else                                                      /* par X */
  {
    if( x1 > x2 )                          /* x1 plus grand que x2? */
    {
      SwapInt( &x1, &x2 );                 /* Oui, échange X1 et X2 */
      SwapInt( &y1, &y2 );                                 /* Y1 et Y2 */
    }

    yincr = ( y2 > y1 ) ? 1 : -1;           /* Fixe le pas vertical */

    dx = x2 - x1;
    dy = abs( y2-y1 );
    d  = 2 * dy - dx;
    aincr = 2 * (dy - dx);
    bincr = 2 * dy;
    x = x1;
    y = y1;

    SetPixel( x, y, on );                  /* Dessine le premier pixel */
    for( x=x1+1; x<=x2; ++x )                 /* Parcourt l'axe des X */
    {
      if( d >= 0 )
      {
        y += yincr;
        d += aincr;
      }
      else
        d += bincr;
      SetPixel( x, y, on );
    }
  }
}

/*************************************************************************
*  SetPalCol : Définit une couleur dans une des 16 palettes              *
*              ou la couleur du cadre d'écran (Overscan-Color)           *
**-----------------------------------------------------------------------**
*  Entrée    : RegNr = Numéro du registre de palette (0 à 15) ou 16      *
*                      pour la couleur du cadre d'écran                  *
*              Col   = Code de la couleur de 0 à 15                      *
*************************************************************************/

void SetPalCol( BYTE RegNr, BYTE Col )
{
  union REGS  Regs;       /*  Registres pour gérer les interruptions */

  Regs.x.ax = 0x1000;           /* Option 00h de la fonction vidéo 10h */
```

```
    Regs.h.bh = Col;                                    /* Code couleur */
    Regs.h.bl = RegNr;   /* Numéro du registre du contrôleur d'attribut */
    int86( 0x10, &Regs, &Regs );       /* Déclenche l'interruption vidéo */
}

/*********************************************************************
*  SetPalAry : installe une nouvelle palette de 16 couleurs sans     *
*              changer la couleur du cadre d'écran.                  *
*--------------------------------------------------------------------*
*  Entrée    : NewColPtr = Pointeur sur une palettes de type PALARY  *
*********************************************************************/

void SetPalAry( BYTE *NewColPtr )
{
  BYTE   i;                                 /* Compteur d'itérations */

  for( i = 0; i < 16; ++i )   /* Parcourt les 16 éléments de la table */
    SetPalCol( i, NewColPtr[i] );   /* Fixe une couleur à chaque fois */
}

/*********************************************************************
*  GetPalCol: Lit le contenu d'un registre de palette                *
*--------------------------------------------------------------------*
*  Entrée : RegNr = Numéro du registre de Palette (0 à 15) ou 16     *
*                   pour la couleur du cadre d'écran                 *
*  Sortie : Code de couleur                                          *
*  Info   : Avec EGA il n'est pas possible de lire le contenu        *
*           des registres de palette. On suppose que dans ce cas     *
*           les registres de palette sont en disposition standard    *
*           et la fonction retourne le numéro correspondant.         *
*********************************************************************/

BYTE GetPalCol( BYTE RegNr )
{
 union REGS Regs;            /*  Registres pour gérer les interruptions */

 if ( CharHauteur == 14 )                                /* Carte EGA? */
  return RegNr;   /*Oui, impossible de lire les registres de palettes */
 else                                                    /* Non, VGA */
 {
   Regs.x.ax = 0x1007;          /* Option 07h de la fonction vidéo 10h */
   Regs.h.bl = RegNr;  /* Numéro du registre du contrôleur d'attribut */
   int86( 0x10, &Regs, &Regs );          /* Interruption vidéo du BIOS */
   return Regs.h.bh;                  /* Contenu du registre de palette */
 }
}

/*********************************************************************
*  GetPalAry: Lit les contenus des 16 registres de palette et les    *
*             enregistre dans une table          .                  *
*--------------------------------------------------------------------*
*  Entrée : ColAryPtr = Pointeur sur une table de palettes du type   *
*                       PALARY, qui va recevoir les codes des couleurs *
*********************************************************************/

void GetPalAry( BYTE *ColAryPtr )
{
 BYTE i;                                   /* Compteur d'itérations */

 for (i = 0; i < 16; ++i )     /* parcourt les 16 éléments du tableau */
  ColAryPtr[i] = GetPalCol( i );      /* lit une couleur à chaque fois */
}

/*********************************************************************
*  Mikado: Démonstration du maniement des routines présentées        *
*          dans ce programme                                         *
*--------------------------------------------------------------------*
*  Entrée : néant                                                    *
*********************************************************************/

void Mikado( void )
{
 typedef struct {                          /* Coordonnées d'un segment */
                int x1, y1,
                    x2, y2;
              } LIGNE;

static PALARY NewCols =
     {              /*-- Couleurs des caractères de texte ordinaires --*/
      BLACK,                                                /* noir */
      BLUE,                                                 /* bleu */
      GREEN,                                                /* vert */
      RED,                                                  /* rouge */
      CYAN,                                                 /* cyan */
      MAGENTA,                                              /* magenta */
      YELLOW,                                               /* jaune */
      WHITE,                                                /* blanc */
                 /*---------Couleurs des graphiques ----------------*/
      LIGHTBLUE,                                            /* bleu clair */
      LIGHTGREEN,                                           /* vert clair */
      LIGHTRED,                                             /* rouge clair */
          LIGHTCYAN,                                        /* cyan clair */
          LIGHTMAGENTA,                                     /* magenta clair */
          BLUE,                                             /* bleu */
          YELLOW,                                           /* jaune */
          WHITE };                                          /* blanc */

int    i, j,                              /* Compteur d'itérations */
       first,                        /* Indice du mikado le plus récent */
       last;                         /* Indice du mikado le plus ancien */
BOOL   clear;                             /* Pour effacer les mikados */
LIGNE  lar[MIKADOS];                          /* Table des mikados */
PALARY OldCols;                        /* Table des anciennes couleurs */

GetPalAry( OldCols );                 /* Détermine les couleurs présentes */
SetPalAry( NewCols );                   /* Installe une nouvelle palette*/
          /*TextColor( 7 );
TextBackGround( 1 );
GotoXY(1,1); */
ClrScr( 0x07 );                                       /* Efface l'écran */
for (i=0; i<25; ++i )   /* puis le remplit avec un jeu de caractères */
 for (j=0; j<80; ++j )
  PrintfAt( j, i, 0x07, "%c", 32 + (((int) i*80+j) % 224) );

/*-- Initialise la zone graphique et fait tomber les mikados---------*/

PrintfAt( 27,6, 0x70, "      M I K A D O      " );
SetCursor(27,6);
InitGraphArea( 27, 7, 25, 10, 1, 0xFF );
GetFontAccess();            /* Assure l'accès aux jeux de caractères */

clear = FALSE;                            /* Pour effacer les mikados */
first = 0;                          /* Commence au début de la table */
last = 0;
do                                                  /* Boucle des mikados*/
 {                                                    /* Wrap-Around? */
  if (first == MIKADOS )
   first = 0;
  lar[first].x1 = random( xmax-1 );                 /* Crée un mikado */
  lar[first].x2 = random( xmax-1 );
  lar[first].y1 = random( ymax-1 );
  lar[first].y2 = random( ymax-1 );
  Line( lar[first].x1, lar[first].y1,                   /* et le dessine */
        lar[first].x2, lar[first].y2, TRUE );
  if ( ++first == MIKADOS )                        /* faut-il effacer ? */
   clear = TRUE;
  if ( clear )                                  /* On efface maintenant ? */
   {                                                          /* Oui */
    Line( lar[last].x1, lar[last].y1,
          lar[last].x2, lar[last].y2, FALSE );
    if ( ++last == MIKADOS )
     last = 0;
   }
 }
while (!kbhit());      /* Répète l'opération jusqu'à frappe de touche */
getch();                        /* Retire la touche du buffer du clavier */
/*-- Termine le programme ------------------------------------------*/

CloseGraphArea();
SetPalAry( OldCols );      /* Restaure l'ancienne palette de couleurs */
SetCursor(0,24);
printf( "\nLe jeu de caractères standard est à nouveau en place .\n" );
}

/*--------------------------------------------------------------------*/
/*---------------------- PROGRAMME PRINCIPAL --------------------------*/
/*--------------------------------------------------------------------*/

void main()
{
 if ( IsEgaVga() )                       /* A-t-on une carte EGA ou VGA ? */
  Mikado();                              /* Oui, c'est parti pour la démo */
 else              /* Non, impossible de faire tourner le programme */
  printf( "Attention: aucune carte EGA ou VGA n'est installée !" );
}
```

Jeux de caractères du mode graphique

Le mode graphique des cartes EGA et VGA a également besoin de tables de caractères lorsqu'il s'agit de sortir des caractères sur l'écran à l'aide des fonctions BIOS appropriées. Le plan de bits #2 n'est plus disponible pour recevoir les tables parce qu'en mode graphique il doit collaborer avec d'autres plans de bits pour calculer les informations de points pour l'écran graphique.

Dans ce mode, le BIOS doit extraire directement les motifs de points de chaque caractère de la structure ROM concernée ou d'un buffer RAM. Les deux possibilités sont admises mais il faut déterminer le jeu à utiliser tout comme dans le mode texte.

En mode graphique, le BIOS sélectionne automatiquement le jeu activé en mode texte 80*25 caractères sans que vous ayez besoin de l'en informer. Avec une carte EGA, il s'agit du jeu 8*14, avec une carte VGA du jeu 8*16. Comme dans les autres cas, le nombre de lignes de texte à afficher sur l'écran graphique correspond au quotient résultant de la résolution verticale et de la hauteur de caractères. Il en est de même pour le nombre de caractères de texte contenus dans une ligne écran en mode graphique où le diviseur vaut toujours huit.

Pour la sortie de texte en mode graphique, vous ne devez pas avoir recours au jeu prédéfini. Sélectionnez au contraire un jeu quelconque prédéfini dans la ROM à l'aide du BIOS. Utilisez à cet effet les sous-fonctions 22h, 23h et 24h de la fonction 11h du BIOS vidéo. Dans les ouvrages spécialisés, on peut lire qu'en dehors des numéros de fonction transmis dans les registres AL et AH, ces fonctions attendent deux autres informations dans les registres BL et DL. A notre connaissance, cela n'a pas d'importance puisque ces paramètres n'influent pas sur la sortie de texte en mode graphique et peuvent donc être ignorés. Le numéro de fonction suffit amplement pour l'appel de ces fonctions.

Sous-fonction	Matrice	EGA	VGA
22h	8*14	■	■
23h	8*8	■	■
24h	8*16		■

Une autre méthode consiste à utiliser la sous-fonction 21h pour charger une table de caractères personnelle en mode graphique. Dans ce cas, elle doit comporter les 256 caractères. En dehors des numéros de fonction dans les registres AH et AL, il faut transmettre la hauteur des différents caractères dans le registre CX. Tout comme pour les autres fonctions BIOS pour la définition de caractères, cette fonction attend un pointeur sur la table dans le registre ES:BP pour son appel. Sa structure doit correspondre à celle des tables de caractères attendue également par les sous-fonctions 00h et 10h.

4.8.2. Smooth Scrolling

Avec les anciennes cartes EGA et VGA, il était fort peu commode pour l'utilisateur de créer l'impression d'une image en mouvement. En mode texte, on ne pouvait décaler un caractère que vers une direction précise et en mode graphique, il fallait recopier la totalité de la RAM vidéo pour peu qu'on souhaite déplacer l'écran d'un point. Dans le domaine de l'animation, il ne faut pas s'étonner alors si le PC est largement distancé par les concurrents.

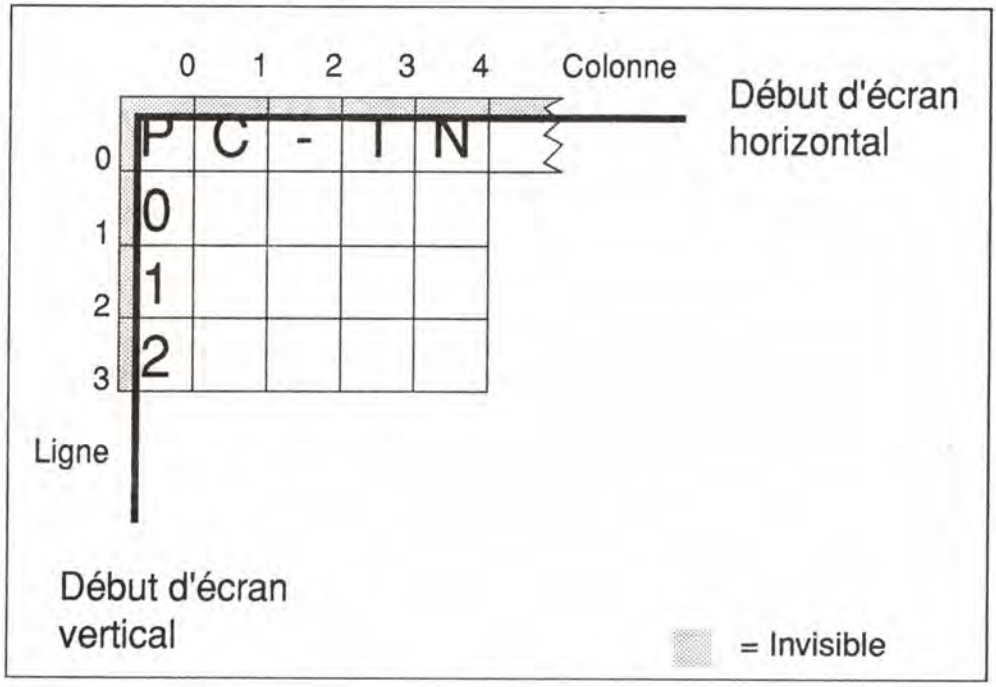
La situation a changé avec les cartes EGA et VGA. Elles offrent, de base, la possibilité de déplacer l'écran en fonction d'un point et réaliser ainsi un défilement plus atténué désigné aussi par Smooth Scrolling. Il devient alors possible de programmer des jeux d'actions basés essentiellement sur un arrière-plan en mouvement. Mais on arrive aussi à créer des effets intéressants en mode texte comme vous allez le constater au cours de cette section. Voici les thèmes étudiés :

- ✓ Smooth Scrolling à l'aide du registre Pel Panning
- ✓ Déplacement du point d'origine de l'écran et modification de la longueur de ligne dans la RAM vidéo
- ✓ Synchronisation du déplacement avec le contrôleur CRT
- ✓ Texte continu en mode texte

Smooth Scrolling à l'aide du registre Pel Panning

Dans les cartes EGA et VGA, la technique permettant de déplacer point par point la portion visible de l'écran est offerte par deux registres dont les fonctionnalités sont complètement différentes. En principe, les deux registres remplissent la même fonction consistant notamment à spécifier le point de départ de l'écran tant sur le plan horizontal que vertical.

Il faut savoir toutefois que le registre Pel Panning vertical fait partie du contrôleur CRT alors que le registre Pel Panning horizontal se situe dans le contrôleur d'attributs. A vrai dire, seuls les développeurs de la première carte EGA peuvent expliquer cette situation.



Déplacement du point de départ de l'écran à l'aide du registre Pel Panning

Commençons notre étude par le registre Pel Panning horizontal. Normalement, il contient la valeur 0 ou 8 en mode texte compte tenu de la chasse des caractères. Ces deux valeurs veillent à ce que les caractères affichés sur l'écran tiennent compte des habitudes optiques sur le plan horizontal.

La valeur 8 représente la position normale pour tous les modes vidéo dans lesquels la chasse est définie à 9 points. Cela concerne principalement la carte VGA ainsi que la carte EGA lorsqu'elles sont reliées à un moniteur monochrome et émule de ce fait une carte MDA. Pour une carte EGA connectée à un moniteur couleur, la valeur standard vaut par contre 0.

En partant de cette valeur de base, vous pouvez à présent faire défiler l'écran point par point vers la gauche en augmentant progressivement la valeur dans le registre Pel Panning horizontal. Avec une carte EGA sur un moniteur couleur, les valeurs se suivent dans l'ordre 1, 2, 3 etc. Plus la valeur est élevée et plus le décalage de l'écran vers la gauche est grand. La valeur 7 correspond à la fin du déplacement car une valeur élevée aurait tendance à décaler l'écran de plus d'un caractère, ce qui n'est pas le cas avec le registre Pel Panning horizontal.

La procédure est quelque peu différente avec des cartes EGA et VGA sur un moniteur monochrome. Après la valeur initiale 8, on rencontre ici la valeur 0 au lieu de 9 ce qui

fait décaler l'écran d'un point vers la gauche. Comme avec la carte EGA, viennent alors les valeurs 1, 2, 3 etc. jusqu'à 7 qui correspond au déplacement maximal.

Etant donné qu'avec le registre Pel Panning horizontal, le déplacement ne s'effectue que vers la gauche, il reste à savoir comment obtenir un déplacement vers la droite. Il suffit ici de commencer par la valeur maximale 7 et revenir progressivement en arrière jusqu'à 0. Avec les cartes VGA et EGA sur un moniteur MDA, vous pouvez tout simplement sauter de 0 à 8 pour que l'écran reprenne son statut initial.

Le registre Pel Panning horizontal est utilisable à la fois en mode texte et en mode graphique. Une différence s'effectue alors entre les modes 256 couleurs d'une carte VGA et tous les autres modes. En mode 256 couleurs, la valeur de base vaut également 0 mais l'écran ne se déplace que de trois points vers la gauche. A cet effet, il ne faut pas charger les valeurs 1, 2 et 3 dans le registre Pel Panning horizontal mais les valeurs 2, 4 et 6.

Dans les autres modes graphiques, la valeur de base vaut 0. Vous pouvez en outre charger les valeurs 1 à 7 dans le registre Pel Panning horizontal pour décaler la portion visible de l'écran de 7 points au maximum vers la gauche.

Pour charger le registre Pel Panning horizontal, vous pouvez programmer directement le contrôleur d'attributs ou utiliser une fonction BIOS qui exécute une tâche complètement différente. Il s'agit de la sous-fonction 00h de la fonction 10h de l'interruption vidéo du BIOS. Elle permet de charger un registre de palettes déterminé après avoir spécifié son numéro.

Sachant que les registres de palette se situent dans le contrôleur d'attributs tout comme le registre Pel Panning horizontal, vous pouvez utiliser cette fonction pour appeler le registre Pel Panning horizontal en précisant tout simplement le numéro de ce registre. En dehors des deux numéros de fonction dans le registre BL, il suffit de charger le numéro du registre, notamment 13h, et la valeur à placer dans le registre dans le registre BH.

Bien qu'elle ne soit pas très rapide, cette méthode convient parfaitement pour accéder directement au contrôleur d'attributs parce qu'elle permet d'isoler un programme des différences entre l'électronique des diverses cartes EGA et VGA. Par ailleurs, les constructeurs de cartes EGA et VGA respectent l'affectation de registres définie par IBM, si bien qu'on n'est pas induit en erreur lors de la programmation directe du registre Pel Panning horizontal.

Si vous choisissez cette méthode directe, vous devez d'abord inscrire le numéro du registre (13h) dans le registre combiné de données et d'indices du contrôleur d'attributs situé à l'adresse de port 3C0h. N'oubliez pas non plus de fixer le bit 5 dans ce registre bien qu'il n'ait rien à avoir avec le numéro de registre. Il représente en fait le commutateur d'activation et de désactivation du contrôleur d'attributs. S'il est effacé, le contrôleur d'attributs achève son travail et l'écran devient noir.

Une fois la valeur 33h (numéro de registre 13h plus bit 5) envoyée au port 3C0h, on peut y placer également la nouvelle valeur du registre appelé. Dans ce cas, il s'agit du nouveau compteur de pixels destiné au registre Pel Panning horizontal.

Voici donc pour le défilement horizontal de la portion visible de l'écran. La procédure est plus simple en ce qui concerne le défilement vertical à travers le registre Pel Panning vertical. Ici, la valeur de base est toujours 0 quelle que soit la carte vidéo ou quel que soit le mode d'affichage. Toute valeur supérieure fait décaler la portion visible de l'écran d'un nombre de points correspondants vers le haut. Dans les modes texte, la limite supérieure dépend de la hauteur des caractères. En mode texte normal de 80*25 caractères, cette valeur vaut 13 avec une carte EGA et 15 avec une carte VGA.

En mode graphique, on peut choisir des valeurs entre 0 et 31 pour le registre Pel Panning vertical où la valeur 31 représente la valeur de base et non 0. Toute valeur inférieure fait décaler la portion visible de l'écran d'un nombre de points correspondants vers le bas. Ici, il convient donc de régler le registre Pel Panning vertical sur sa position de base minimale afin de pouvoir décaler ensuite la portion visible de l'écran vers le haut en incrémentant ce registre.

Pour faire défiler la portion visible de l'écran vers le bas en mode texte, on procède comme pour le déplacement horizontal : commencer par la valeur supérieure et la décrémenter de un jusqu'à atteindre la valeur de base.

Contrairement au registre Pel Panning horizontal, on doit appeler le registre Pel Panning vertical à travers le contrôleur CRT - sans avoir recours à une fonction BIOS. Dans le registre d'index du contrôleur CRT, il faut préciser d'abord le numéro de registre, ici 08h. L'adresse de port où se situe le registre d'index dépend du mode d'exploitation de la carte vidéo : en mode couleur, il se trouve à l'adresse de port 3D4h, en mode monochrome à l'adresse 3B4h. Vient ensuite le registre de données devant contenir la nouvelle valeur du registre Pel Panning vertical. Au lieu de sélectionner ici une opération 8 bits, on peut opter pour une opération 16 bits et sortir simultanément la valeur pour le registre de données et le registre d'index.

Il ne faut pas croire que la programmation des deux registres Pel Panning consiste tout simplement à décaler la portion visible de l'écran par rapport à la hauteur ou la largeur d'un caractère. L'écran doit en plus se déplacer continuellement dans une direction donnée d'un caractère à l'autre.

A première vue, on est tenté de déplacer la portion de l'écran d'un caractère dans la direction souhaitée à l'aide des registres Pel Panning, de régler ensuite le registre Pel Panning modifié sur sa valeur initiale et déplacer simultanément le contenu de la RAM vidéo pour que tous les caractères soient décalés d'une position dans la direction souhaitée. On peut naturellement répéter cette opération autant de fois que nécessaire pour donner à l'utilisateur l'impression que l'écran défile en permanence.

En pratique, cette procédure n'apporte pas entière satisfaction parce que le déplacement des caractères dans la RAM prend trop de temps. L'utilisateur s'en aperçoit très vite

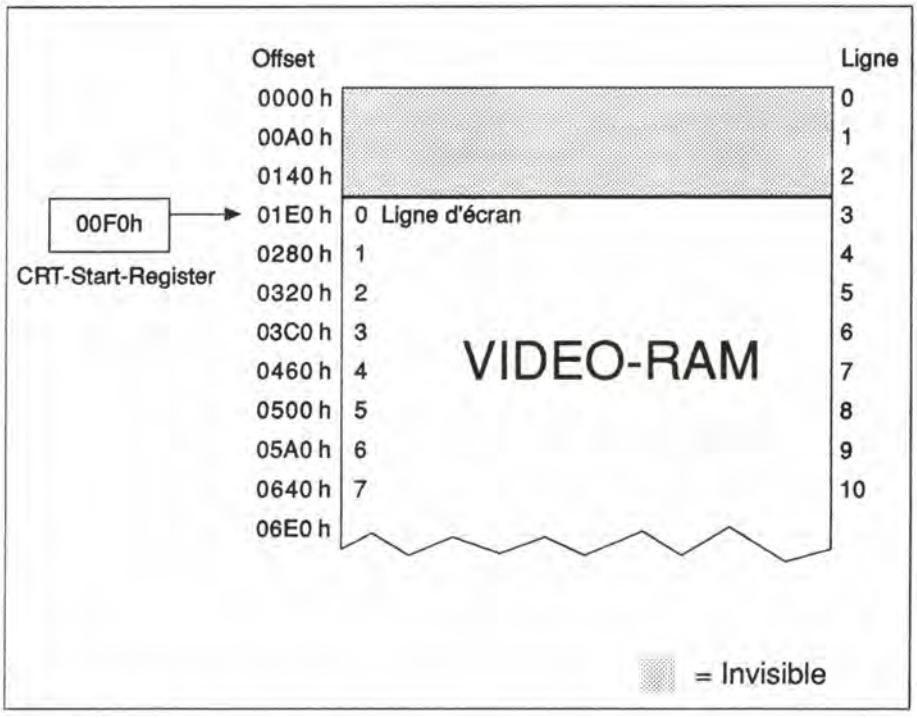
surtout lorsqu'on ne travaille pas sur deux pages écran différentes dont on traite toujours une en arrière-plan avant de la rendre visible.

On préfère par conséquent une autre technique qui permet de sélectionner librement l'adresse de départ de l'écran et modifier la longueur de ligne dans la RAM vidéo.

Déplacement de l'adresse de départ de l'écran dans la RAM vidéo et modification de la longueur de ligne

Pour décaler le contenu de l'écran d'une ligne vers le haut en mode texte, on peut tout simplement décaler tout le contenu de la RAM vidéo ou la page de texte affichée de 160 octets vers le haut, ce qui correspond à la longueur d'une ligne en mode texte 80*25 caractères. On peut également incrémenter l'adresse de départ de la page écran actuelle de 160 octets ce qui oblige le contrôleur CRT à commencer directement à la seconde ligne écran en cours lors de la prochaine configuration de l'écran. Du point de vue de l'utilisateur, le résultat est identique dans les deux cas, mais du point de vue du programme concerné, il se passe deux choses complètement différentes.

Au lieu de recopier en effet la RAM vidéo, on reprogramme un des registres du contrôleur CRT destiné à recevoir l'adresse de départ de la page écran actuelle et responsable également de la commutation entre les diverses pages écran. Cette procédure n'est pas seulement rapide mais permet de conserver la ligne écran résultant du déplacement dans la RAM vidéo pour qu'elle puisse être réinsérée par la suite.



Déplacement du point de départ de l'écran à travers l'adresse de départ

Le déplacement de l'adresse de départ de la RAM vidéo est similaire au déplacement du curseur dans un programme de traitement de texte où un extrait de texte peut être lu dans la fenêtre de texte visible. Ici, les touches du curseur sont remplacées par le registre de départ CRT et le texte par le contenu de la RAM vidéo qui est rendu visible au niveau de l'écran.

Concrètement, il s'agit des registres CRT 0Ch et 0Dh qui contiennent l'adresse de départ de la portion visible de la RAM vidéo. Ces registres ne sont pas adressables directement à travers le BIOS. Il faut donc avoir recours au contrôleur CRT en faisant attention à deux circonstances. Premièrement, l'adresse de départ est stockée sous la forme d'une adresse d'offset 16 bits dont l'octet de poids fort se trouve dans le registre 0Ch et l'octet de poids faible dans le registre 0Dh. Eu égard à l'organisation des mots dans la mémoire, l'ordre entre l'octet de poids faible et l'octet de poids fort est inversé puisque dans ce cas l'octet de poids fort précède l'octet de poids faible. Deuxièmement, l'offset étant calculé en mots et non en octets, il faut le diviser par deux avant de l'inscrire dans les registres.

Sur le plan vertical, la portion visible de l'écran peut aisément être décalée d'une douzaine, voire d'une centaine de lignes à travers la combinaison entre Pel Panning et le déplacement du point de départ de l'écran. Sur le plan horizontal, ce mécanisme révèle des défauts inattendus. A la suite du déplacement de l'écran d'un caractère vers

la gauche à l'aide du registre Pel Panning horizontal, on doit faire face à une situation fâcheuse lorsqu'on souhaite décaler également le point de départ de l'écran d'un caractère vers la gauche en incrémentant l'adresse de départ pour commencer à l'adresse d'offset 0002h et non 0000h.

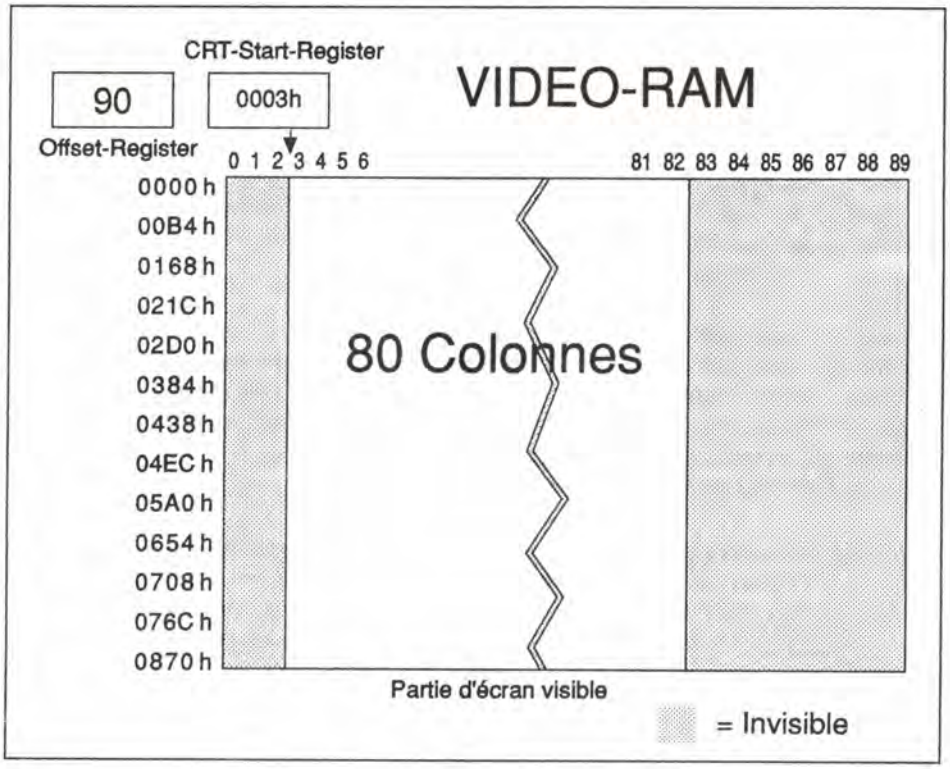
Dans le coin supérieur gauche de l'écran, on aperçoit en effet le caractère qui était affiché dans la seconde colonne de la première ligne. La conséquence est que tous les caractères de la première colonne se retrouvent dans la dernière colonne de la ligne précédente. Ce phénomène est désigné par Wrap Around.

En raison de l'organisation de la structure de la RAM vidéo, ce problème ne peut pas être résolu à l'aide du registre de départ CRT. Il faut donc recopier le contenu de la RAM vidéo. Mais on peut se décharger de cette tâche grâce à un registre du contrôleur CRT qui est en l'occurrence le registre d'offset. Il reçoit la longueur d'une ligne dans la RAM vidéo et conserve normalement la valeur 80 en mode texte parce que 80 mots et par conséquent 160 octets par ligne sont nécessaires.

Si on augmente cette valeur de 1, on n'aperçoit pas pour autant davantage de caractères par ligne sur l'écran mais le compteur d'adresse interne du contrôleur CRT se voit incrémenté d'une plus grande valeur par ligne. Par exemple, si on entre la valeur 82 au lieu de 80 dans ce registre, chaque ligne de la RAM vidéo se compose alors de 82 caractères dont seulement 80 sont affichés sur l'écran. On peut néanmoins lire les caractères restants sur la portion visible de l'écran en augmentant la valeur du registre de départ CRT de 1 ou 2. Mais cela fait disparaître les caractères de la marge gauche de l'écran.

La largeur du registre offset portant le numéro 13h à l'intérieur du contrôleur CRT étant de 8 bits, on peut étirer au maximum une ligne jusqu'à 255 caractères dans la RAM vidéo. Dans la RAM vidéo, elle occupe alors 510 octets. Il faut faire attention à cela au moment d'inscrire des caractères dans la RAM vidéo destinés à être affichés par la suite sur l'écran.

Jusqu'à présent, on devait toujours multiplier par 160 la ligne spécifiée pour calculer l'adresse d'offset d'un caractère dans la RAM vidéo. Désormais, ce facteur est porté à 510, 320 ou toujours davantage. Dans tous les cas, il correspond au double du nombre de caractères à stocker dans une ligne dans la RAM vidéo.



Déplacement de la portion visible de l'écran en élargissant les lignes à l'aide du registre offset

Si vous définissez une chasse interne plus élevée à l'aide du registre offset, le déplacement horizontal de la portion visible de l'écran subit des modifications résultant de la combinaison entre le registre Pel Panning et le point de départ de l'écran. Dans ce cas - et cela vaut également pour le déplacement vertical - il faut penser à la synchronisation des événements sinon la théorie ne correspond plus à la pratique.

Synchronisation du déplacement avec le contrôleur CRT

Si on cherche à mettre de l'ordre entre le déplacement de la portion visible de l'écran à travers le registre Pel Panning et le point de départ de l'écran, il ne faut pas oublier de tenir compte du contrôleur CRT qui joue un rôle important.

Tout d'abord, il faut toujours adresser le registre Pel Panning pendant le retour vertical du rayon électronique, autrement dit quand aucune information d'image n'est en cours de réalisation. Il faut s'assurer que la modification de ce registre ne change pas la portion déjà configurée de l'écran pour lequel les registres modifiés ont été déjà pris en compte.

Si le registre Pel Panning et le registre de départ CRT doivent être programmés simultanément pour rétablir par exemple les paramètres par défaut dans le registre Pel Panning tout en déplaçant le point de départ de l'écran, cela est interdit. Alors que les contenus modifiés des registres Pel Panning entrent en service pour la prochaine configuration de l'écran, il est généralement trop tard pour le registre de départ CRT.

Cela s'explique par le fait que le début du retour vertical est communiqué avec un peu de retard au programme même s'il s'agit là de quelques secondes ou de millièmes de secondes. Cela dépend de la consultation du retour vertical qui s'effectue normalement à travers le bit 3 dans le registre Input Status. Il indique l'état du retour vertical et contient toujours la valeur 1 pendant ce retour, sinon 0. Le retard non souhaité est dû au fait que ce registre doit être constamment consulté dans une boucle d'autant plus que plusieurs instructions de langage machine sont impliquées dans cette consultation. Leur exécution nécessite naturellement un temps relativement long. Cela est encore plus gênant lorsqu'on programme en langage évolué où il n'est pas possible de créer des codes aussi optimum que ceux offerts par la programmation directe en Assembleur.

Il suffit qu'un programme rencontre un retour vertical pour qu'il s'écoule un millième de seconde qui permet déjà au contrôleur CRT de charger l'adresse d'offset pour la prochaine configuration de l'écran à partir du registre de départ CRT. Il ne faut plus espérer pouvoir modifier ce registre lors de la prochaine configuration de l'écran mais il faut attendre la troisième.

Comme les modifications apportées aux registres Pel Panning ont été prises en compte lors de la prochaine configuration de l'écran, il se crée ici des dissonances qui ne laissent pas indifférent l'utilisateur. Il est donc indispensable de suivre une méthode concise lors de la modification simultanée des registres Pel Panning et du registre de départ CRT.

A travers le registre Input Status 1, on doit attendre le début puis la fin d'un retour vertical. Le nouveau point de départ de l'écran sera ensuite chargé dans les registres qui conviennent du contrôleur CRT. Cela ne nuit pas à la configuration de l'écran parce que le contrôleur CRT a déjà chargé l'adresse de départ dans son compteur d'adresse interne dès le début de la nouvelle configuration de l'écran et ne doit plus y accéder.

On attend ensuite la fin de la configuration de l'écran et par conséquent le début du retour vertical. Dès que cet événement a eu lieu, on doit immédiatement programmer les registres Pel Panning dont la modification montre les effets de la configuration de l'écran qui vient de commencer. Cela vaut également pour la nouvelle adresse de départ dans le registre de départ CRT qui est sûre d'être chargée dans le compteur d'adresse interne avant la nouvelle configuration de l'écran.

Le fait d'attendre le retour vertical du rayon électronique a pour conséquence de synchroniser l'exécution du programme indépendamment de la vitesse de l'ordinateur avec la fréquence d'image de la carte vidéo. Cela est un atout majeur dans le domaine des jeux. Si vous appelez toujours successivement la routine de définition des registres Pel Panning et celle du registre de départ CRT, vous pouvez alors décaler la portion visible de l'écran d'un ou plusieurs points à chaque configuration. Selon la carte et le

mode vidéo que vous utilisez, vous pouvez obtenir entre 50 et 70 déplacements par seconde. L'oeil humain n'étant pas apte de capter cela, les déplacements à travers des points multiples garantissent en même temps un effet de mouvement continu.

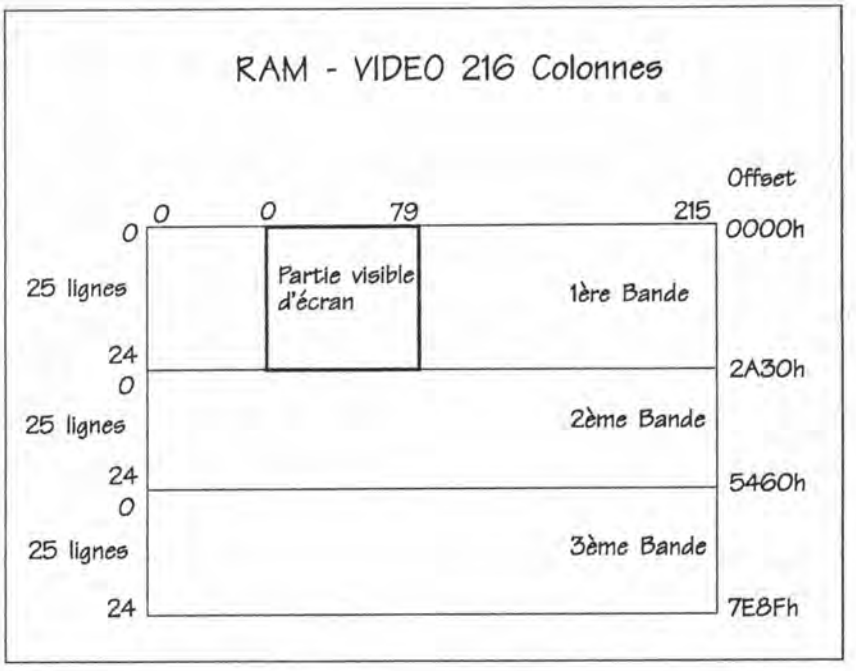
Texte continu en mode texte

Pour illustrer la programmation combinée des registres Pel Panning et du registre de départ CRT, nous vous proposons un programme permettant de faire défiler un texte écrit en grandes lettres depuis la droite vers la gauche de l'écran. Compte tenu de la fonction qu'ils doivent remplir, les deux programmes d'exemple s'appellent CONTC.C et CONTP.PAS.

Deux routines appelées ShowContText et SetOrigin jouent le rôle principal dans les deux programmes. ShowContText a pour tâche de recevoir le texte à afficher et l'installer dans la RAM vidéo à l'insu de l'utilisateur. Pour pouvoir utiliser la totalité de la RAM vidéo de 32 Ko, les deux programmes la divisent en trois bandes sur le plan interne ayant chacune une largeur de 216 caractères et une hauteur de 25 lignes. Chaque bande occupe 10800 octets dans la RAM vidéo ce qui donne en tout 32400 octets dont seulement 368 octets sont utilisés.

Après avoir logé le texte dans la RAM vidéo, ShowContText commence à l'afficher à l'aide de la procédure SetOrigin. Elle sert à spécifier le point de départ actuel de l'écran avec comme obligation d'indiquer la bande à afficher, la colonne et la ligne de départ ainsi que le déplacement du point. Les autres paramètres entrés dans les registres Pel Panning restent inchangés. Quant au numéro de la bande, il est lié aux colonne et ligne de départ d'une adresse de départ de la portion visible de l'écran et inscrit dans le registre de départ CRT.

Pour donner à l'utilisateur l'impression que le texte est en train de défiler, SetOrigin est appelée avec une valeur de ligne et colonne constante à l'intérieur de ShowContText en incrémentant progressivement le compteur de pixels pour le déplacement horizontal. Lorsque le déplacement maximal d'un caractère est atteint, le compteur de pixels horizontal est ramené à 0 ou 8 et seule la colonne de départ est incrémentée.



Sectionnement de la RAM vidéo à l'intérieur de CONTP.PAS et CONTC.C

Le déplacement peut s'effectuer à des vitesses variables et la différence de chasse entre les cartes EGA et VGA est prise en compte. A cet effet, ShowContText attend deux paramètres en dehors du texte à afficher. Ils indiquent la vitesse souhaitée et le type de la carte vidéo installée. Pour la vitesse, vous pouvez utiliser les constantes FAST, MEDIUM et SLOW et pour le type de la carte vidéo les deux constantes EGA et VGA.

Lors du déplacement du point de départ de l'écran à l'intérieur de ShowContText, les deux constantes sont formulées sous forme d'indice dans le tableau SetTable déclaré dans cette routine. Ici, les valeurs sont stockées séparément pour les cartes EGA et VGA et en tenant compte de la vitesse spécifiée. SetOrigin s'en sert successivement pour définir le compteur de pixels horizontal. La fin d'une telle série est indiquée dans le tableau par la valeur 255 qui signale qu'il faut maintenant poursuivre avec la colonne suivante de l'écran.

Pour varier la vitesse d'affichage du texte continu, on utilise des largeurs de pas différentes dans le compteur de pixels horizontal. En mode SLOW, il fait défiler chaque point avec un intervalle de 0 à 7 ou 8 à 7. En mode MEDIUM, l'action se déroule tous les deux points ce qui permet de faire défiler une distance double dans le même laps de temps. La vitesse augmente davantage en mode FAST où deux points supplémentaires sont parcourus à l'intérieur de chaque point.

L'incrémentation de la colonne de départ atteint sa limite lorsque la dernière colonne d'une bande, la colonne 216, apparaît sur l'écran. Il faut passer alors à la bande suivante et reprendre à partir de sa première colonne. Le contenu du dernier écran ne défilant plus vers la gauche dans la bande précédente, il faut recommencer avec le même caractère au début de la bande suivante qui se trouvait dans le dernier écran de la bande précédente. Ce n'est qu'ainsi qu'on peut créer un parcours entre les diverses bandes sans que l'utilisateur s'en aperçoive.

Le nombre de caractères pouvant être affichés et captés par l'utilisateur dépend essentiellement de la chasse. Une routine appelée PrintChar est responsable de la construction des caractères dans la RAM vidéo. Outre le caractère, elle attend comme arguments la colonne et la bande où doit apparaître le caractère. La ligne est spécifiée à l'aide d'une constante (STARTZ) et ne doit donc pas être transmise. Cette routine lit le modèle de points des caractères dans le jeu 8*14 qui est stocké dans une structure ROM des cartes EGA et VGA. Ce jeu peut être lu une fois que son adresse est communiquée à l'aide de la sous-fonction 30h de la fonction 11h de l'interruption vidéo du BIOS.

Cette routine existe aussi par le fait que la version C du programme ContText ne fonctionne pas sans l'aide d'un petit module en Assembleur portant le nom CONTCA.ASM. Comme la fonction BIOS citée ci-dessus retourne surtout des informations dans le registre BP, les fonctions normales C ne conviennent pas pour l'appel des interruptions et doivent donc être remplacées par une routine d'appoint.

PrintChar sort sur l'écran la matrice 8*14 points des caractères en reproduisant chaque point de la matrice sur une position d'écran. Chaque caractère se compose ainsi de 8 colonnes et 14 lignes. Une bande peut contenir 27 caractères (216/8) dont 10 (80/8) s'affichent simultanément. Sachant que le dernier écran de la bande précédente doit être répété dans les deuxième et troisième bandes, on ne peut y insérer que 17 caractères nouveaux au lieu de 27. Cette méthode permet par conséquent de stocker 61 (27 + 17 + 17) caractères dans la RAM vidéo et de les afficher.

Listing : CONTP.PAS

```
{*****************************************************************
*                          C O N T P                             *
*----------------------------------------------------------------*
*  Fonction : montre comment programmer un défilement continu de texte *
*             avec une carte EGA ou VGA                          *
*----------------------------------------------------------------*
*  Auteur           : MICHAEL TISCHER                            *
*  Développé le     : 23.01.1990                                 *
*  Dernière MAJ     : 14.01.1991                                 *
*****************************************************************}

program TextContinu;

uses dos,                                           { Inclut les unités }
     crt;

const SLOW      = 1;            { Valeurs de SPEED pour ShowDefilText() }
      MEDIUM    = 2;
      FAST      = 3;

const COULEUR   = $5E;                        { Jaune sur fond lilas }
      COULEUR1  = $5F;                        { Blanc sur fond lilas }
      LARGEUR   = 8;              { Largeur des caractères en pixels }
      HAUTEUR   = 14;    { Hauteur des caractères en lignes de trame }
      COLONNES  = 216;      { Nb colonnes par ligne en mémoire écran }
      BANDSIZE  = 10800;                      { Taille d'une bande }
      BANDES    = 3;                          { Nombre de bandes }
      MAXLEN    = 61;                 { Nombre maximal de caractères }
      STARTL    = 6;  { Ligne où commencent les caractères sur écran }

      CrtAttr   = $3C0;          { Registre du contrôleur d'attribut }
      CrtStatus = $3da;                              { Port d'état }
      CrtAdr    = $3d4;                { Port d'adresses du moniteur }


type VRAM = array[1..BANDES,1..25,1..COLONNES] of word;{ Mémoire écran }
     VPTR = ^VRAM;                     { Pointeur sur la mémoire d'écran }

var vp          : vptr;                 { Pointe sur la mémoire d'écran }

type CARTE = ( EGA, VGA, NINI );                  { Type de carte vidéo }

procedure CLI; inline( $FA );                { Inhibe les interruptions }
procedure STI; inline( $FB );               { Rétablit les interruptions }

{*****************************************************************
*  SetOrigin : Fixe la partie visible de la mémoire d'écran par  *
*              programmation du contrôleur vidéo                 *
*----------------------------------------------------------------*
*  Entrées :   Band    = Numéro de la bande à afficher  (1-5)    *
*              Colonne = Numéro de la colonne et de la ligne     *
*              Ligne     affichées en haut à gauche (Origine= 0,0) *
*              PixX,   = Offset en pixels                        *
*              PixY                                              *
*****************************************************************}

procedure SetOrigin( band, colonne, ligne, pixx, pixy : byte );

var offset : integer;           { Offset de début de la mémoire d'écran }
ch : char;

begin
  offset := ( BANDSIZE div 2 ) * (band-1) + ligne * COLONNES + colonne;

        (-- Attend un retour vertical du faisceau puis sa fin ---------)

  repeat until port[CrtStatus] and 8 = 8;
  repeat until port[CrtStatus] and 8 = 0;

( Ecrit dans les registres $0C et $0D l'offset de début de mém. écran  )
(  N'est pris en compte qu'après le prochain retour faisceau vertical  )

  CLI;                                          { Inhibe les interruptions }
  portw[ CrtAdr ] := hi( offset ) shl 8 + $0c;
  portw[ CrtAdr ] := lo( offset ) shl 8 + $0d;
  STI;

    (-- Attend le début du retour vertical du faisceau pour fixer le --)
(-- nouvel offset en pixels et la nouvelle adresse de début d'écran --)

  repeat until port[CrtStatus] and 8 = 8;

      (-- Ecrit l'offset en pixels dans les registres $08 / $13 du ---)
                                        (-- contrôleur d'attribut ---)

  CLI;
  portw[ CrtAdr ] := pixy shl 8 + $08;
  port[ CrtAttr ] := $13 or $20;          { N'accède que octet par octet }
  port[ CrtAttr ] := pixx;                   { au contrôleur d'attribut }
  STI;                                     { Rétablit les interruptions }

                                                      { ch := readkey; }
end;

{*****************************************************************
*  PrintChar : Ecrit un caractère en dehors de la zone visible de *
*              la mémoire d'écran                                *
*----------------------------------------------------------------*
*  Entrées :   Caractere= Caractère à écrire                     *
*              Band     = Numéro de la bande (0-4)               *
*              Colonne = Colonne en mémoire d'écran où doit commencer *
*                        le caractère                            *
*  Info    :   Le caractère écrit ne peut être rendu visible que par *
*              défilement continu de l'écran                     *
*              Le motif binaire du caractère est tiré du jeu de  *
*              caractères 14*8 pixels de la ROM EGA ou VGA       *
*****************************************************************}

procedure PrintChar( caractere : char; band, colonne : byte );

type CARDEF = array[0..255,1..14] of byte;  { Struct. du jeu de caract }
     CARPTR = ^CARDEF;                  { Pointe sur le jeu de caractères }

var  Regs  : Registers;        { Registres pour gérer les interruptions }
     ch    : char;                                  { Pixel du caractère }
     i, k,                                      { Compteurs d'itérations }
     masque : byte;          { Masque binaire pour dessiner le caractère }

const fptr : CARPTR = NIL;     { Pointeur sur jeu de caractères en ROM }

begin
  if fptr = NIL then          { Le pointeur a-t-il déjà été déterminé ? }
    begin                                                      { Non }
      Regs.AH := $11;                      { Appelle l'option $30 de      }
      Regs.AL := $30;                      { l'interruption $11 du BIOS   }
      Regs.BH := 2;                 { Renvoie un pointeur sur le jeu  8*14 }
      intr( $10, Regs );
      fptr := ptr( Regs.ES, Regs.BP );             { Compose le pointeur }
    end;

    (-- Parcourt et dessine le caractère ligne par ligne ------------)

  for i := 1 to HAUTEUR do
    begin
      masque := fptr^[ord(caractere),i]; { Motif binaire pour une ligne }
      for k := 1 to LARGEUR do     { Passe en revue les différents bits }
        begin
          if masque and 128 = 128 then ch := #219   { Dessine un pixel }
                                  else ch := #32;    { Efface un pixel }
          vp^[band, STARTL+i, (colonne-1)*LARGEUR+k] :=
                                                 ord(ch) + COULEUR shl 8;
          masque := masque shl 1;                { Passe au bit suivant }
        end;
    end;
end;

{*****************************************************************
*  IsEgaVga : Teste la présence d'une carte EGA ou VGA           *
*----------------------------------------------------------------*
*  Entrée  : néant                                               *
*  Sortie  : EGA, VGA ou NINI                                    *
*****************************************************************}

function IsEgaVga : CARTE;

var Regs : Registers;          { Registres pour gérer les interruptions }

begin
  Regs.AX := $1a00;                { La fonction 1Ah n'existe qu'en VGA }
  Intr( $10, Regs );
  if ( Regs.AL = $1a ) then          { La fonction est-elle disponible }
    IsEgaVga := VGA
  else
    begin
      Regs.ah := $12;                        { Appelle l'option $10 de   }
      Regs.bl := $10;                                { la fonction $12 }
      intr($10, Regs);                          { Interruption vidéo }
      if ( Regs.bl <> $10 ) then IsEgaVga := EGA
                            else IsEgaVga := NINI;
    end;
end;

{*****************************************************************
*  ShowDefilText : Fait défiler un texte sur l'écran             *
```

```
{**************************************************************}
{*  Entrées : DTEXT = Texte à faire défiler sous forme de chaîne  *}
{*            SPEED = Vitesse de défilement (Constantes SLOW, FAST etc.)*}
{*            VC    = Type de carte vidéo (EGA ou VGA)          *}
{**************************************************************}

procedure ShowDefilText( dtext : string; speed : byte; vc : carte );

var band,                                          { Bande courante }
    colonne,                                       { Colonne courante }
    index,                 { Indice courant dans la chaîne de caractères }
    len,                                  { Longueur du texte à afficher }
    i, k : integer;                                { Compteurs d'itérations }
    step,                                            { Pas de défilement }
    uplimit : byte;                                 { Nombre d'itérations }
    Regs : Registers;                 { Registres pour les interruptions }

const steptable : array [EGA..VGA,1..3,1..10] of byte =
      (
        (                                        { Pas de défilement pour EGA }
          (  0,  1,  2,  3,  4,  5,  6,  7, 255, 255 ),
          (  0,  2,  4,  6, 255, 255, 255, 255, 255, 255 ),
          (  0,  4, 255, 255, 255, 255, 255, 255, 255, 255 )
        ),
        (                                        { Pas de défilement pour VGA }
          (  8,  0,  1,  2,  3,  4,  5,  6,  7, 255 ),
          (  8,  2,  5, 255, 255, 255, 255, 255, 255, 255 ),
          (  8,  3, 255, 255, 255, 255, 255, 255, 255, 255 )
        )
      );

begin
  vp := ptr( $B800, $0000 );              { Pointe sur la mémoire d'écran }

{-- Remplit toute la mémoire d'écran avec des espaces --}

  for index := 1 to BANDES do
    for i := 1 to 25 do
      for k := 1 to COLONNES do
        vp^[ index, i, k ] := COULEUR shl 8 + 32;

      {-- Trace des guides horizontaux ------------------------------}

  for k := 1 to BANDES do
    for i := 1 to COLONNES do
      begin
        vp^[ k, STARTL-2, i ] := ord('') + COULEUR1 shl 8;
        vp^[ k, STARTL + HAUTEUR + 2, i ] := ord('') + COULEUR1 shl 8;
      end;

  gotoxy( 1, 1 );                    { Le curseur disparaît au défilement }

        {-- Fixe la couleur du cadre d'écran ------------------------}

  Regs.AH := $10; { Numéro de la fonction "Fixer la couleur du cadre "}
  Regs.AL := $01;                                    { Numéro d'option }
  Regs.BH := COULEUR shr 4;                          { Couleur du cadre }
  intr( $10, Regs );

     {-- Fixe le nombre de colonnes par ligne dans la mémoire écran --}

  portw[ CrtAdr ] := ( COLONNES div 2 ) shl 8 + $13;

          {-- Ecrit le texte défilant dans la mémoire d'écran -------}

  if length( dtext ) > MAXLEN then len := MAXLEN
                              else len := length ( dtext );
  colonne := 1;
  band    := 1;
  index   := 1;
  while ( index <= len ) do
    begin
      PrintChar( dtext[index], band, colonne ); { Dessine le caractère }
      inc( colonne );                              { Colonne suivante  }
      inc( index );                                { Caractère suivant }
      if ( colonne > ( COLONNES div LARGEUR ) ) then  { Change bande ? }
        begin                                                  { Oui }
          colonne := 1;                              { revient en colonne 1 }
          inc( band );                          { Passe à la bande suivante }
          dec( index, ( 80 div LARGEUR ) );       { Une page en arrière }
        end
    end;

        {-- Fait défiler le texte de droite à gauche sur l'écran - --}

  colonne := 0;                          { Commence en colonne 0 et bande 1 }
  band    := 1;
  for i := 1 to (len-( 80 div LARGEUR )) * LARGEUR do
    begin
      k := 1;
      while ( steptable[vc, speed, k] <> 255 ) do
        begin
          SetOrigin( band, colonne, 0, steptable[vc, speed, k], 0 );
          inc( k );
        end;

      inc( colonne );                                  { Colonne suivante ? }
      if ( colonne = COLONNES - 80 ) then      { Changement de bande ? }
        begin                                                  { Oui }
          colonne := 0;                          { Recommence en colonne 0 }
          inc( band );                             { Incrémente la bande }
        end
    end;

  {-- Remet 80 caractères par ligne en mémoire d'écran    ------------}

  portw[ CrtAdr ] := 40 shl 8 + $13;

                  {-- Rétablit le cadre d'écran ---------------------}

  Regs.AH := $10;   { Numéro de la fonction "Fixer la couleur du cadre"}
  Regs.AL := $01;                                    { Numéro d'option }
  Regs.BH := 0;                                      { Cadre en noir }
  intr( $10, Regs );

  if ( vc = EGA ) then                       { Paramétrage par défaut }
    SetOrigin( 1, 0, 0, 0, 0 )                                { pour EGA }
  else
    SetOrigin( 1, 0, 0, 8, 0 );                               { pour VGA }
  ClrScr;
end;

{-- PROGRAMME PRINCIPAL }

var  ch : char; i : integer;
     vc : carte;

begin
  vc := IsEgaVga;                    { Détermine le type de carte vidéo }
  if vc = EGA then halt;
  if ( vc = MINI ) then
    begin
      writeln( 'TEXTE CONTINU - (c) 1990 by MICHAEL TISCHER'#13#10 );
      writeln( 'Attention: aucune carte EGA ou VGA n''est installée !');
    end
  else
    ShowDefilText('+++ BIBLE PC (c) 1987-1991 par Micro Application +++'
                  + '        ', FAST, vc );
  while keypressed do
    ch := readkey;
end.
```

Listing : CONTC.C

```
/*****************************************************************
*                          C O N T C . C                         *
*----------------------------------------------------------------*
* Fonction        : montre comment programmer un défilement continu *
*                   de texte ( Smooth Scrolling) avec une carte EGA *
*                   ou VGA                                       *
*----------------------------------------------------------------*
* Auteur          : MICHAEL TISCHER                              *
* Développé le    : 26.08.1990                                   *
* Dernière MAJ    : 14.02.1992                                   *
*----------------------------------------------------------------*
* (MICROSOFT C)                                                  *
* Compilation     : CL /AS Contc.c Contca;                       *
*----------------------------------------------------------------*
* (BORLAND TURBO C)                                              *
* Compilation     : Utiliser un fichier de projet avec le contenu *
*                   suivant                                      *
*                     Contc.c                                    *
*                     Contca.obj                                 *
*----------------------------------------------------------------*
* Appel           : Contc                                        *
*****************************************************************/

#include <dos.h>                                    /* Fichiers en-tête */
#include <stdarg.h>
#include <string.h>
#include <stdio.h>

#ifdef __TURBOC__                          /* Compilation avec Turbo C? */
  #define CLI()           disable()
  #define STI()           enable()
  #define outpw( p, w )   outport( p, w )
  #ifndef inp
    #define outp( p, b )  outportb( p, b )
    #define inp( p )      inportb( p )
  #endif
#else                                      /* Non, avec QuickC 2.0 ou MSC */
  #include <conio.h>
  #define MK_FP(seg,ofs) ((void far *)\
                         (((unsigned long)(seg) << 16) | (ofs)))
  #define CLI()           _disable()
  #define STI()           _enable()
#endif

#define FAST        2       /* Valeurs de SPEED dans ShowContText() */
#define MEDIUM      1
#define SLOW        0

#define COULEUR     0x5E                        /* Jaune sur fond lilas */
#define COULEUR1    0x5F                        /* Blanc sur fond lilas */
#define LARGEUR     8           /* Largeur des caractères en pixels */
#define HAUTEUR     14   /* Hauteur des caractères (lignes de trame) */
#define COLONNES    216                /* Nb de caractères à l'écran */
#define BANDSIZE    10800                    /* Taille d'une bande */
#define BANDES      3                        /* Nombre de bandes */
#define MAXLEN      61              /* Nombre maximal de caractères */
#define STARTL      5    /* Ligne de départ des caractères à l'écran */

#define CrtAttr     0x3C0       /* Registre du contrôleur d'attribut */
#define CrtStatus   0x3DA                       /* Port d'état */
#define CrtAdr      0x3D4           /* Port d'adresses du moniteur */

#define TRUE        (0 == 0)
#define FALSE       (0 == 1)

#define EGA         0                           /* Types de cartes */
#define VGA         1
#define NINI        2

typedef unsigned char BYTE;                 /* Bricolage d'un type BYTE */
typedef unsigned int  WORD;                      /* idem pour  WORD */
typedef BYTE          BOOL;

typedef WORD   VRAM[BANDES][25][COLONNES];     /* Structure mém écran */
typedef VRAM   far *VPTR;       /* Pointeur FAR sur la mémoire d'écran */

typedef BYTE   CARDEF[256][14];     /* Structure du jeu de caractères */
typedef CARDEF far *CARPTR;         /* Pointeur sur jeu de caractères */

/*-- Fonctions externes ------------------------------------------*/

extern void    far *getfontptr( void );     /* Fonction en assembleur */

/*-- Variables globales ------------------------------------------*/

VPTR           vp;                  /* Pointeur sur la mémoire d'écran */
```

```
/*****************************************************************
*  SetOrigin : Fixe la partie visible de la mémoire d'écran par  *
*              programmation du contrôleur vidéo                 *
*----------------------------------------------------------------*
*  Entrées :   Band    = Numéro de la bande à afficher(1-5)      *
*              Colonne,= Numéro de la colonne et de la ligne     *
*              Ligne     affichées en haut à gauche              *
*              PixX,   = Offset en pixels                        *
*              PixY                                              *
*  Sortie  :   néant                                             *
*****************************************************************/

void SetOrigin( BYTE band, BYTE colonne, BYTE ligne,
                BYTE pixx, BYTE pixy)
{
  int     offset;               /* Offset de début de la mémoire d'écran */

  offset = ( BANDSIZE >> 1) * band + ligne * COLONNES + colonne;

  /*- Attend un retour vertical du faisceau et sa fin ----------------*/

  while ( !(( inp(CrtStatus) & 8 ) == 8 ));
  while ( !(( inp(CrtStatus) & 8 ) == 0 ));

          /* Ecrit dans les registres 0x0C et 0x0D l'offset de début *
           * de la mémoire d'écran                                  *
           * N'est pris en compte qu'après le prochain retour de    *
           * balayage vertical                                      */

  CLI();                                    /* Inhibe les interruptions */
  outpw( CrtAdr, ( offset & 0xFF00 ) + 0x0C );
  outpw( CrtAdr, ( (BYTE) offset << 8 ) + 0x0D );
  STI();                                    /* Rétablit les interruptions */

        /* Attend le début du retour vertical du faisceau pour fixer  *
         * le nouvel offset en pixels et la nouvelle adresse de début *
         * de l'écran                                                 */

  while ( !(( inp( CrtStatus ) & 8 ) == 8 ));

        /* Ecrit l'offset en pixels dans les registres 0x08 / 0x13 du *
         * contrôleur d'attribut                                      */

  CLI();                                    /* Inhibe les interruptions */
  outpw( CrtAdr, ( pixy << 8 ) + 0x08 );
  (void) outp( CrtAttr, 0x13 | 0x20 );
  (void) outp( CrtAttr, pixx );
  STI();                                    /* Rétablit les interruptions */
}

/*****************************************************************
*  PrintChar : Ecrit un caractère en dehors de la zone visible de la *
*              mémoire d'écran                                   *
*----------------------------------------------------------------*
*  Entrée  :   Caractere = Caractère à écrire                    *
*              Band      = Numéro de la bande (0-4)              *
*              Colonne   = Colonne en mémoire d'écran où doit commencer*
*                          le caractère                          *
*  Info    :   Le caractère écrit ne peut être rendu visible que *
*              par défilement continu de l'écran .               *
*              Le motif binaire du caractère est tiré du jeu de  *
*              caractères 14*8 pixels de la ROM EGA ou VGA .     *
*****************************************************************/

void PrintChar( char caractere, BYTE band, BYTE colonne )
{
  char            ch;                       /* Pour stocker un pixel */
  BYTE            i, k,                     /* Compteurs d'itérations */
                  masque; /* Masque binaire pour dessiner le caractère */
  static CARPTR   fptr = (CARPTR) 0;        /* Jeu de caractères en ROM */

  if ( fptr == (CARPTR) 0 )                 /* Pointeur déjà déterminé ? */
    fptr = getfontptr();    /* Le charge avec la fonction en assembleur */

  /*- Parcourt et dessine le caractère ligne par ligne ---------------*/

  for ( i = 0; i < HAUTEUR; ++i )
  {
    masque = (*fptr)[caractere][i];     /* Motif binaire pour une ligne */
    for ( k = 0; k < LARGEUR; ++k )     /* Parcourt les différents bits */
    {
      ch = ( masque & 128 ) ? 219 : 32;
      (*vp)[band][STARTL+i][colonne*LARGEUR+k] =
                              (BYTE) ch+( COULEUR << 8 );
      masque <<= 1;                             /* Passe au bit suivant */
    }
  }
```

```
)

/*************************************************************************
*  IsEgaVga : Teste la présence d'une carte EGA ou VGA                   *
**-----------------------------------------------------------------------**
*  Entrée  : néant                                                       *
*  Sortie  : EGA, VGA ou NINI                                            *
*************************************************************************/

BYTE IsEgaVga( void )
{
  union REGS    Regs;          /* Registres pour gérer les interruptions */

  Regs.x.ax = 0x1a00;                 /* N'existe que pour les cartes VGA */
  int86( 0x10, &Regs, &Regs );
  if( Regs.h.al == 0x1a )                     /* La fonction disponible ? */
    return VGA;
  else
  {
    Regs.h.ah = 0x12;                              /* Appelle l'option 10h */
    Regs.h.bl = 0x10;                                /* de la fonction 12h */
    int86(0x10, &Regs, &Regs );                  /* Appelle le BIOS vidéo */
    return ( Regs.h.bl != 0x10 ) ? EGA : NINI;
  }
}

/*************************************************************************
*  ShowContText : Fait défiler un texte sur l'écran                      *
**-----------------------------------------------------------------------**
*  Entrée :  dtext = Texte à faire défiler sous forme de chaîne          *
*            speed = Vitesse de défilement (SLOW, MEDIUM ou SLOW)        *
*            vc    = carte vidéo (EGA ou VGA)                            *
*************************************************************************/

void ShowContText( char * dtext, BYTE speed, BYTE vc )
{
  int       band,                                    /* Bande courante */
            colonne,                                /* Colonne courante */
            index,     /* Indice courant dans la chaîne de caractères */
            len,                          /* Longueur du texte à afficher */
            i, k,                            /* Compteur d'itérations */
            pixx;                          /* Valeur de panning horizontal */
  WORD far  *wptr;      /* Pointeur pour parcourir la mémoire d'écran */
  union REGS Regs;              /* Registres pour gérer l'interruption */

  static BYTE steptable[2][3][10] =
        {
          {                                 /* Pas de défilement pour EGA */
            {  0,  1,  2,  3,  4,  5,  6,  7, 255, 255 },
            {  0,  2,  4,  6, 255, 255, 255, 255, 255, 255 },
            {  0,  4, 255, 255, 255, 255, 255, 255, 255, 255 }
          },
          {                                 /* Pas de défilement pour VGA */
            {  8,  0,  1,  2,  3,  4,  5,  6,  7, 255 },
            {  8,  2,  5, 255, 255, 255, 255, 255, 255, 255 },
            {  8,  3, 255, 255, 255, 255, 255, 255, 255, 255 }
          }
        };

  vp = MK_FP( 0xB800, 0x0000 );          /* Pointe sur la mémoire d'écran */

  /*- Remplit toute la mémoire d'écran avec des espaces --------------*/

  for( index = 0; index < BANDES; ++index )
    for( i = 0; i < 25; ++i )
      for( k = 0; k < COLONNES; ++k )
        (*vp)[index][ i ][ k ] = ( COULEUR << 8 ) + 32;

  /*- Trace des guides horizontaux -----------------------------------*/

  for( k = 0; k < BANDES; ++k )
    for( i = 0; i < COLONNES; ++i )
    {
        (*vp)[ k ][ STARTL-2 ][ i ] =
                    (BYTE) '' + ( COULEUR1 << 8 );
        (*vp)[ k ][ STARTL + HAUTEUR + 2][ i ] =
                    (BYTE) '' + ( COULEUR1 << 8 );
    }

  /*- Retire le curseur clignotant -----------------------------------*/

  Regs.h.ah = 0x02;   /* Numéro de la fonction "Set Cursor"  */
  Regs.h.bh = 0;     /* Page d'écran */
  Regs.x.dx = 0;       /* Coordonnées */
  int86( 0x10, &Regs, &Regs );

  /*- Fixe la couleur du cadre d'écran -------------------------------*/

  Regs.h.ah = 0x10;       /* Numéro de la fonction "Fixer couleur cadre"*/
  Regs.h.al = 0x01;                                 /* Numéro d'option */
  Regs.h.bh = COULEUR >> 4;                         /* Couleur du cadre */
  int86( 0x10, &Regs, &Regs );

  /*- Fixe à COLONNES le nbre de colonnes/ligne dans la mémoire écran -*/

  outpw( CrtAdr, ( ( COLONNES >> 1 ) << 8 ) + 0x13 );

  /*-- Ecrit le texte défilant dans la mémoire d'écran ----------------*/

  if ( ( len = strlen( dtext ) ) > MAXLEN )     /* Chaîne trop longue ? */
    *(dtext + ( len = MAXLEN )) = '\0';                 /* Oui, on abrège */

  for( colonne = band = index = 0; index < len; )
  {                                                  /* Ecrit les caractères */
      PrintChar( *(dtext+index++), band, colonne++ );
      if ( colonne >= COLONNES / LARGEUR )     /* Changement de bande ? */
      {                                                          /* Oui */
          colonne = 0;                          /* Recommence en colonne 1 */
          ++band;                              /* Passe à la bande suivante */
          index -= 80 / LARGEUR;                  /* Une page en arrière */
      }
  }

  /*-- Fait défiler le texte de droite à gauche sur l'écran -----------*/

  for( colonne = band = 0 , i = (len - ( 80/LARGEUR )) * LARGEUR;
       i > 0;
       --i )
  {
    for( k = 0; ( pixx = steptable[vc][speed][k]) != 255 ; ++k )
      SetOrigin( band, colonne, 0, pixx, 0 );

    if( ++colonne == COLONNES - 80 )            /* Changement de bande */
    {                                                          /* Oui*/
      colonne = 0;                            /* Recommence en colonne 0 */
      ++band;                                  /* Incrémente la bande */
    }
  }

  /*- Remet 80 caractères par ligne en mémoire d'écran ---------------*/

  outpw( CrtAdr, ( 40 << 8 ) + 0x13 );

  SetOrigin( 0, 0, 0, 8, 0 );                    /* Paramétrage par défaut */

  /*- Rétablit le curseur --------------------------------------------*/

  Regs.h.ah = 0x02;                /* Numéro de la fonction "Set Cursor" */
  Regs.h.bh = 0;                                        /* Page d'écran */
  Regs.x.dx = 0;                                        /* Coordonnées */
  int86( 0x10, &Regs, &Regs );

  /*- Restaure la couleur du cadre d'écran ---------------------------*/

  Regs.h.ah = 0x10;              /* "Fixer la couleur du cadre d'écran" */
  Regs.h.al = 0x01;                                 /* Numéro de l'option */
  Regs.h.bh = 0;                                        /* Cadre en noir */
  int86( 0x10, &Regs, &Regs );

  /*-- Efface l'écran -------------------------------------------------*/

  for( wptr = (WORD far *) vp, i = 80*25; i-- ; )
    *wptr++ = 0x0720;
}

/*-------------------------------------------------------------------*/
/*--                        PROGRAMME PRINCIPAL                    --*/
/*-------------------------------------------------------------------*/

void main( void )
{
  BYTE     vc;                          /* Type de carte vidéo installée */

  if( ( vc = IsEgaVga() ) == NINI )
    printf( "TEXTE DEFILANT -  (c) 1990, 92 by MICHAEL TISCHER\n" \
            "Attention : aucune carte EGA ou VGA n'est installée !\n" );
  else
    ShowContText( "+++ La Bible PC (c) 1990-1992 Micro Application +++ "\
                  "                ", FAST, vc );
}
```

Listing : CONTCA.ASM

```
;*****************************************************************;
;*                        C O N T C A . A S M                    *;
;*---------------------------------------------------------------*;
;*  Fournit une fonction à inclure dans le programme CONTC.C     *;
;*  qui renvoie un pointeur sur le jeu de caractères 8*14        *;
;*  d'une carte EGA ou VGA                                       *;
;*---------------------------------------------------------------*;
;*   Auteur          : MICHAEL TISCHER                           *;
;*   Développé le    : 23.08.1990                                *;
;*   Dernière MAJ    : 14.02.1992                                *;
;*---------------------------------------------------------------*;
;*   Assemblage      : MASM /mx CONTCA                           *;
;*         ou        : TASM -mx LAUFTCA                          *;
;*         ... puis lier avec CONTC.C                            *;
;*****************************************************************;

IGROUP group _text                  ; Regroupe les segments de programme
DGROUP group _bss, _data            ; Regroupe les segments de données

        assume CS:IGROUP, DS:DGROUP, ES:DGROUP, SS:DGROUP

_BSS    segment word public 'BSS'   ; Segment pour toutes les variables
_BSS    ends                        ;          statiques non initialisées

_DATA   segment word public 'DATA'  ; Segment des variables globales
                                    ;          et statiques initialisées


_DATA   ends

;-- Programme ----------------------------------------------------;

_TEXT   segment byte public 'CODE'          ; Segment de programme

        public  _getfontptr                 ; Fonction accessible

;-- GETFONTPTR: Renvoie un pointeur FAR sur le jeu de caractères 8*14---
;-- Déclaration : void far * getfontptr( void ) ----------------------

_getfontptr     proc near

        push    bp                                  ; Sauvegarde BP

        mov     ax,1130h                ; Charge les registres avant
        mov     bh,2
        int     10h                     ; d'appeler l'interruption vidéo

        mov     dx,es                   ; Transfère ES:BP en DX:AX
        mov     ax,bp

        pop     bp                      ; Reprend BP sur la pile

        ret                             ; Retourne à l'appelant

_getfontptr     endp                    ; Fin de la procédure

;-- Fin ----------------------------------------------------------

_text   ends                            ; Fin du segment de programme
        end                             ; Fin de la source en assembleur
```

4.8.3. Désactiver l'écran

Nombreuses sont les situations où l'affichage écran d'une carte vidéo doit être inhibé comme par exemple lors du reposement de l'écran. Les cartes EGA et VGA connaissent également des méthodes variées pour interdire la création d'une image vidéo et nous allons décrire l'une de ces techniques dans le cadre de cette section. Nous éluciderons en outre les points suivants :

- ✓ Le rôle du contrôleur d'attributs dans la création de l'image vidéo
- ✓ Utilisation du contrôleur d'attributs pour désactiver l'affichage écran

Le rôle du contrôleur d'attributs dans la création de l'image vidéo

Parmi le quartet composé du contrôleur CRT, du contrôleur graphique, du contrôleur d'attributs et du séquenceur pilotant les cartes EGA et VGA, c'est le contrôleur d'attributs qui assure la connexion entre les informations d'image et de couleur. Il suffit que l'on interdise son fonctionnement pour qu'aucune information de couleur ne soit disponible. Le résultat est que l'écran reste sombre. Cela revient à désactiver l'affichage écran et remplit par conséquent le but visé.

En principe, il n'est pas si facile d'interdire le fonctionnement des contrôleurs d'une carte EGA et VGA mais ce n'est pas le cas du contrôleur d'attributs. Ici, il existe un bit dans le registre d'index qui permet pratiquement de désactiver le contrôleur. Il s'agit du

bit 5 qu'il suffit de régler sur 0 pour empêcher la transmission des informations de couleur à travers le contrôleur.

La situation de ce bit est quelque peu déplorable puisqu'avec les autres contrôleurs, le registre d'index sert uniquement à recevoir le numéro d'un registre à adresser. Mais comme il en va autrement avec le registre d'index du contrôleur d'attributs, interviennent ici les registres d'attributs, notamment les registres de palette. Etant donné qu'avant l'accès à un registre de palette dans lequel il faut appeler le registre d'index, il convient de spécifier l'action du contrôleur d'attributs, il suffit tout simplement d'interdire le bit concerné dans le registre d'index. Cela est d'autant plus utile lorsqu'on sait que le contrôleur d'attributs ne dispose que de 21 registres. Seuls les bits 0 à 4 sont nécessaires pour recevoir le numéro de registre dans le registre d'index.

Mais avant de supprimer le cinquième bit du contrôleur d'attributs, il faut lire auparavant le registre d'état du contrôleur CRT. Le reset du contrôleur d'attributs ne devient effectif qu'à cette condition. Cette action est généralement utile lorsqu'il faut remettre ce bit sur 1 pour que le contrôleur d'attributs puisse poursuivre son travail.

Désactivation de l'écran dans la pratique

Les deux programmes VONOFFP.PAS et VONOFFC.C démontrent l'action d'activer et de désactiver l'écran à travers le contrôleur d'attributs. Cette opération ne peut s'effectuer qu'après avoir prouvé l'existence d'une carte EGA ou VGA. Dans le cas contraire, le programme s'interrompt en émettant un bref message.

Dès qu'une carte EGA ou VGA est découverte, le programme renvoie un bref message et laisse cinq secondes à l'utilisateur pour le lire. L'écran est ensuite désactivé à l'aide de la procédure ou fonction ScrOff.

Comme nous l'avons déjà décrit, le registre d'état du contrôleur CRT est d'abord lu dans ScrOff. Pour que des cartes reliées à un moniteur couleur ou leurs homologues monochromes puissent profiter de la routine, un double reset est exécuté : une fois à travers l'adresse monochrome du registre d'état (3BAh) et une autre fois à travers l'adresse couleur (3DAh). Il n'y a rien à craindre ici parce que l'un des deux ports reste inoccupé ce qui ne nuit pas à la lecture.

Il faut inscrire ensuite la valeur 0 dans le registre d'index du contrôleur d'attributs à l'adresse de port 3C0h, régler le bit 5 sur 0 et interdire l'activité du contrôleur. L'écran devient immédiatement noir.

Rien de spécial ne se produit dans la procédure ScrOn. Elle est appelée dans les deux programmes une fois que l'utilisateur a appuyé sur une touche. Il suffit d'inscrire ici la valeur 20h dans le registre d'index du contrôleur d'attributs pour qu'une image vidéo apparaisse aussitôt sur l'écran.

Listing : VONOFFP.PAS

```
{*********************************************************************
*                          V O N O F F P . P A S                     *
**---------------------------------------------------------------------**
*  Fonction  : Montre comment éteindre et rallumer un écran EGA ou VGA *
**---------------------------------------------------------------------**
*  Auteur          : MICHAEL TISCHER                                  *
*  Développé le    : 05.08.1990                                       *
*  dernière MAJ    : 14.01.1991                                       *
*********************************************************************}

program VOnOffP;

uses DOS, CRT;                                  { Inclut les unités }

{-- Constantes ------------------------------------------------------}

const EV_STATC      = $3DA;       { Registre d'état couleur EGA/VGA }
      EV_STATM      = $3BA;         { Registre d'état mono EGA/VGA }
      EV_ATTR       = $3C0;       { Contrôleur d'attribut EGA/VGA }

procedure CLI; inline( $FA );               { Inhibe les interruptions}
procedure STI; inline( $FB );             { Rétablit les interruptions }

{*********************************************************************
*  ScrOff : Désactive un écran EGA ou VGA.                           *
**---------------------------------------------------------------------**
*  Entrée  : néant                                                   *
*********************************************************************}

procedure ScrOff;

var dummy : BYTE;           { Pour mémoriser les contenus des registres }

begin
  cli;                              { Pas d'interruption pour le moment }
  dummy := port[EV_STATC];           { Reset du registre d'état couleur }
  dummy := port[EV_STATM];             { Reset du registre d'état mono }
  port[EV_ATTR] := $00;                      { Efface le bit 5 ce qui }
                       { supprime la liaison avec le contrôleur d'écran }
  sti;                                  { Rétablit les interruptions }
end;

{*********************************************************************
*  ScrOn : Réactive un écran EGA ou VGA.                             *
**---------------------------------------------------------------------**
*  Entrée : néant                                                    *
*********************************************************************}

procedure ScrOn;

var dummy : BYTE;           { Pour mémoriser les contenus des registres }

begin
  cli;                                      { Inhibe les interruptions }
  dummy := port[EV_STATC];           { Reset du registre d'état couleur }
  dummy := port[EV_STATM];             { Reset du registre d'état mono }
  port[EV_ATTR] := $20;                      { Active le bit 5 ce qui }
                                  { rétablit la liaison avec le contrôleur d'écran}
  sti;                                  { Autorise à nouveau les interruptions }
end;

{*********************************************************************
*  IsEgaVga : Teste la présence d'une carte EGA ou VGA .             *
**---------------------------------------------------------------------**
*  Entrée : néant                                                    *
*  Sortie : TRUE, si carte EGA ou VGA, sinon FALSE                   *
*********************************************************************}

function IsEgaVga : boolean;

var Regs : Registers;             { Resgitres pour gérer l'interruption }

begin
  Regs.AX := $1a00;                 { La fonction 1Ah n'existe qu'en VGA }
  Intr( $10, Regs );
  if ( Regs.AL = $1a ) then                      { Est-elle disponible ?}
    IsEgaVGa := TRUE
  else
    begin
      Regs.ah := $12;                              { Appelle l'option 10h }
      Regs.bl := $10;                             { de l'interruption 12h }
      Intr($10, Regs);                  { Déclenche l'interruption vidéo }
      IsEgaVga := ( Regs.bl <> $10 );
    end;
end;

{*********************************************************************}
{--                        PROGRAMME PRINCIPAL                      --}
{*********************************************************************}

var ch : char;                                  { Mémorise une touche }

begin
  ClrScr;
  writeln( 'VONOFFP  -  (c) 1990, 1991 by MICHAEL TISCHER'@13@10 );
  if IsEgaVga then                                { Carte EGA ou VGA ? }
    begin                                                { Oui, on y va }
      writeln( 'Attention !''écran va s''éteindre dans 5 secondes');
      writeln( 'Appuyez ensuite sur une touche pour le rallumer');

      Delay( 5000 );                                { Attente de 5 secondes }
      while KeyPressed do                  { Vide le buffer du clavier }
        ch := ReadKey;
      ScrOff;                                            { Eteint l'écran }
      ch := ReadKey;                                 { Attend une frappe }
      ScrOn;                                          { Restaure l'écran }
      writeln ( @13@10@10 + 'C''est tout ...' );
    end
  else                                          { Non, pas de carte EGA ou VGA }
    writeln( 'Attention ! Ce programme exige ' +
             'une carte EGA ou VGA.' );
end.
```

Listing : VONOFFC.C

```
/*********************************************************************
*                          V O N O F F C . C                         *
**---------------------------------------------------------------------**
*  Fonction         : Montre comment éteindre et rallumer un écran    *
*                     EGA ou VGA                                      *
**---------------------------------------------------------------------**
*  Auteur           : MICHAEL TISCHER                                 *
*  Développé le     : 26.08.1990                                      *
*  Dernière MAJ     : 14.02.1992                                      *
**---------------------------------------------------------------------**
*  (MICROSOFT C)                                                      *
*  Compilation      : CL /AS vonoffc.c                                *
**---------------------------------------------------------------------**
*  (BORLAND TURBO C)                                                  *
*  Compilation      : dans l'environnement de développement intégré   *
*********************************************************************/

#include <dos.h>                                /* Fichiers d'inclusion */
#include <conio.h>
#include <stdio.h>

#ifdef __TURBOC__                               /* Compilation par  Turbo C? */
  #define CLI()         disable()
  #define STI()         enable()
  #define outpw( p, w ) outport( p, w )
  #ifndef inp
    #define outp( p, b )  outportb( p, b )
    #define inp( p )      inportb( p )
  #endif
#else                                           /* Non , par QuickC 2.0 ou MSC */
  #include <conio.h>
  #define MK_FP(seg,ofs) ((void far *)\
                         (((unsigned long)(seg) << 16) | (ofs)))
  #define CLI()         _disable()
  #define STI()         _enable()
#endif

/*-- Constantes ------------------------------------------------------*/

#define EV_STATC 0x3DA                  /* Registre d'état couleur EGA/VGA */
```

```
#define EV_STATM 0x3BA                  /* Registre d'état mono EGA/VGA */
#define EV_ATTR  0x3C0                  /* Contrôleur d'attribut EGA/VGA */

/*****************************************************************
*  ScrOff : Désactive un écran EGA ou VGA                        *
*----------------------------------------------------------------*
*  Entrée : néant                                                *
*****************************************************************/

void ScrOff( void )
{
 CLI();                                 /* Inhibe les interruptions */
 inp( EV_STATC );                       /* Reset du registre d'état couleur */
 inp( EV_STATM );                       /* Reset du registre d'état Mono */
(void)outp( EV_ATTR, 0x00 );            /* Efface le bit 5 ce qui supprime */
                                        /* la liaison avec le contrôleur d'écran */
 STI();                                 /* Rétablit les interruptions */
}

/*****************************************************************
*  ScrOn : Réactive un écran EGA ou VGA                          *
*----------------------------------------------------------------*
*  Entrée : néant                                                *
*****************************************************************/

void ScrOn( void )
{
 CLI();                                 /* Inhibe les interruptions */
 inp( EV_STATC );                       /* Reset du registre d'état couleur */
 inp( EV_STATM );                       /* Reset du registre d'état Mono */
(void)outp( EV_ATTR, 0x20 );            /* Active le bit 5 , ce qui */
                                        /* rétablit la liaison avec le contrôleur d'écran */
 STI();                                 /* Rétablit les interruptions */
}

/*****************************************************************
*  IsEgaVga : Teste la présence d'une carte EGA ou VGA           *
*----------------------------------------------------------------*
*  Entrée  : néant                                               *
*  Sortie  : TRUE, si carte EGA ou VGA, sinon FALSE              *
*****************************************************************/

int IsEgaVga( void )
{
 union REGS Regs;                       /* Registres pour gérer l'interruption */

 Regs.x.ax = 0x1a00;                    /* La fonction 1Ah n'existe qu'en VGA */
 int86( 0x10, &Regs, &Regs );
 if ( Regs.h.al == 0x1a )               /* Est-elle disponible ? */
  return 1;
 else
  {
   Regs.h.ah = 0x12;                    /* Appelle l'option 10h */
   Regs.h.bl = 0x10;                    /* de la fonction 12h */
   int86(0x10, &Regs, &Regs);           /* Déclenche l'interruption vidéo */
   return ( Regs.h.bl != 0x10 );
  }
}

/*****************************************************************
*  Delay : Introduit une temporisation à l'aide du timer du BIOS *
*----------------------------------------------------------------*
*  Entrée  : Durée de la temporisation en s                      *
*  Sortie  : néant                                               *
*****************************************************************/

void Delay( int pauslen )
{
 unsigned int temps_hi,                 /* Compteurs de temps */
              temps_lo,
              ticks;
 union REGS   inregs,                   /* Registres du processeur */
              outregs;

 ticks = pauslen * 182 / 10;
 inregs.h.ah = 0;                       /* Fonction 00h = Lit le timer */
 int86( 0x1a, &inregs, &outregs );
 temps_hi = outregs.x.cx;               /* Mémorise le temps */
 temps_lo = outregs.x.dx;

 while ( ticks )                        /* Répète l'opération jusqu'à ce que */
 {                                      /* le compteur de tops soit à 0 */
  int86( 0x1a, &inregs, &outregs );     /* Lit le temps */

          /*-- Nouveau top ? ------------------------------------*/

  if ( temps_hi != outregs.x.cx  ||  temps_lo != outregs.x.dx )
  {                                                 /* Oui */
   temps_hi = outregs.x.cx;             /* Note les valeurs des compteurs */
   temps_lo = outregs.x.dx;
   --ticks;                             /* Décrémente le nombre de tops restants */
  }
 }
}

/*****************************************************************/
/*--                    PROGRAMME PRINCIPAL                    --*/
/*****************************************************************/

void main( void )
{
 int i;                                 /* Compteur d'itérations */

 for ( i=0; i<25; ++i )                 /* Eteint l'écran */
  printf( "\n" );

 printf( "VONOFFC  -  (c) 1990, 92 by MICHAEL TISCHER\n\n" );
 if ( IsEgaVga() )                      /* EGA ou VGA ? */
 {                                      /* Oui, on y va */
  printf( "Attention l'écran va s'éteindre ! \
     "dans 5 secondes.\nActionnez ensuite une touche quelconque" \
     " pour le rallumer..." );
  Delay( 5 );                           /* On attend 5 secondes */
  while ( kbhit() )                     /* Retire les touches du buffer du clavier */
    getch();
  ScrOff();                             /* Eteint l'écran */
  getch();                              /* Attend une frappe de touche */
  ScrOn();                              /* Rallume l'écran */
  printf( "\n\n\nC'est tout ....\n" );
 }
 else                                   /* Non pas de carte EGA ou VGA */
  printf( "Attention ! Ce programme exige " \
     "une carte EGA ou VGA.\n" );
}
```

4.8.4. Le principe des plans de bits

L'histoire des cartes vidéo PC est manifestement imprégnée du souci d'accroître constamment la résolution avec un nombre de couleurs toujours plus important. Mais l'augmentation de la résolution s'accompagne d'un besoin en RAM vidéo plus accru d'autant plus que les innombrables couleurs utilisées pour coder les points ne nécessitent plus un seul bit mais 4 et même 8 bits. C'est d'ailleurs le cas des modes 256 couleurs des cartes VGA modernes. La conséquence est qu'aujourd'hui, les cartes EGA et VGA sont fournies avec une RAM de base de 256 Ko qui peut être entièrement utilisée en fonction du mode vidéo et du nombre de pages écran gérées.

Un problème survient avec le dépassement de la limite des 64 Ko. La zone d'adresse du PC s'élevant à 1 Mo ne contient plus de place disponible susceptible de contenir une

RAM vidéo supplémentaire. Cette section tente d'expliquer comment intégrer la RAM vidéo de 256 Ko des cartes EGA et VGA dans la zone d'adresse du PC et quelles sont les conséquences en matière de programmation de ces cartes. Voici les thèmes que nous allons aborder :

- ✓ La segmentation de la RAM vidéo en plans de bits
- ✓ La fonction et la signification des registres Latch
- ✓ Le rôle du contrôleur graphique des registres Latch
- ✓ La programmation et le fonctionnement des divers modes Read et Write.

Segmentation de la RAM vidéo en plans de bits

Dans la zone d'adresse du PC, les cartes vidéo ne disposent que des segments de mémoire A (à partir de A000:0000) et B (à partir de B000:0000) où le segment B est déjà réservé pour la RAM vidéo des cartes MDA, CGA et Hercules. Il ne reste plus que le segment A pour les cartes EGA et VGA. Les constructeurs des premières cartes EGA le destinaient à une tâche inimaginable consistant à rendre adressable une RAM vidéo de 256 Ko à travers une zone de 64 Ko seulement.

La solution trouvée se présente sous la forme de quatre plans de bits répartis dans la RAM vidéo d'une carte EGA ou VGA. Avec une capacité de 256 Ko, la carte vidéo libère ainsi 64 Ko sur chaque plan de bits adressable entièrement à travers le segment de mémoire A à partir de l'adresse A000:0000. Si la quantité de Ko est moindre, comme les premières cartes EGA originales d'IBM qui n'étaient fournies qu'avec 64 Ko, chaque plan de bits reçoit naturellement une portion de RAM vidéo moins importante.

Ce principe d'organisation apparu en 1984 concerne, aujourd'hui comme autrefois, tous les types de cartes depuis EGA à VGA même s'il a été amélioré avec les cartes Super VGA tel que nous le montre la section 4.9.

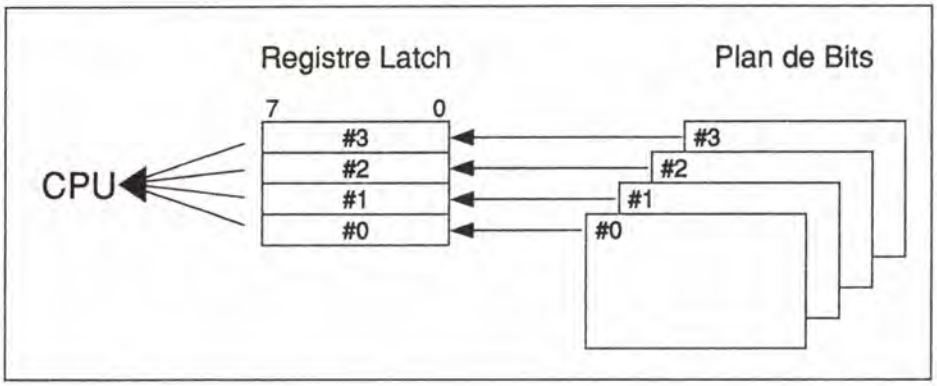
Fonction et signification des registres Latch

En mode texte, l'intérêt des plans de bits ne peut être apprécié que pour la programmation lorsqu'on souhaite accéder manuellement aux jeux de caractères. Mais en mode graphique, ils agissent comme une épée de Damoclès. Peu importe comment les informations de points sont distribuées sur les divers plans de bits puisque cela dépend exclusivement du mode graphique en cours. Sans tenir compte de cette circonstance, l'électronique de la carte EGA/VGA active les quatre registres 8 bits entre l'unité centrale et la RAM vidéo appelés aussi registres Latch. Chaque registre de ce type correspond à l'un des quatre plans de bits mais n'est pas directement adressable par un programme.

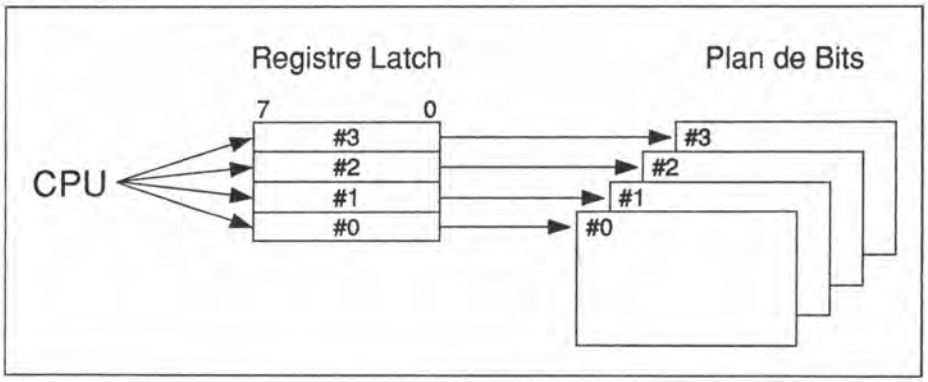
Si un programme exécute un accès en lecture sur un octet de la RAM vidéo, les quatre registres Latch sont chargés avec l'octet à partir de leur plan de bits. Son adresse d'offset

est indiquée en tenant compte de l'instruction de langage machine ayant déclenché l'accès en lecture. Si l'adresse d'offset vaut par exemple 9, c'est le dixième octet (0 étant le premier) issu du premier plan de bits qui est chargé dans le premier registre Latch, le dixième octet du second plan de bits dans le second registre Latch et ainsi de suite pour les troisième et quatrième registres Latch.

La même procédure s'applique lors d'un accès en écriture à la RAM vidéo. Ici, le contenu des quatre registres Latch est inscrit dans leur plan de bits correspondant, notamment à l'adresse d'offset spécifiée lors de l'accès en écriture.



Chargement des quatre registres Latch lors d'un accès en lecture à la RAM vidéo



Manipulation des quatre registres Latch lors d'un accès en écriture à la RAM vidéo

Ce n'est pas tout en ce qui concerne le chargement et l'écriture des registres Latch. Lors d'un accès en lecture à la RAM vidéo, il faut retourner un octet à l'unité centrale. De même, un octet doit être transféré depuis l'unité centrale vers la RAM vidéo lors d'un accès en écriture.

Sachant que lors d'un accès en lecture, il n'est possible de transférer qu'un seul octet et non quatre à l'unité centrale, il reste à savoir lequel des quatre octets des registres Latch sera transmis à l'unité centrale. Un problème similaire se pose lors de l'écriture dans la RAM vidéo. Dans ce cas, on ne voudra sûrement pas inscrire tous les quatre registres et par conséquent inscrire un octet avec la même valeur dans les quatre plans de bits. Ainsi, les quatre plans de bits auraient grosso modo le même contenu et on ne pourrait travailler qu'avec 64 Ko et s'épargner le va-et-vient entre les quatre plans de bits et les registres Latch.

Rôle du contrôleur graphique dans la programmation graphique

Les neuf registres du contrôleur graphique répondent à toutes ces questions. Ils représentent un élément important d'une carte EGA/VGA. Ils déterminent la provenance, la destination et la manière de tous les accès en lecture et écriture à la RAM vidéo. Cela ne concerne pas seulement les divers modes graphiques, mais aussi les modes texte. Dans le cadre de la programmation graphique, on doit fréquemment avoir recours à ces registres et leur affecter toutes sortes de valeurs en fonction des opérations envisagées. Tel n'est pas le cas en mode texte.

Ici, les registres sont définis directement lors de l'initialisation d'un mode de ce type à travers le BIOS de sorte qu'il n'est plus nécessaire de les adresser. Mais en mode texte, l'entrée et la sortie dans et depuis la RAM vidéo s'effectue à travers les quatre registres Latch même si ce processus reste accessible à un programme.

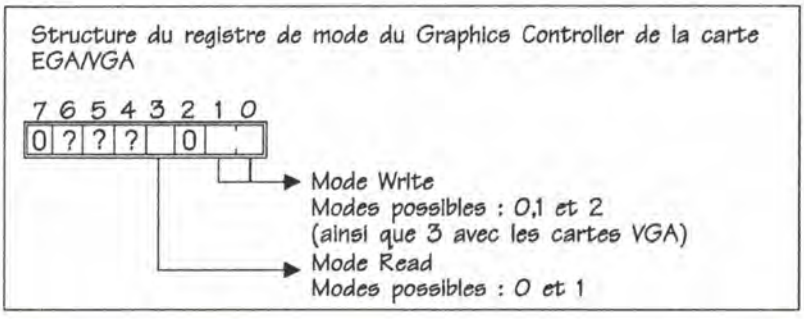
Les neuf registres du contrôleur graphique de la carte EGA et leurs valeurs par défaut

Registre	Signification	Défaut
00h	Set / Reset	00h
01h	Enable Set / Reset	00h
02h	Color Compare	00h
03h	Function Select	00h
04h	Read Map Select	00h
05h	Mode	00h
06h	Miscellaneous	divers
07h	Color Don't Care	0Fh
08h	Bit Mask	FFh

La programmation de ces registres ressemble à l'accès aux registres CRTC de la carte graphique Hercules. Ici aussi existe à l'adresse de port 3CEh un registre d'adresse dans lequel doit tout d'abord être chargé le numéro du registre auquel il s'agit d'accéder à

l'intérieur du Graphic Controller. La valeur destinée à ce registre peut ensuite être envoyée au registre de données, qui se trouve immédiatement après le registre d'adresse, à l'adresse de port 3CFh. Il n'est pas nécessaire d'accéder séparément à ces deux ports et on peut donc réaliser cet accès à l'aide d'une instruction OUT 16 bits sur le registre d'adresse. Le registre AX, qui est dans ce cas envoyé sur ce port, doit contenir pour cela le numéro du registre dans sa partie de plus faible poids (le registre AL) et la valeur à y charger dans sa partie de plus fort poids (le registre AH). On peut ainsi charger des valeurs dans les différents registres du Graphics-Controllers, mais tout accès en lecture est impossible avec la carte EGA.

Le déroulement des opérations lors de l'accès à la RAM vidéo est réglé par le contenu du registre numéro 5, le registre de mode. Son contenu définit les modes Read et Write actuels, c'est-à-dire la façon dont les données des registres Latch doivent être combinées avec les autres registres du contrôleur graphique et avec les données de l'unité centrale.



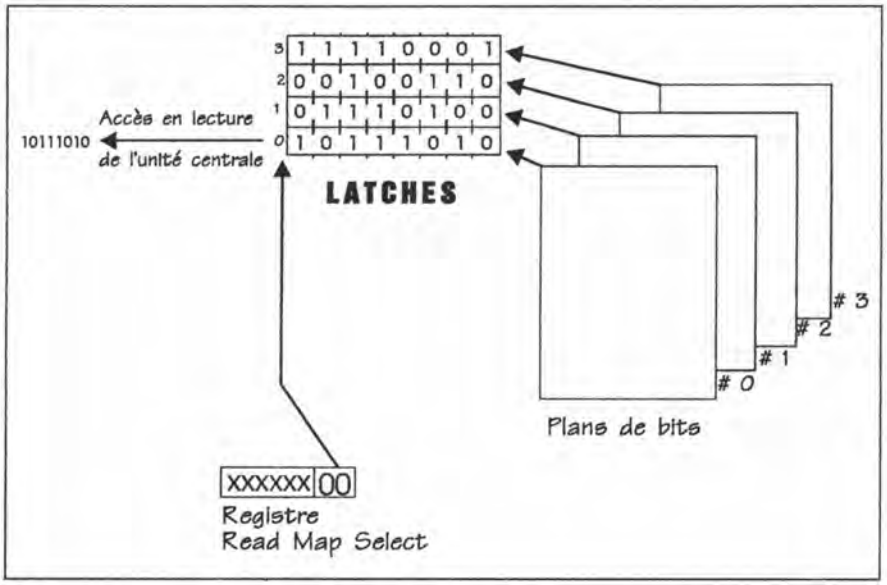
Les paragraphes suivants décrivent en détail le fonctionnement des modes Read et Write, leur fonction et la signification des autres registres du contrôleur graphique pour ces modes. Vous constaterez très vite que dans la vie pratique vous n'aurez pas besoin de tous les modes parce que les autres modes s'avèrent tout simplement superflus.

Mode Read 0

Le mode Read 0 permet à un programme de lire des octets provenant d'un plan de bits déterminé. Cela est par exemple intéressant lorsqu'il s'agit de sauvegarder une partie de la RAM vidéo. A cet effet, les quatre plans de bits s'exécutent successivement dans une boucle et la zone souhaitée est chargée à partir de chaque plan de bits et stockée dans la mémoire principale.

Le mode Read 0 est le plus simple des deux modes Read. Sous ce mode, lors d'un accès en écriture à la RAM vidéo, l'octet appelé pour chacun des quatre plans de bits est tout d'abord chargé dans les quatre registres Latch correspondants. Le contenu du registre Latch dont le numéro est spécifié dans les deux bits inférieurs du registre Read Map Select (Registre 4) est ensuite transmis à l'unité centrale.

Dans ce registre, seuls les deux bits inférieurs sont occupés et sont déterminants pour le numéro du registre Latch dont le contenu doit arriver jusqu'à l'unité centrale. Lors de la programmation de ce registre, faites attention au fait que le premier plan de bits porte le numéro 0. N'inscrivez donc pas la valeur 1 dans ce registre si vous souhaitez accéder au premier plan de bits.



Accès en lecture à la RAM vidéo en mode Read 0

La séquence d'instructions en Assembleur suivante démontre le mode d'emploi du mode Read 0. Les premiers 8 Ko sont extraits du second plan de bits et chargés dans la mémoire principale. Il faut d'abord définir le mode Read 0 et charger ensuite le registre Read Map Select avec la valeur 1 pour pouvoir lire le second plan de bits. L'instruction Assembleur REP MOVSB est utilisée ensuite pour lire octet par octet les premiers 8 Ko depuis la RAM vidéo et les copier dans un buffer.

```
mov ax,ds           ;ES:DI placer sur le buffer cible
mov es,ax
mov di,offset puffer
mov ax,0A000h       ;DS:SI sur l'adresse de départ du plan de bits dans
                    ;la RAM vidéo
mov ds,ax
xor si,si
mov cx,8*1024       ;copier 8Ko

mov dx,3CEh         ;adresser le contrôleur graphique
mov ax,0005h        ;écrire mode Read 0 dans le registre Mode
out dx,ax
mov ax,0104h        ;écrire la valeur 1 (numéro de plan) dans
out dx,ax           ;le registre Read Map Select
```

```
rep movsb          ;copier les 8 Ko
```

Si vous examinez de près la séquence Assembleur ci-dessus, vous vous dîtes certainement qu'on pourrait accélérer la procédure de recopie en copiant 4096 mots au lieu des 8192 octets. On remplace alors l'instruction REP MOVSB par REP MOVSW et on charge le registre CX avec 4096 au lieu de 8192. Cette méthode convient parfaitement pour la recopie des cellules de mémoire dans la mémoire principale, mais elle risque de provoquer des conséquences désastreuses dans ce contexte.

Comme les accès 16 bits sont interdits dans la RAM vidéo, l'unité centrale exécute dans ce cas deux accès en lecture au niveau octet avant d'autoriser l'accès en écriture. Le second accès en lecture écrase par ailleurs le premier accès en lecture à l'intérieur du registre Latch avant même qu'il ne parvienne jusqu'à l'unité centrale. La conséquence est que dans le buffer cible on ne trouve que des mots entiers dans l'octet de poids fort et faible possède la même valeur. Seul l'octet de poids fort correspond au contenu réel de la RAM vidéo. En conclusion, les accès en lecture 16 bits à la RAM vidéo sont à éviter sur les cartes EGA et VGA de quelque nature que ce soit !

Mode Read 1

Si le sens et le fonctionnement du mode Read 0 sont relativement faciles à comprendre, l'affaire se complique avec le mode Read 1. Ici, il ne s'agit pas d'envoyer tel quel le contenu d'un registre Latch à l'unité centrale, mais de définir toutes sortes de combinaisons logiques concernant le contenu des quatre registres Latch.

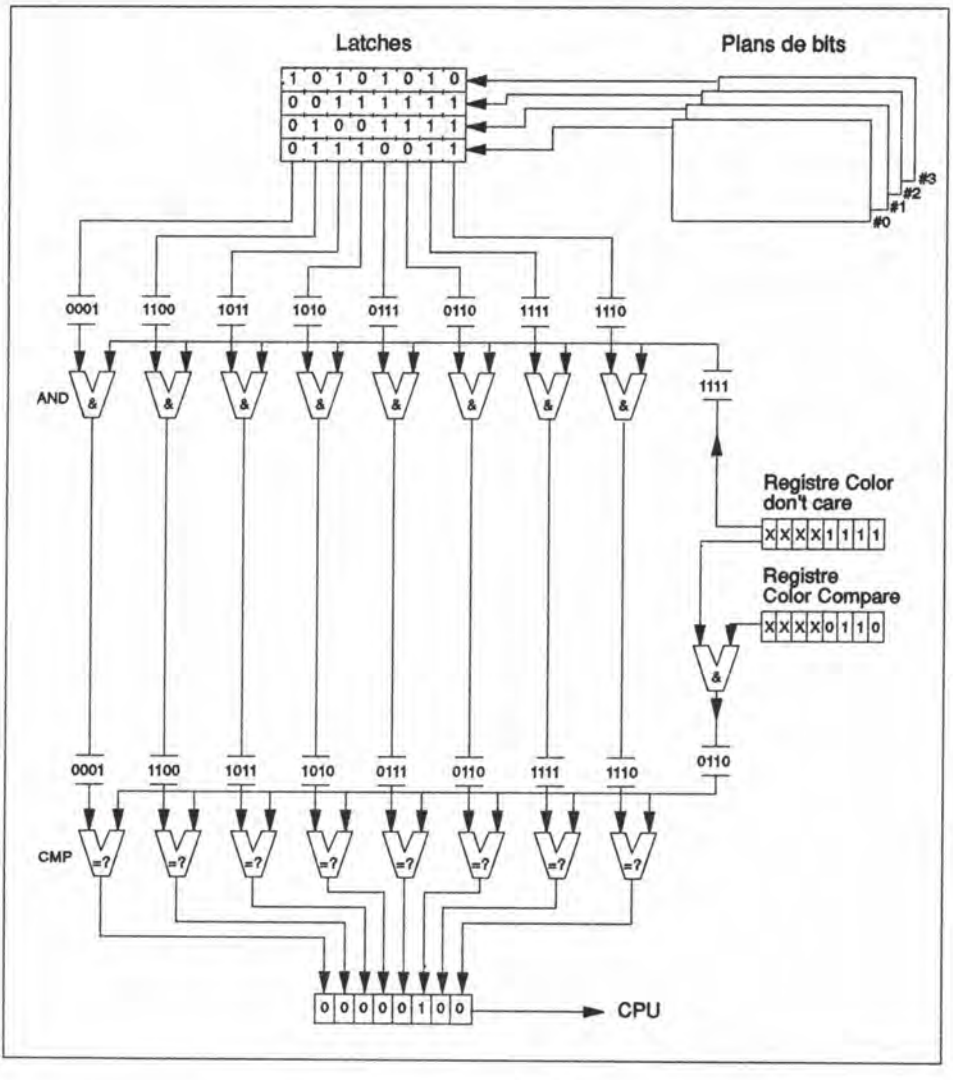
La tâche de ce mode consiste à vérifier si les bits des quatre registres Latch occupent une valeur déterminée. Cela peut être utilisé par la suite dans les modes graphiques des cartes EGA et VGA 16 couleurs pour la recherche des points dotés d'un code couleur spécifique. En pratique, ce mode est rarement utilisé.

Une fois les quatre registres Latch chargés, huit groupes composés chacun de quatre bits sont constitués. Chaque groupe contient les quatre bits qui occupent une position de bit identique à l'intérieur des quatre registres Latch. Par exemple, tous les bits situés à la position 0 sont réunis dans un groupe, de même que tous les bits situés à la position 1 et ceux des autres positions.

Chaque groupe de quatre est comparé ensuite avec la valeur fournie par le registre Color Compare préalablement chargé dans ce registre avant l'accès en lecture. L'octet issu de la comparaison est transmis à l'unité centrale en guise de résultat de l'opération de lecture.

Dans cet octet, tous les bits mis sur 1 sont ceux dont le groupe correspondant a obtenu la valeur fournie par le registre Color Compare. Tous les bits restants contiennent la

valeur 0. Après l'accès en lecture, on arrive ainsi à identifier avec précision le groupe de quatre qui correspond à la valeur transmise par le registre Color Compare.



Accès en lecture à la RAM vidéo en mode Read 1

A première vue, cela paraît insensé, mais c'est la pure vérité. Durant toute l'opération, il faut signaler la présence du registre Color Don't Care. Tant qu'il contient la valeur 00001111b, les divers groupes sont confrontés avec la valeur de comparaison issue du registre Color Compare. Chaque bit des quatre bits inférieurs du registre Color Don't Care correspond notamment à l'un des quatre plans de bits, le bit 0 au premier plan de

bits, le bit 1 au second et ainsi de suite. Lorsque l'un de ces bits contient la valeur 1, le plan correspondant est inclus dans le test de comparaison.

Si la valeur est 0, il se passe comme si la valeur de ce plan de bits correspond dans tous les cas au bit concerné du registre Color Compare. Cette procédure se poursuit parce que lors de l'indication de la valeur 0 dans le registre Color Don't Care, la valeur 11111111b est toujours retournée à l'unité centrale indépendamment du contenu des quatre registres Latch et du contenu du registre Color Compare. En fait, aucune comparaison entre les huit groupes et la valeur de comparaison n'a lieu mais tous les groupes sont considérés comme admis.

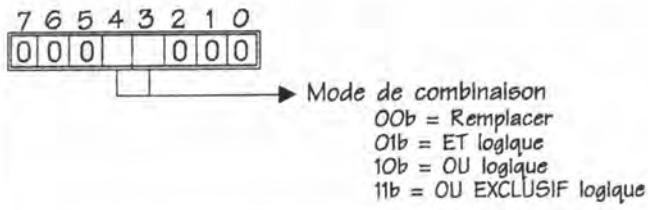
La séquence d'instructions Assembleur suivante montre l'utilité du mode Read 1 sous un aspect très théorique. Il s'agit de prouver quel groupe issu du premier octet contient la valeur 5 dans les quatre plans de bits de la RAM vidéo. Le registre Color Don't Care n'est pas explicitement programmé puisqu'on considère qu'il contient sa valeur par défaut 00001111b ce qui fait inclure tous les plans de bits dans la comparaison.

```
mov ax,0A000h       ;fixer ES sur la RAM vidéo
mov es,ax

mov dx,3CEh         ;adresser le contrôleur graphique
mov ax,0805h        ;écrire mode Read 1 dans
out dx,ax           ;le registre Mode
mov ax,0502h        ;écrire le code couleur 5 dans
                    ;le registre Color Compare
mov al,es:[0]       ;lire et comparer les points,
                    ;retourner le résultat en AL
or al,al            ;pas de bit sur 1?
Toutdifferent       ;non, groupe de quatre non trouvé
```

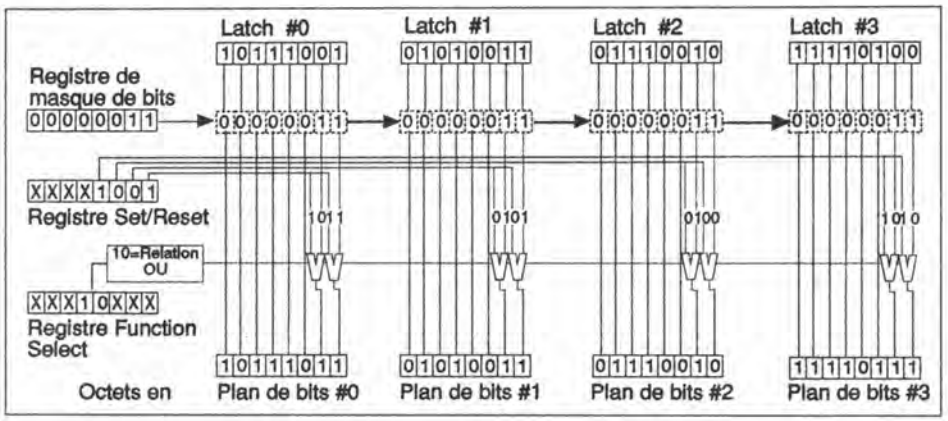
Mode Write 0

De nombreuses combinaisons liées au contenu des registres se créent lors de l'accès à la RAM vidéo à travers le mode Write 0. En fait, le contenu du registre Bit Mask décide si le contenu d'un bit des quatre registres Latch doit être transmis tel quel dans les quatre plans de bits ou il nécessite quelque traitement. Les différents bits du registre Bit Mask correspondent aux différents bits des quatre registres Latch. Si un bit contient la valeur 0 dans le registre Bit Mask, le bit correspondant dans les quatre registres Latch est transmis tel quel dans les quatre plans de bits. Si le bit contient au contraire la valeur 1, une combinaison se crée dans ce cas. Sa structure est définie par le contenu du registre Function Select. Comme le montre la figure suivante, les bits peuvent tout simplement être remplacés ou manipulés à l'aide des combinaisons logiques ET, OU ainsi que OU EXCLUSIF.

Structure du registre Function Select du Graphics Controller des cartes EGA/VGA



Le contenu du registre Enable Set/Reset décide quel devra être le partenaire de ces bits dans l'opération logique. Si les quatre bits inférieurs contiennent la valeur 1, l'opération logique se fera avec le contenu des 4 bits inférieurs du registre Set/Reset. Chacun de ces bits sera combiné avec un Latch par une opération définie par le contenu du registre Function Select. Tous les bits du registre Latch 0 seront alors combinés avec le bit 0 du registre Set/Reset par l'opération logique sélectionnée. Tous les bits à manipuler des registres Latch 1, 2 et 3 seront combinés de la même façon avec les bits 1, 2 et 3 respectivement du registre Set/Reset. L'octet de l'unité centrale qui est inévitablement transféré vers le contrôleur graphique lors d'un accès en écriture, ne joue donc aucun rôle dans ce cas, l'accès en écriture servant uniquement à déclencher l'opération et ne pouvant donc nullement influer sur le contenu des registres Latch ni, par conséquent, sur celui des plans de bits.



Accès en lecture à la RAM vidéo en mode Write 0 lorsque le registre Enable Set/Reset contient la valeur 00001111b

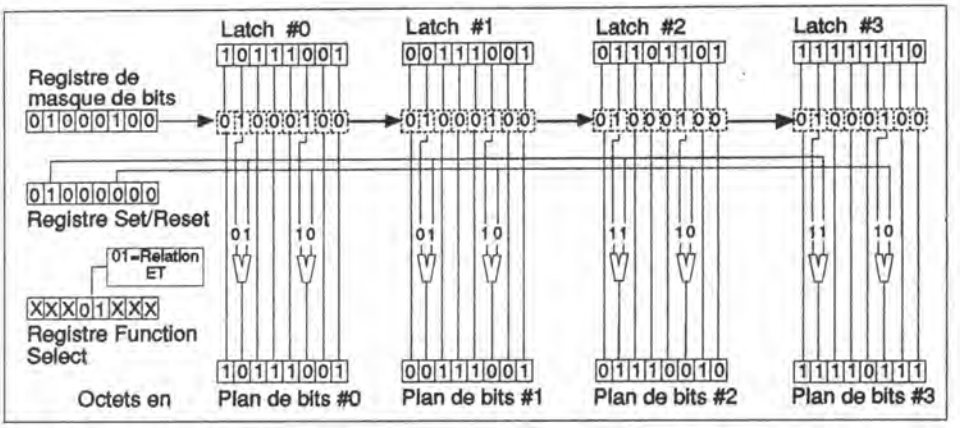
La séquence assembleur suivante a pour but de doter de la couleur 1011b les groupes du bit 2 du premier octet de la RAM vidéo sans agir sur le contenu des autres groupes. Comme vous le remarquerez dans le cadre d'une des sections suivantes, il s'agit là d'une

technique qui est toujours utilisée pour définir les points dans les modes graphiques 16 couleurs des cartes EGA et VGA.

Comme la couleur des autres groupes ne doit pas être modifiée, leur contenu sera tout d'abord chargé dans les registres Latch par un accès en lecture à la RAM vidéo. Peu importe ici quel mode Read est activé puisque ce n'est pas l'octet transmis à l'unité centrale qui nous intéresse mais seulement le simple fait que les registres Latch soient chargés. Seuls les groupes de la position 2 devant être manipulés, alors que les autres devront être renvoyés inchangés dans les plans de bits, la valeur 00000100b (04h) est tout d'abord chargée dans le registre de masque bits. La valeur 0 est ensuite écrite dans le registre Function Select car les bits 0 et 2 à manipuler doivent être remplacés par une nouvelle combinaison de bits. Nous écrivons dans le registre Set/Reset la couleur (1011b = 0Bh) que devront recevoir les deux bits. Pour que cette couleur soit tirée de ce registre, le dernier accès aux registres du contrôleur graphique doit consister à écrire la valeur 1111b (0Fh) dans le registre Enable Set/Reset. Immédiatement après cela peut être effectué l'accès en écriture à la RAM vidéo, lors duquel l'octet du processeur transmis ne joue toutefois aucun rôle.

```
mov ax,0A000h       ;adresse de segment de la RAM vidéo
mov ds,ax           ;dans DS
mov al,ds:[0]       ;charger octet 0 dans le registre Latch
mov dx,3CEh         ;adresser le contrôleur graphique
mov ax,0005h        ;mode Read 0, mode Write 0
out dx,ax           ;écrire dans le registre Mode
mov al,03h          ;écrire 0 dans le registre
out dx,ax           ;Function Select
mov ax,0408h        ;écrire le masque de bits dans
out dx,ax           ;le registre Bit Mask
mov ax,0B00h        ;écrire nouveau code couleur dans
out dx,ax           ;le registre Set/Reset
mov ax,0F01h        ;écrire 1111b dans le registre
out dx,ax           ;Enable Set/Reset
mov ds:[0],al       ;manipuler et réécrire le registre Latch
```

Il en va autrement si le registre Enable Set/Reset contient la valeur 0. Dans ce cas en effet, tous les bits à manipuler des quatre registres Latch sont combinés Latch par Latch avec l'octet de l'unité centrale. Ici aussi, la nature de la combinaison logique dépend du contenu du registre Function Select. Si C'est par exemple l'opération logique OU qui a été choisie et si ce sont les bits 1, 2, 4 et 6 qui doivent être manipulés, ces bits dans les quatre registres Latch seront combinés individuellement, par un OU logique avec les bits 1, 2, 4 et 6 de l'octet de l'unité centrale. Du fait que ce mode consiste à combiner les différentes positions de bits des quatre registres Latch avec les valeurs correspondantes de l'octet de l'unité centrale, il est plus rarement utilisé que la combinaison à travers le registre Set/Reset, telle qu'elle a été décrite plus haut.



Accès en écriture à la RAM vidéo en mode Write 0, lorsque le registre Enable-Set/Reset contient la valeur 00000000b

Mode Write 1

Par comparaison avec les combinaisons complexes du mode Write 0, le mode Write 1 paraît discret et peu impressionnant. Le contenu des registres et celui de l'octet transmis par l'unité centrale ne jouent aucune rôle ici car le contenu des quatre registres Latch est écrit tel quel à l'adresse d'offset indiquée dans les quatre plans de bits. Cela peut être intéressant, par exemple, lorsqu'il s'agit de copier les codes de couleur de 8 points graphiques contigus dans huit autres points graphiques. Dans ce cas, l'octet contenant les 8 points peut tout d'abord être lu avec un mode Read quelconque et être ainsi chargé dans les différents registres Latch. L'étape suivante consistera alors à effectuer un accès en écriture à l'octet de la RAM vidéo dans lequel les codes couleur pour les 8 points graphiques sont censés être copiés. Le contrôleur graphique transfère alors automatiquement les quatre registres Latch inchangés dans la position spécifiée des quatre plans de bits.

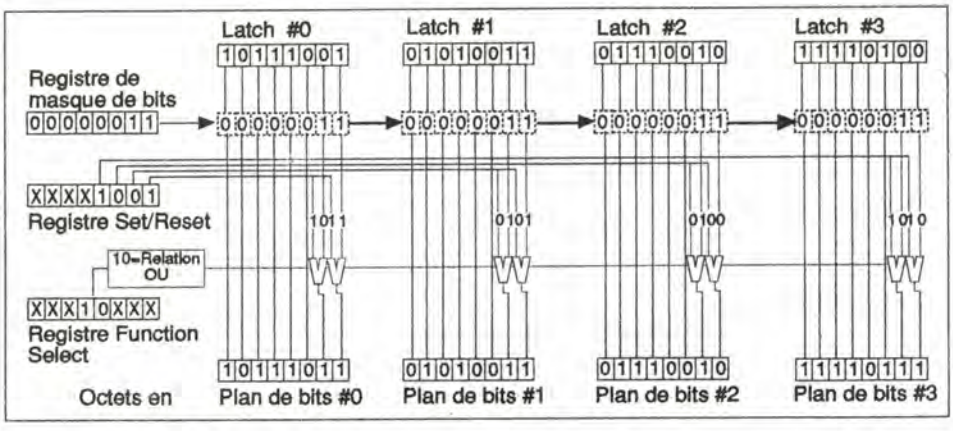
Si ces codes couleurs doivent être encore copiés dans d'autres points graphiques, il suffira ensuite d'effectuer des accès en écriture à la cellule de mémoire voulue à l'intérieur de la RAM vidéo. Un accès en lecture n'est plus nécessaire puisque les registres Latch ont déjà été chargés et que leur contenu n'a pas été modifié par l'accès en écriture.

```
mov ax,0A000h       ;adresse de segment de la RAM vidéo
mov ds,ax           ;dans DS et ES
mov es,ax
mov si,0000h        ;zone source commence à 0000h
mov di,0200h        ;zone cible commence à 0200h
mov cx,100h         ;copier 256 Ko
cld                 ;compter par ordre croissant les instructions chaîne
```

```
mov dx,3CEh          ;adresser le contrôleur graphique
mov ax,0105h         ;mode Read 0, mode Write 1
out dx,ax            ;écrire dans le registre Mode

rep movab            ;copier en une seule passe les quatre plans de bits
                     ;dans la zone
```

Mode Write 2

Le mode Write 2 est un peu une combinaison des différents modes du mode Write 0. Comme sous le mode Write 0, c'est ici aussi le registre de masque bits qui définit quels bits des registres Latch doivent être repris inchangés et lesquels doivent être manipulés auparavant. C'est à nouveau le mode de combinaison, défini par le registre Enable Set/Reset, qui fixe comment les différents bits doivent être manipulés. Quel que soit le contenu du registre Enable Set/Reset, sous le mode Write 2, les 4 bits inférieurs de l'octet de l'unité centrale sont systématiquement combinés avec les registres Latch. Le bit 0 de l'octet de l'unité centrale est combiné avec tous les bits du registre Latch 0 qui doivent être manipulés. Il en va de même pour les bits 1, 2 et 3 de l'unité centrale, qui sont combinés respectivement avec tous les bits des registres 1, 2 et 3.



Accès en écriture à la RAM vidéo en mode Write 2

Ce mode convient donc particulièrement bien à fixer la couleur de points isolés, comme nous l'avons montré dans l'exemple pour le mode Write 0. Par rapport au mode Write 0, toutefois, la séquence assembleur correspondante est plus courte puisqu'il n'est pas besoin ici de programmer ni le registre Enable Set/Reset, ni le registre Set/Reset. Voici donc le même exemple pour le mode Write 2 :

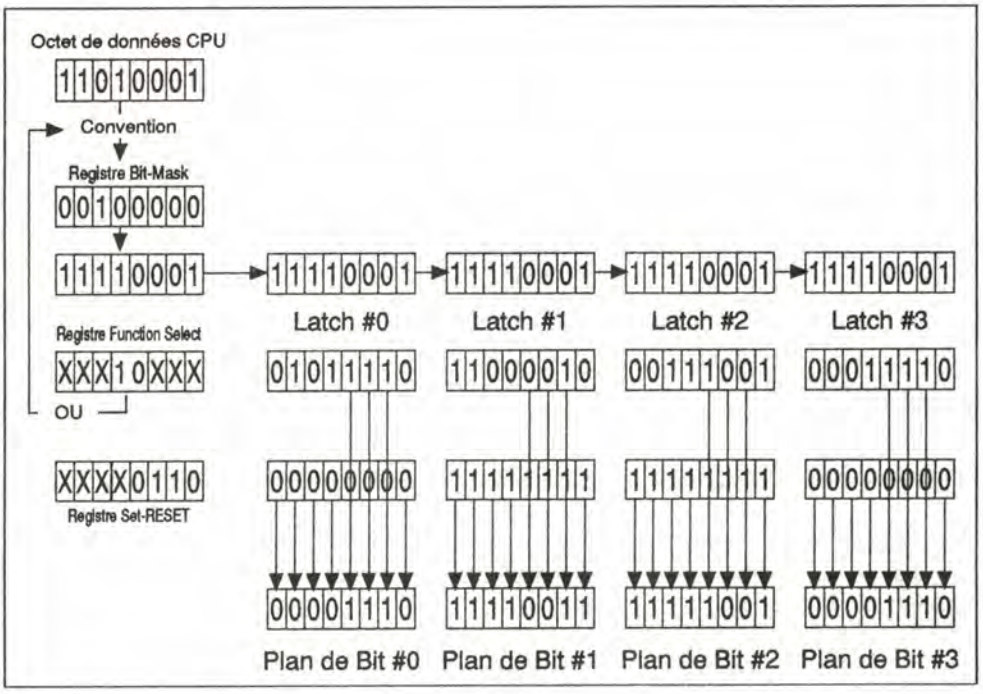
```
mov ax,0A000h       ;adresse de segment de la RAM vidéo
mov ds,ax           ;dans DS
mov al,ds:[0]       ;charger octet 0 dans le registre Latch
mov dx,3CEh         ;adresser le contrôleur graphique
mov ax,0205h        ;mode Read 0, mode Write 2
out dx,ax           ;écrire dans le registre Mode
mov ax,0003h        ;écrire mode REPLACE (0) dans
out dx,ax           ;le registre Function Select
mov ax,0408h        ;écrire masque de bits dans
out dx,ax           ;le registre Bit Mask
mov byte ptr ds:[0],0Bh    ;nouveau code couleur dans la RAM vidéo
```

Mode Write 3

Si les trois modes Write de la carte EGA ont posé tant d'énigmes aux développeurs de logiciels, un autre mode mystérieux leur est tombé sur la tête avec l'apparition des cartes VGA et qui n'est disponible que sur ces cartes. Il se définit comme les modes précédents, ce qui consiste à écrire son numéro dans le champ de bits approprié du registre Mode du contrôleur graphique.

Lors d'un accès en écriture à la RAM vidéo dans ce mode, les quatre bits inférieurs du registre Set/Reset sont d'abord combinés avec les divers bits des quatre registres Latch. Contrairement au mode Write 0, ici le contenu du registre Enable Set/Reset ne joue aucun rôle. Mais le type de la combinaison est défini comme à l'accoutumée à l'aide de la fonction Select Register.

Dans ce mode, l'octet CPU est combiné avec le contenu du registre Bit Mask par un ET logique. Le résultat détermine les bits des registres Latch qui doivent être inscrits tels quels dans les plans de bits et ceux issus de la combinaison du registre Set/Reset avec les divers bits Latch. L'octet CPU assume le même rôle que celui du registre Bit Mask en mode Write 0 et 2.



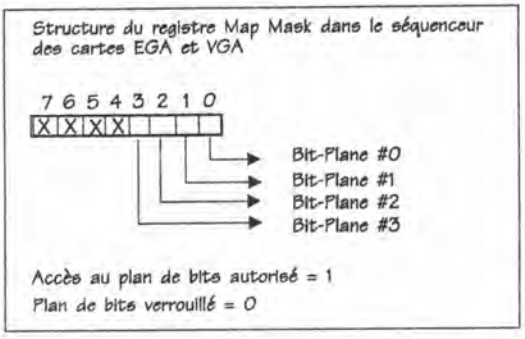
Accès en écriture à la RAM vidéo en mode Write 3

Nous n'avons pas encore pu déceler une utilisation intéressante de ce mode, c'est pourquoi nous ne vous présentons pas un exemple Assembleur dans ce contexte.

Le registre Map Mask

Il reste à évoquer un dernier registre qui agit comme une épée de Damoclès sur tous les modes Write : le registre Map Mask provenant du séquenceur. Il permet de verrouiller les différents plans de bits pour qu'ils soient protégés contre des accès en écriture et en lecture. Cela est utile lorsqu'il s'agit uniquement de manipuler le contenu d'un plan de bit déterminé.

Dans ce registre, le statut des plans de bits est représenté par les quatre bits inférieurs. Chacun d'entre eux correspond à un plan de bits et doit être mis sur 1 pour autoriser l'accès au plan de bits concerné.



Les sections suivantes montrent comment utiliser concrètement les divers modes Read et Write pour placer par exemple des points dans les modes graphiques ou pour les consulter, pour copier des zones écran à l'intérieur de la RAM vidéo ou entre la RAM vidéo et la mémoire principale.

4.8.5. Les modes graphiques 16 couleurs des cartes EGA et VGA

Le principal pouvoir acquis par la carte EGA est sans conteste les modes graphiques 16 couleurs disponibles dans les résolutions tournant autour de 320*200 et 640*350 points. Si on doit se contenter de quatre couleurs avec une carte CGA, les 16 couleurs de la carte EGA représentent déjà un énorme progrès. Mais elle reste encore fort éloignée du but final consistant à atteindre un affichage similaire à la photographie. Plus de 16 millions de couleurs différentes sont nécessaires à cet effet mais un premier pas est déjà franchi.

Cette section décrit les divers modes graphiques des cartes EGA et VGA où on peut afficher simultanément 16 couleurs. Voici les questions auxquelles nous allons répondre :

- ✓ Pourquoi une variété de modes graphiques 16 couleurs ?
- ✓ Comment la RAM vidéo est-elle construite dans les modes graphiques 16 couleurs ?
- ✓ Comment les différents points peuvent-ils être adressés dans ces modes ?

Pourquoi une variété de modes graphiques 16 couleurs

Si l'on considère l'histoire des cartes vidéo PC du point de vue des exigences toujours croissantes en matière de résolution et de couleurs, la question de savoir pourquoi la carte EGA offre deux modes graphiques 16 couleurs en résolution moindre à côté du mode hautement élevé 640*350 points vient naturellement à l'esprit. Les tableaux suivants apportent la réponse souhaitée. Dans ces modes, si la résolution est peu importante, la place mémoire nécessitée par une page écran est inférieure également. Cela permet de stocker simultanément plusieurs pages écran dans la RAM vidéo ce qui

est particulièrement appréciable dans des certaines applications. L'exemple le plus significatif est la programmation des sprites, une composante essentielle de l'animation, décrite au paragraphe 4.8.9. On y explique également comment le reste de la RAM vidéo est utilisée pour des tâches précises.

Le tableau suivant montre que les modes graphiques 16 couleurs ne sont nullement la propriété de la carte EGA puisqu'ils sont pleinement reconnus par la carte VGA également. Mais elle augmente toutefois la portée de ces modes grâce à une résolution 640*480 points. Celle-ci est largement distancée par les nombreux modes de la carte Super VGA mais elle représente toujours le mode graphique à la plus forte résolution dans le domaine des cartes standard VGA. La plupart des applications y ont d'ailleurs recours.

Les modes graphiques 16 couleurs des cartes EGA et VGA

Mode	Résolution	Mémoire	Pages	Octets restants
0Dh	320*200	32000	8	6144
0Eh	640*200	64000	4	6144
10h	640*350	112000	2	38144
12h*	640*480	153600	1	108544
* VGA seulement				

Structure de la RAM vidéo dans les modes graphiques 16 couleurs

Si on observe le tableau précédent avec les yeux d'un programmeur, on se demande immédiatement pourquoi on a mélangé toutes sortes de modes avec des résolutions tout aussi variées. La réponse est fournie par la forme d'organisation de la RAM vidéo. Dans tous les modes cités, les informations de couleur des différents points écran sont codées selon un schéma homogène, compte tenu des plus petites différences en raison de la diversité des résolutions.

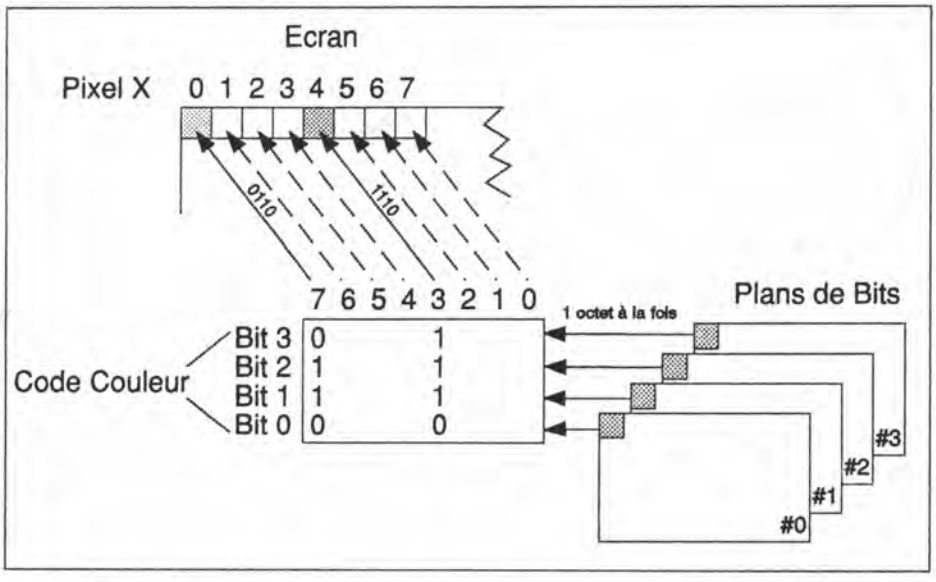
Dans chaque octet de la RAM vidéo, l'information de couleur correspondant à huit points situés côte à côte est codée dans une ligne de points. Autrement dit, chacun des huit points représente un bit distinct. Cela ne suffit nullement pour les besoins d'un affichage en 16 couleurs qui nécessite quatre bits différents par point écran. Pour obtenir ce résultat, on rassemble à chaque fois quatre octets consécutifs issus des quatre plan de bits comme le montre la figure suivante.

Les huit premiers points graphiques du coin supérieur gauche de l'écran sont représentés par exemple par les quatre octets situés à l'adresse d'offset 0000h dans les quatre plans de bits. Pour obtenir l'information de couleur nécessaire aux différents points, on

regroupe les quatre bits issus des quatre octets à une position de bit déterminée. Cette méthode crée huit groupes de quatre recevant la couleur d'un point écran.

Le bit du plan de bits #0 crée le bit 0 du code couleur, le bit du plan de bits #1 le bit 1. Le même schéma est utilisé pour créer les bits 2 et 3 du code couleur. Ces codes couleur sont convertis ensuite en noir (0), bleu (1) et vert (2) comme la méthode adoptée avec les cartes CGA.

Le mot "groupe de quatre" vous rappelle sans doute le sectionnement de la RAM vidéo en plans de bits et le fonctionnement des modes Read et Write décrits au paragraphe 4.8.5. Ici aussi, la notion de groupe de quatre joue un rôle important car toutes sortes de modes Read et Write opèrent avec ces groupes de quatre pour que les différents points deviennent accessibles dans les modes graphiques 16 couleurs à travers les registres Latch.



Structure de la RAM vidéo dans les modes graphiques 16 couleurs des cartes EGA et VGA

Mis à part la structure des différents octets, leur disposition dans la RAM vidéo joue également un rôle important pour la programmation en mode graphique. Dans les modes graphiques 16 couleurs, les cartes EGA et VGA appliquent un schéma fort simple : en commençant par le début de la RAM vidéo à l'adresse d'offset 0000h, chaque ligne de points occupe notamment un nombre d'octets consécutifs précis. Dans tous les modes graphiques avec une résolution horizontale de 640 points, on obtient par exemple 80 octets qui diminuent à 40 en mode 0D avec 320*200 points.

Les différentes lignes de points se succèdent automatiquement dans la mémoire si bien que l'adresse d'offset d'un point peut se calculer à l'aide de la formule suivante :

```
offset = Y * (résolution horizontale/8) + int(X/8)
```

L'octet de l'adresse d'offset ainsi calculée contient cependant les informations de points pour huit points consécutifs. Il faut donc isoler un bit dans les quatre octets de plans de bits pour atteindre le code couleur du point souhaité. Le numéro de ce bit résulte de la formule suivante :

```
Bit = 7 - (X mod 8)
```

Les différences entre les modes graphiques 16 couleurs ne résultent pas du codage des points mais tout simplement de la longueur des différentes lignes graphiques dans la RAM vidéo, leur nombre et donc la longueur des pages écran. Il convient naturellement d'en tenir compte pour la programmation de ces modes graphiques.

Accès aux points dans les modes graphiques 16 couleurs

Nous allons prendre deux exemples pour démontrer la conversion de ces formules et l'accès aux points à l'aide des modes Read et Write. Les programmes d'exemple sont écrits en Pascal et C, portent les noms V16COLP.PAS et V16COLC.C et se trouvent sur la disquette du livre. Ils ne démontrent pas uniquement la programmation d'un mode graphique 16 couleurs bien précis. Ils conviennent à tous les modes graphiques 16 couleurs et peuvent faire la preuve de leurs capacités dans l'un de ces modes.

Comme nous le verrons à la section 4.9, cela n'exclut pas le plus petit mode graphique des cartes Super VGA utilisant également les 16 couleurs avec une résolution de 800*600 points. Lisez la section citée pour de plus amples informations.

En raison de la forte vitesse d'exécution, les deux programmes en langage évolué sont soutenus respectivement par un module en Assembleur. Il contient les principales routines pour l'accès aux différents points et pour l'utilisation des diverses pages écran. Ces modules s'appellent V16COLCA.ASM pour le programme C et V16COLPA.ASM pour la version en Pascal.

En ce qui concerne les noms des variables globales, les noms des routines et les tâches qu'elles accomplissent, les deux modules en langage évolué ainsi que les deux modules en Assembleur sont pratiquement identiques. Nous pouvons donc nous contenter d'une présentation générale des programmes sans entrer dans les détails.

Commençons par les modules en Assembleur puisque c'est là que se réalise le travail proprement dit. Dans les déclarations PUBLIC du début des deux modules Assembleur, vous apercevez déjà les routines qui sont nécessaires pour soutenir les programmes en langage évolué. Il s'agit en particulier de quatre routines désignées par INIT640480, INIT640350, INIT640200 et INIT320200. Elles permettent de définir l'un des quatre modes graphiques 16 couleurs des cartes EGA et VGA. GETPIX et SETPIX viennent s'ajouter à ces routines. Elles permettent de fixer ou lire un point dans le mode graphique

spécifié. Les modules Assembleur se terminent par les routines SETPAGE et SHOWPAGE. D'une part, elles servent à spécifier la page écran adressable par GETPIX et SETPIX. D'autre part, elles permettent de rendre visible une page écran déterminée.

Le module Assembleur du programme C contient une autre routine appelée GETFONTPTR. Elle n'a rien à voir avec la programmation graphique puisqu'elle retourne un pointeur sur le jeu 8*8 stocké dans la ROM des cartes EGA et VGA. Ce pointeur est utile dans les deux programmes en langage évolué pour afficher en mode graphique des chiffres et des lettres sur l'écran à l'aide du jeu prédéfini. Cela concerne à la fois le programme C et le programme Pascal puisque le pointeur nécessaire sur le jeu peut être obtenu à partir du programme en langage évolué. Une routine GETFONTPTR s'avère inutile dans ce contexte ce qui explique son absence dans le module Assembleur V16COLPA.ASM.

Les routines INIT d'initialisation du mode graphique remplissent essentiellement deux fonctions : activer le mode vidéo souhaité à l'aide de la fonction 00h de l'interruption vidéo BIOS et régler trois variables globales. Celles-ci seront nécessaires pour adresser les différents points et accéder aux pages écran. L'appel de la fonction BIOS 00h efface simultanément l'écran.

Les variables citées portent les noms VIO_SEG, LARGEURL et PAGEOFS. La première variable VIO_SEG reçoit l'adresse de segment de la RAM vidéo mais représente simultanément la page écran actuelle puisque son adresse d'offset est calculée dans l'adresse de segment. Lors du calcul de l'adresse d'un point, les routines SETPIX et GETPIX ne doivent pas intégrer l'adresse de départ de la page écran actuelle dans le calcul. En revanche, elles peuvent s'en tenir à l'adresse de segment issue de VIO_SEG qui contient déjà l'adresse de départ de la page écran. Prenons un exemple :

En mode graphique 640*200 points, chaque page écran nécessite 64000 octets répartis sur les quatre plans de bits. Chaque page se succède avec un intervalle de 8000 octets à l'intérieur d'un plan de bits. En numérotation hexadécimale, cet intervalle donne 1F40h octets.

La première page écran commence à l'adre A000:0000, la seconde à A000:1F40, la troisième à A000:3E80, la quatrième à A000:5DC0 et ainsi de suite. Si on calcule ces adresses d'offset dans l'adresse de segment, les différentes pages écran commencent alors à l'adresse d'offset 0000h. En clair, la première page écran commence à A000:0000, la seconde à A1F4:0000, la troisième à A3E8:0000 et la quatrième à A5DC:0000.

Pour que le début des pages écran puisse être calculé en fonction du mode vidéo en cours, il faut naturellement spécifier la longueur d'une page. Les routines INIT stockent cette information dans les variables globales PAGEOFS où l'offset est d'abord divisé par 16 avant d'aboutir à l'adresse de segment.

La variable globale LARGEURL s'avère nécessaire pour calculer le début d'une page écran ainsi que les diverses adresses de points. La largeur des lignes de points c'est-à-dire

le facteur décrit avec l'expression "(résolution horizontale/8)" dans la formule précédente est stocké en octets dans cette variable.

A partir des informations fournies par les variables globales VIO_SEG et ZBREITE, les routines telles que SETPIX et SHOWPAGE peuvent se mettre au travail après l'appel d'une routine INIT. SETPIX et GETPIX convertissent les coordonnées écran transmises dans une adresse d'offset pour l'accès à la RAM vidéo. A cet effet, l'ordonnée Y est tout simplement multipliée par la valeur issue des variables globales LARGEURL, l'ordonnée X est ensuite divisée par huit et le résultat des deux opérations est additionné.

Dans les deux cas, la position bit du point réclamé est calculé à partir de l'ordonnée X. Les deux routines utilisent l'instruction Assembleur AND au lieu des opérandes MOD de la formule précédente pour installer tous les bits sur les trois inférieurs issus de l'ordonnée X. Cela équivaut à une opération MOD par 8 mais cette méthode est nettement plus rapide et c'est une vieille astuce connue dans la programmation graphique.

Les autres actions réalisées dans GETPIX et SETPIX étant totalement différentes, il convient de les présenter séparément. SETPIX convertit d'abord la position bit d'un point en un masque de bits utilisé par la suite pour accéder au point en mode Write 2. A cet effet, la valeur 1 est décalée vers la gauche en fonction du numéro de la position bit. Avec la position bit quatre, on obtient par exemple la valeur 0010000b.

Cette valeur est ensuite chargée dans le registre Bit Mask du contrôleur graphique puis définie dans le registre Mode du mode Write 2 ainsi que du mode Read 0. L'adresse de segment de la RAM vidéo y compris l'adresse d'offset de la page écran actuelle sont chargées dans le registre ES pour pouvoir faire référence à la RAM vidéo à travers ces valeurs.

Bien qu'un point puisse être spécifié et non lu, il faut d'abord charger l'octet contenant le point à adresser. La valeur transmise à l'unité centrale importe peu car l'intérêt de l'opération consiste seulement à charger les quatre registres Latch avec les quatre octets des plans de bits. La couleur du point à adresser se trouve codée dans ces derniers. La couleur est fixée lorsque le code couleur est tout simplement inscrit dans la RAM vidéo en tant que valeur quatre bits.

Compte tenu de la conception du mode Write 2, la carte EGA ou VGA inscrit d'abord cette valeur bit dans tous les groupes de quatre constitués de bits à l'intérieur des registres Latch dont la position bit correspondante contient un 1 dans le registre Bit Mask. Du moment qu'il ne s'agit ici que de la position bit du point à adresser, tous les groupes de quatre restants ne sont pas touchés par l'accès en écriture. Dans l'opération d'écriture qui suit immédiatement, seuls les quatre bits représentant le point écran à adresser sont modifiés dans les quatre octets de plans de bits.

Ainsi se termine la tâche de SETPIX. Mais avant de redonner le contrôle à l'utilisateur, les registres modifiés dans le contrôleur graphique sont remis sur leur valeur par défaut. Cette opération doit toujours avoir lieu à la suite d'une manipulation de ces registres

pour qu'une routine puisse se mettre en route avec les valeurs par défaut des registres et modifier seulement les registres dont le contenu ne coïncide pas exactement.

Cela concerne également GETPIX. Ici, un seul registre du contrôleur graphique a été programmé. Il s'agit du registre Read Map qui se règle automatiquement sur sa valeur par défaut à la fin de la routine. Mais la lecture d'un point avec GETPIX ne s'effectue pas aussi facilement qu'avec SETPIX. Les constructeurs de cartes EGA et VGA ayant oublié de fournir un mode Read approprié, GETPIX doit faire appel au mode Read 0.

A chaque accès en lecture à la RAM vidéo, il ne retourne qu'un seul octet issu du plan de bits dont le numéro est stocké dans le registre Read Map du contrôleur graphique. Cela explique pourquoi GETPIX exécute quatre fois la même boucle pour lire l'octet dans l'un des quatre plan de bits contenant le point à adresser. Mais avant de lire l'octet GETPIX programme le registre Read Map à l'intérieur de cette boucle.

L'accès porte d'abord sur le plan de bits 3, puis le plan de bits 2, le plan de bits 1 et en dernier lieu le plan de bits 0. En fin d'exécution de la boucle, le registre Read Map contient à nouveau sa valeur par défaut, soit 0.

Il reste à reproduire la couleur du point souhaité dans la boucle à partir des quatre octets lus. Tous les bits ne faisant pas partie du point souhaité sont d'abord masqués dans l'octet lu au moyen d'un masque de bits préalablement construit. En guise de résultat, on obtient un octet dans lequel un bit est mis ou rien du tout. Le dernier cas signifie que le bit concerné issu du code couleur du point n'était pas réglé dans le plan de bits en cours.

Le statut de ce bit est fourni par l'instruction NEG qui s'exécute une fois que les bits inutiles ont été masqués. L'essence de cette instruction est que derrière le bit 7 du registre concerné correspond exactement le statut du bit unique qui n'a pas été masqué lors de l'opération précédente. Si le seul bit de couleur a été mis, la même chose vaut pour le bit 7 de ce registre après l'opération NEG. Le bit 7 est ensuite effacé lorsque le bit de couleur contenait la valeur 0.

Le bit 7 ainsi obtenu est ensuite commuté depuis le registre BH vers le registre BL à l'aide de l'instruction ROL. Comme cette opération s'exécute successivement quatre fois, on regroupe la couleur du point souhaité dans les quatre bits inférieurs du registre BL. A chaque exécution de la boucle, le résultat obtenu décale en effet d'une position vers la gauche et s'ajoute à la position inférieure.

Il est donc important d'exécuter les quatre plans de bits depuis le plan de bits 3 jusqu'au plan de bits 0 et non dans le sens inverse. Dans ce cas, les quatre bits du code couleur risquent d'être intervertis ce qui vous fait obtenir une sorte de code couleur en vidéo inverse. Toujours est-il que GETPIX peut retourner le véritable code couleur du point souhaité même si l'opération est un peu longue et dure plus longtemps que la définition d'un point.

Les routines SETPAGE et SHOWPAGE sont nettement moins complexes que SETPIX et GETPIX. Cela concerne en particulier SETPAGE qui sert à configurer la page écran dans laquelle opèrent SETPIX et GETPIX. L'opération consiste tout simplement à multiplier le numéro de la page écran transmis avec la longueur des différentes pages spécifiées dans les variables PAGEOFS. Il faut ajouter ensuite l'adresse de segment A000h et stocker le résultat dans les variables globales VIO_SEG.

Des appels ultérieurs des routines SETPIX et GETPIX s'effectuent automatiquement sur cette page écran. Mais un appel de SETPAGE ne provoque pas automatiquement l'affichage d'une page écran sur l'écran.

Cela est conçu de telle manière que SETPIX et GETPIX puissent permettre de traiter une page en arrière-plan pendant qu'une autre est affichée. SHOWPAGE sert alors à visualiser une page ainsi traitée.

A cet effet, SHOWPAGE calcule d'abord le numéro de page transmis dans un offset à l'intérieur de la RAM vidéo. Elle se sert des variables PAGEOFS. Elle doit multiplier par 16 le résultat de la multiplication de ces variables avec le numéro de page parce que la valeur de PAGEOFS se rapporte à une adresse de segment et non d'offset comme cela est exigé dans ce contexte.

L'adresse d'offset du début d'écran doit être chargée dans deux registres du contrôleur CRT pour rendre une page écran visible sur l'écran. N'importe quelle valeur peut en principe être chargée dans ces registres comme le montre la section 4.8.3. Mais pour obtenir une image convenable sur l'écran, il faut que le début d'écran coïncide avec l'adresse ou les adresses de début que les routines SETPIX ou GETPIX considèrent comme point de départ pour le traitement d'une page écran. Cela explique l'utilité de multiplier le numéro de page par PAGEOFS.

A partir des routines Assembleur des modules V16COLPA.ASM et V16COLCA.ASM, nous arrivons aux deux programmes en langage évolué dans lesquels ces routines sont appelées. Les deux programmes peuvent fonctionner dans les modes graphiques 16 couleurs des cartes EGA et VGA. Il convient toutefois de spécifier le mode à activer avant la compilation.

Dans la partie constantes des deux programmes, vous trouvez par conséquent une constante appelée MODUS. Elle doit être affectée à l'une des constantes A320200, A640200, A640350 ou A640480. Elles représentent les divers modes graphiques.

Compte tenu de la constante spécifiée, on vérifie d'abord si le mode concerné peut effectivement être initialisé dans le programme principal. Pour le mode 640*480 points, cela signifie qu'une carte VGA doit être installée. Dans tous les autres modes, le programme se contente d'une carte EGA les cartes VGA étant naturellement acceptées.

Si la carte vidéo installé a compris ce test, les variables globales MAXX, MAXY et PAGES sont alors chargées en respectant le mode vidéo en cours. Elles reçoivent les ordonnées maximales X et Y de l'écran ainsi que le nombre de pages écran affichables.

Ces informations sont très importantes pour la routine DEMO. Elle est appelée par la suite pour démontrer le fonctionnement des diverses routines des modules Assembleur. DEMO exécute d'abord les pages écran dans une boucle, les définit à travers SETPAGE comme périphérique de sortie pour SETPIX et GETPIX et les affiche sur l'écran avec SHOWPAGE. Elle appelle ensuite les routines COLORBOX, DRAWAXIS et GRAPRINT ou GRAFPRINTF (dans la version C) à l'intérieur de cette boucle pour remplir l'écran.

COLORBOX dessine une boîte avec des traits pleins tracés avec LINE sur l'écran. LINE est réalisée dans le programme en langage évolué concerné et se base sur le fameux algorithme de Bresenham qui permet de dessiner des lignes sans utiliser l'arithmétique complexe à virgule flottante. Les différents points de la ligne sont fixés avec SETPIX.

Cette méthode de travail ne tend pas à accélérer la routine mais nous avons écrit exprès cette routine en Assembleur. Il est certes quelque peu compliqué de vouloir accélérer cette routine et il nous aurait fallu beaucoup de peine pour convertir l'algorithme de Bresenham pour optimiser le code. A notre avis, il est inutile de se donner tant de mal lorsqu'une routine LINE suffit amplement. Si on a besoin d'une telle routine dans un programme graphique de quelque nature qu'il soit, alors il existe toujours une routine pour remplir des surfaces, pour dessiner des cercles et polygônes et toutes sortes de figures géométriques.

Plusieurs semaines peuvent en effet s'écouler si vous vous mettez à étudier le développement et surtout l'optimisation de ces routines à l'aide d'ouvrages spécialisés. Nous avons donc prévu des routines pour les cartes EGA et VGA qui ne conviennent pas tout à fait à d'autres cartes graphiques. Au lieu de développer ces routines par vos propres soins, nous vous proposons d'utiliser des routines prédéfinies. A l'heure actuelle, elles se présentent non seulement sous forme de bibliothèques spéciales mais sont aussi intégrées dans les compilateurs de Borland ou Microsoft. Elles ouvrent la voie à un ensemble diversifié de routines graphiques disponibles au moyen de drivers pouvant être chargés pour de nombreux types de cartes vidéo.

Pour illustrer le caractère démo des programmes V16COLP.PAS et V16COLC.C, une routine LINE en Pascal ou C suffit amplement. LINE n'est pas la seule à fixer des points sur SETPIX mais il existe aussi GRAFPRINT ou GRAFPRINTC. Elles affichent des lettres et des chiffres sur l'écran en lisant le modèle de bits des caractères dans le jeu 8*8 stocké dans la ROM puis les écrivant point par point dans la RAM vidéo à travers SETPIX.

Bien que ces opérations ne se déroulent pas très vite, ces routines illustrent parfaitement la fonctionnalité des routines Assembleur. Vous pouvez vous en servir pour vos besoins personnels liés aux modes graphiques 16 couleurs des cartes EGA et VGA.

Listing : V16COLPA.ASM

```
;*****************************************************************;
;*                      V16COLPA.ASM                             *;
;*---------------------------------------------------------------*;
;*   Fonction : contient diverses routines pour travailler dans les *;
;*              modes graphiques à 16 couleurs des cartes EGA et VGA *;
;*---------------------------------------------------------------*;
;*   Auteur          : MICHAEL TISCHER                           *;
;*   Développé le    :  5.12.1990                                *;
;*   Dernière MAJ    : 14.01.1991                                *;
;*---------------------------------------------------------------*;
;*   Assemblage      : MASM /mx V16COLPA     ou      TASM -mx V16COLPA  *;
;*****************************************************************;

;-- Constantes ---------------------------------------------------

SC_INDEX        = 3c4h      ;Registre d'index du contrôleur du séquenceur.
SC_MAP_MASK     = 2                     ;Numéro du registre Map-Mask
SC_MEM_MODE     = 4                 ;Numéro du registre de mode mémoire

GC_INDEX        = 3ceh          ;Registre d'index du contrôleur graphique
GC_FN_SELECT    = 3         ;Numéro du registre de sélection de fonction
GC_READ_MAP     = 4                     ;Numéro du registre Read Map
GC_GRAPH_MODE   = 5                 ;Numéro du registre de mode graphique
GC_MISCELL      = 6                     ;Numéro du registre divers
GC_BIT_MASK     = 8                 ;Numéro du registre de masque binaire

CRTC_INDEX      = 3d4h          ;Registre d'index du contrôleur d'écran
CC_MAX_SCAN     = 9     ;Numéro du registre du maximum de lignes balayées
CC_START_HI     = 0Ch                   ;Numéro du registre Hi_Start
CC_UNDERLINE    = 14h               ;Numéro du registre de soulignement
CC_MODE_CTRL    = 17h           ;Nummer du registre de contrôle de mode

DAC_WRITE_ADR   = 3C8h                          ;Adresse DAC Write
DAC_READ_ADR    = 3C7h                          ;Adresse DAC Read
DAC_DATA        = 3C9h                      ;Registres de données DAC

VERT_RETRACE    = 3DAh                  ;Registre d'état des entrées #1

;-- Segment de données -------------------------------------------

DATA    segment word public     ;à initialiser au moment de l'exécution

vio_seg     dw (?)                      ;Segment vidéo page courante
largeur1    dw (?)              ;Largeur d'une ligne de pixels en octets
pageofs     dw (?)                  ;Offset de la page dans le segment

DATA    ends

;-- Programme ----------------------------------------------------

CODE        segment byte public                 ; Segment de programme

        assume cs:code, ds:data

;-- Déclarations publiques ---------------------------------------

public    init640480                ;initialise le mode 640*480 pixels
public    init640350                ;initialise le mode 640*350 pixels
public    init640200                ;initialise le mode 640*200 pixels
public    init320200                ;initialise le mode 320*200 pixels
public    setpix                            ;Dessine un pixel
public    getpix                        ;Lit la couleur d'un pixel
public    showpage                      ;Affiche la page 0 ou 1
public    setpage                 ;Fixe la page pour setpix et getpix

;-----------------------------------------------------------------
;-- INIT640350: initialise le mode graphique EGA 640*350 en 16 couleurs
;-- Appel depuis TP: init640350;

init640350 proc near

        mov   al,10h                        ;Installe le mode 10h
        mov   cx, 28000 / 16                  ;Offset de la page

init16:  mov   bx,640/8                     ;Largeur de la ligne

init:    mov   largeur1,bx                  ;Mémorise la largeur
         mov   pageofs,cx   ;Mémorise offset de page pour adr. de seg.

         xor   ah,ah                ;Appelle la fonction 00h pour fixer
         int   10h                                       ;le mode

         mov   vio_seg,0A000h     ;Fixe le segment de la mémoire écran
         ret                                         ;Rend la main

init640350 endp                                  ;Fin de la procédure

;-----------------------------------------------------------------
;-- INIT640480 : initialise le mode graphique VGA 640*480 en 16 couleurs
;-- Appel depuis TP: init640480;

init640480 proc near

        mov   al,12h                        ;Installe le mode 12hn

              ;-- L'offset de la page n'est pas intéressant car une seule
                                                ;-- page est représentable

        jmp   init16

init640480 endp                                  ;Fin de la procédure

;-----------------------------------------------------------------
;-- INIT640200 : initialise le mode graphique EGA 640*200 en 16 couleurs
;-- Appel depuis TP: init640200;

init640200 proc near

        mov   al,0Eh                        ;Installe le mode 0Eh
        mov   bx,640/8                       ;Largeur des lignes
        mov   cx,( 64000 / 4 ) / 16           ;Offset de la page
        jmp   init16

init640200 endp                                  ;Fin de la procédure

;-----------------------------------------------------------------
;-- INIT320200 : initialise le mode graphique EGA 320*200 en 16 couleurs
;-- Appel depuis TP: init320200;

init320200 proc near

        mov   al,00h                        ;Installe le mode 00h
        mov   bx,320/8                       ;Largeur des lignes
        mov   cx,( 32000 / 4 ) / 16           ;Offset de la page
        jmp   init

init320200 endp                                  ;Fin de la procédure

;-----------------------------------------------------------------
;-- SETPIX: Dessine un pixel dans une couleur donnée
;-- Appel depuis TP: setpix( x , y : integer; couleur : byte );

setpix     proc near

sframe     struc                            ;Structure d'accès à la pile
bp0        dw ?                                        ;Mémorise BP
ret_adr0   dw ?                         ;Adresse de retour à l'appelant
couleur    dw ?                                       ;Couleur
y0         dw ?                                    ;Ordonnée Y
x0         dw ?                                    ;Abscisse X
sframe     ends                                 ;Fin de la structure

frame      equ [ bp - bp0 ]         ;Adresse les éléments de la structure

           push  bp                   ;Prépare l'adressage des paramètres
           mov   bp,sp                                        ;par BP

  ;--Calcule d'abord l'offset en mémoire d'écran et le décalage      --

        mov   ax,frame.y0                          ;Charge l'ordonnée Y
        mov   dx,largeur1                ;Multiplie par la largeur de ligne
        mul   dx
        mov   bx,frame.x0                          ;Charge l'abscisse Y
        mov   cl,bl                     ;Mémorise l'octet faible pour décalage

        shr   bx,1                              ;Divise l'abscisse X par 8
        shr   bx,1
        shr   bx,1
        add   bx,ax                     ;Ajoute offset provenant de la multipl

        and   cl,7                       ;Calcule masque binaire à partir de X
        xor   cl,7
        mov   ah,1
        shl   ah,cl

        mov   dx,GC_INDEX                  ;Accède au contrôleur graphique
        mov   al,GC_BIT_MASK           ;Charge masque bin. dans reg. masquage
        out   dx,ax

        mov   ax,(02h shl 8) + GC_GRAPH_MODE                ;mode Write 2 &
        out   dx,ax                                           ;mode Read 0

        mov   ax,vio_seg                ;Charge en ES le segment de mém écran
        mov   es,ax                                           ;avec la page
```

```
        mov   al,es:[bx]                    ;Charge le registre latch
        mov   al,byte ptr frame.couleur     ;Fixe la couleur du point
        mov   es:[bx],al                    ;Réécrit dans le registre latch

            ;-- Rétablit les valeurs par défaut dans les registres du --
                        ;-- contrôleur graphqiue qui ont été modifiés

        mov   ax,(0FFh shl 8 ) + GC_BIT_MASK
        out   dx,ax

        mov   ax,(00h shl 8) + GC_GRAPH_MODE
        out   dx,ax

        pop   bp
        ret   6                             ;Retour au programme appelant
                                            ;Retire les paramètres de la pile
setpix  endp                                ;Fin de procédure

;-------------------------------------------------------------------
;-- GETPIX : Détermine la couleur d'un pixel
;-- Appel depuis TP: x := getpix( x , y : integer );

getpix  proc near

sframe1 struc                               ;Structure d'accès à la pile
bp1     dw ?                                ;Mémorise BP
ret_adr1 dw ?                               ;Adresse de retour à l'appelant
y1      dw ?                                ;Ordonnée Y
x1      dw ?                                ;Abscisse X
sframe1 ends                                ;Fin de la structure

frame   equ [ bp - bp1 ]            ;adresse les éléments de la structure

        push bp                     ;Prépare l'adressage des paramètres
        mov  bp,sp                                  ; par BP

        : Calcule d'abord l'offset en mémoire d'écran et le décalage ---

        mov  ax,frame.y1                        ;Charge l'ordonnée Y
        mov  dx,largeur1                ;Multiplie par la largeur de ligne
        mul  dx
        mov  si,frame.x1                        ;Charge l'abscisse X
        mov  cx,si                              ;et prépare le décalage

        shr  si,1                               ;Divise X par huit
        shr  si,1
        shr  si,1
        add  si,ax              ;Ajoute offset en provenance de la multipl

        and  cl,7               ;Calcule le masque binaire à partir de X
        xor  cl,7
        mov  ch,1
        shl  ch,cl

        mov  ax,vio_seg     ;Charge en ES le segment de la mémoire écran
        mov  es,ax                                      ;avec la page

        mov  dx,GC_INDEX                ;Accède au contrôleur graphique
        mov  ax,(3 shl 8)+ GC_READ_MAP          ;Lit d'abord le plan #3
        xor  bl,bl                                      ;
gp1:    out  dx,ax                              ;Indique le plan à lire
        mov  bh,es:[si]             ;Lit le contenu du registre latch
        and  bh,ch                  ;Ne conserve que le pixel concerné
        neg  bh                             ;Fixe le bit 7 du pixel
        rol  bx,1       ;Effectue une rot bit 7 de BH dans bit 1 de BL

        dec  ah                 ;Prépare le traitement du plan suivant
        jge  gp1                  ;sup ou égal à 0 ? ---> on continue

        mov  al,bl                      ;Résultat de la fonction en AL

        pop  bp
        ret  4                      ;Retourne à l'appelant en retirant
                                            ; les paramètres de la pile
getpix  endp                                ;Fin de la procédure

;-------------------------------------------------------------------
;-- SETPAGE : Sélectionne la page concernée par les appels aux fonctions
;--         setpix et getpix
;-- Appel depuis TP: setpage( page : byte );

setpage proc near

        pop  cx                             ;Dépile l'adresse de retour
        pop  ax                                         ;et l'argument

        push cx                 ;Empile à nouveau la seule adresse de retour
        mul  pageofs            ;Multiplie numéro de page par offset de page

        add  ax,0A000h                      ;Ajoute le segment de base
        mov  vio_seg,ax             ;Mémorise la nouvelle adr. de segment

        ret                                 ;Retourne à l'appelant, l'argument
                                            ;ayant déjà été retiré de la pile
setpage endp                                ;Fin de la procédure

;-------------------------------------------------------------------
;-- SHOWPAGE : Affiche l'une des pages d'écran disponibles
;-- Appel depuis TP: showpage( page : byte );

showpage proc near

        pop  cx                             ;Dépile l'adresse de retour
        pop  ax                                     ;ainsi que l'argument

        push cx                 ;Empile à nouveau la seule adresse de retour
        mul  pageofs            ;Multiplie le numéro de la page par l'offset
        mov  cl,4                           ;de la page et le tout par 16
        shl  ax,cl

        mov  bl,al                      ;Mémorise l'octet de poids faible

        mov  dx,CRTC_INDEX              ;Adresse le contrôleur d'écran
        mov  al,CC_START_HI             ;Affiche d'abord l'octet fort
        out  dx,ax
        inc  al                                     ;puis l'octet faible
        mov  ah,bl
        out  dx,ax

            ;-- Attend le début d'un rafraîchissement d'écran -----

        mov  dx,VERT_RETRACE        ;Attend la fin du balayage vertical
sp3:    in   al,dx
        test al,8
        jne  sp3

sp4:    in   al,dx                  ;puis le début du retour du faisceau
        test al,8
        je   sp4

        ret                                 ;Retourne à l'appelant

showpage endp                               ;Fin de la procédure

;== Fin ============================================================

CODE    ends                                ;Fin du segment de code
        end                                         ;Fin du programme
```

Listing : V16COLP.PAS

```
{*********************************************************************
*                        V16COLP.PAS                                 *
**--------------------------------------------------------------------**
*  Fonction : Montre comment programmer les modes graphiques à 16     *
*             couleurs des cartes EGA et VGA.                         *
*             Ce programme utilise les routines en assembleur du      *
*             module V16COLPA.ASM                                     *
**--------------------------------------------------------------------**
*  Auteur           : MICHAEL TISCHER                                 *
*  Développé le     : 20/12/1990                                      *
*  dernière MAJ     : 14/01/1991                                      *
*********************************************************************}

program V16COLP;

uses dos, crt;

{-- Déclarations de types ------------------------------------------}

type BPTR = ^byte;

{-- Références externes aux routines en assembleur-----------------}

{$L v16colpa}     { Inclut le module en assembleur }

procedure Init640480; external;
procedure Init640350; external;
procedure Init320200; external;
procedure Init640200; external;
procedure setpix( x, y : integer; couleur : byte ); external;
function  getpix( x, y: integer ) : byte ; external;
procedure setpage( page : integer ); external;
procedure showpage( page : integer ); external;

{-- Constantes ----------------------------------------------------}

const A320200 = 1;                      { Résolutions et modes possibles }
      A640200 = 2;
      A640350 = 3;
      A640480 = 4;

      MODUS   = A640350;          { Mettez ici la constante exprimant }
                                  {  le mode souhaité }

{-- Déclarations de types -----------------------------------------}

type CARTE = ( EGA, VGA, NINI );        { Type de carte vidéo installée}

{-- Variables globales --------------------------------------------}

var MaxX,                                   { Coordonnées maximales }
    MaxY  : integer;
    Pages : byte;                           { Nombre de pages d'écran }

{*********************************************************************
*  IsEgaVga : Teste la présence d'une carte EGA ou VGA .             *
**--------------------------------------------------------------------**
*  Entrée : néant                                                    *
*  Sortie  : EGA, VGA ou NINI                                        *
*********************************************************************}

function IsEgaVga : CARTE;

var Regs : Registers;          { Registres pour gérer les interruptions }

begin
  Regs.AX := $1a00;                { La fonction 1Ah n'existe qu'en VGA }
  Intr( $10, Regs );
  if ( Regs.AL = $1a ) then        { La fonction est-elle disponible ? }
    IsEgaVga := VGA
  else
    begin
      Regs.ah := $12;                           { Appelle l'option $10 }
      Regs.bl := $10;                           { de la fonction $12 }
      intr($10, Regs);          { Déclenche l'interruption vidéo du BIOS}
      if ( Regs.bl <> $10 ) then IsEgaVga := EGA
                            else IsEgaVga := NINI;
    end;
end;

{*********************************************************************
*  PrintChar : Affiche un caractère en mode graphique                *
**--------------------------------------------------------------------**
*  Entrée :   caractere = le caractère à afficher                    *
*             x, y      = Coordonnées du coin sup gauche             *
*             cc        = Couleur du caractère                       *
*             cf        = Couleur du fond                            *
*  Info   : Le caractère est dessiné dans une matrice de 8*8 pixels  *
*           sur la base du jeu de caractères 8*8 en ROM              *
*********************************************************************}

procedure PrintChar( caractere : char; x, y : integer; cc, cf : byte );

type CARADEF = array[0..255,0..7] of byte; { Struct. du jeu de caract. }
     CARAPTR = ^CARADEF;                   { Pointe sur le jeu de caractères }

var  Regs  : Registers;        { Registres pour gérer les interruptions }
     ch    : char;                                    { Pixel du caractère }
     i, k,                                    { Compteur d'itérations }
     Masque : byte;          { Masque binaire pour dessiner le caractère}

const fptr : CARAPTR = NIL;   { Pointe sur le jeu de caractères en ROM }

begin
  if fptr = NIL then              { A-t-on déjà déterminé ce pointeur ? }
    begin                                                      { Non}
      Regs.AH := $11;                    { Appelle l'option $1130 de la }
      Regs.AL := $30;                                { fonction vidéo du BIOS }
      Regs.BH := 3;              { pour obtenir un pointeur sur le jeu 8*8 }
      intr( $10, Regs );
      fptr := ptr( Regs.ES, Regs.BP );           { Compose le pointeur }
    end;

  if ( cf = 255 ) then                          { Caractère transparent ? }
    for i := 0 to 7 do { Oui ne dessine que les pixels du premier plan }
      begin
        Masque := fptr^[ord(caractere),i];
                                          { Lit le motif binaire pour une ligne }
        for k := 0 to 7 do
          begin
            if ( Masque and 128 <> 0 ) then        { Pixel à dessiner? }
              setpix( x+k, y+i, cc );                                { Oui }
            Masque := Masque shl 1;
          end;
      end
  else                                          { Non, tient compte du fond }
    for i := 0 to 7 do                              { Parcourt les lignes }
      begin
        Masque := fptr^[ord(caractere),i];
                                          {Lit le motif binaire pour une ligne }
        for k := 0 to 7 do
          begin
            if ( Masque and 128 <> 0 ) then          { Premier plan ? }
              setpix( x+k, y+i, cc )                               { Oui }
            else
              setpix( x+k, y+i, cf );                          { Non, fond}
            Masque := Masque shl 1;
          end;
      end;
end;

{*********************************************************************
*  Line : Trace un segment dans la fenêtre graphique en appliquant   *
*         l'algorithme de Bresenham                                  *
**--------------------------------------------------------------------**
*  Entrée : X1, Y1  = Coordonnées de l'origine (0 - ...)             *
*           X2, Y2  = Coordonnées de l'extrémité terminale           *
*           COULEUR = couleur du segment                             *
*********************************************************************}

procedure Line( x1, y1, x2, y2 : integer; couleur : byte );

var d, dx, dy,
    aincr, bincr,
    xincr, yincr,
    x, y                : integer;

{-- Procédure accessoire pour échanger deux variables entières --------}

procedure SwapInt( var i1, i2: integer );

var dummy : integer;

begin
  dummy := i2;
  i2    := i1;
  i1    := dummy;
end;

{-- Procédure principale -----------------------------------------}

begin
  if ( abs(x2-x1) < abs(y2-y1) ) then  (Sens : par l'axe des X ou des Y)
    begin                                                 ( par l'axe des Y )
```

```
    if ( y1 > y2 ) then                          { y1 supérieur à y2 ? }
      begin
    Swapint( x1, x2 );                           { Oui, échange X1 et X2, }
    Swapint( y1, y2 );                                    { Y1 et Y2 }
      end;

    if ( x2 > x1 ) then xincr := 1               { Fixe le pas horizontal }
                   else xincr := -1;

    dy := y2 - y1;
    dx := abs( x2-x1 );
    d  := 2 * dx - dy;
    aincr := 2 * (dx - dy);
    bincr := 2 * dx;
    x := x1;
    y := y1;

    Setpix( x, y, couleur );                     { Dessine le premier point }
    for y:=y1+1 to y2 do                           { Parcourt l'axe des Y }
      begin
        if ( d >= 0 ) then
          begin
            inc( x, xincr );
            inc( d, aincr );
          end
        else
          inc( d, bincr );
        Setpix( x, y, couleur );
      end;
  end
else                                                    { par l'axe des X }
  begin
    if ( x1 > x2 ) then                          { x1 plus grand que x2? }
      begin
    Swapint( x1, x2 );                           { Oui, échange X1 et X2 }
    Swapint( y1, y2 );                                    { Y1 et Y2 }
      end;

    if ( y2 > y1 ) then yincr := 1               { Fixe le pas vertical }
                   else yincr := -1;

    dx := x2 - x1;
    dy := abs( y2-y1 );
    d  := 2 * dy - dx;
    aincr := 2 * (dy - dx);
    bincr := 2 * dy;
    x := x1;
    y := y1;

    Setpix( x, y, couleur );                     { Dessine le premier point }
    for x:=x1+1 to x2 do                           { Parcourt l'axe des X }
      begin
        if ( d >= 0 ) then
          begin
            inc( y, yincr );
            inc( d, aincr );
          end
        else
          inc( d, bincr );
        Setpix( x, y, couleur );
      end;
  end;
end;

{*****************************************************************
 * GrafPrint : Affiche une chaîne formatée sur l'écran graphique *
 *---------------------------------------------------------------*
 * Entrées : X, Y   = Coordonnées de départ (0-...)              *
 *           CC     = Couleur des caractères                     *
 *           CF     = Couleur du fond    (255 = transparent)     *
 *           STRING = Chaîne avec indications de formatage       *
 *****************************************************************}

procedure GrafPrint( x, y : integer; cc, cf : byte; strt : string );

var i : integer;                                  { Compteur d'itérations }

begin
  for i:=1 to length( strt ) do
    begin
      printchar( strt[i], x, y, cc, cf );        { Affiche par printchar }
      inc( x, 8 );                    { x à la position du caractère suivant }
    end;
end;

{*****************************************************************
 * ColorBox : Dessine un rectangle et le remplit avec un motif composé *
 *            lignes.                                            *
 *---------------------------------------------------------------*
 * Entrées : X1, Y1  = Coordonnées du coin sup gauche de la fenêtre *
 *           X2, Y2  = Coordonnées du coin inf droit de la fenêtre *
 *           COULMAX = Code de couleur maximal                   *
 * Info    : Les couleurs des lignes sont répétées dans un cycle de 0 *
 *           à COULMAX                                           *
 *****************************************************************}

procedure ColorBox( x1, y1, x2, y2 : integer; coulmax : byte );

var x, y,                                         { Compteur d'itérations }
    sx, sy : integer;             { Point de départ de la dernière boucle }

begin
  Line( x1, y1, x1, y2, 15 );                  { Cadre autour du rectangle }
  Line( x1, y2, x2, y2, 15 );
  Line( x2, y2, x2, y1, 15 );
  Line( x2, y1, x1, y1, 15 );

  for y := y2-1 downto y1+1 do          { Du coin inf gauche au bord droit }
    Line( x1+1, y2-1, x2-1, y, y mod coulmax );

  for y := y2-1 downto y1+1 do          { Du coin inf droit au bord gauche }
    Line( x2-1, y2-1, x1+1, y, y mod coulmax );

  {-- Du milieu du rectangle au bord supérieur ----------------------}

  sx := x1+ (x2-x1) div 2;
  sy := y1+ (y2-y1) div 2;
  for x := x1+1 to x2-1 do
    Line( sx, sy, x, y1+1, x mod coulmax );
end;

{*****************************************************************
 * DrawAxis: Dessine des axes en haut et à gauche de l'écran     *
 *---------------------------------------------------------------*
 * Entrées : XSTEP = Pas selon l'axe X                           *
 *           YSTEP = Pas selon l'axe Y                           *
 *           CC    = Couleur des caractères                      *
 *           CF    = Couleur de fond    (255 = transparent)      *
 *****************************************************************}

procedure DrawAxis( stepx, stepy : integer; cc, cf : byte );

var x, y     : integer;                          { Variables d'itération }
    grad : string[3];

begin
  Line( 0, 0, MAXX, 0, cc );                                { Trace l'axe X }
  Line( 0, 0, 0, MAXY, cc );                                { Trace l'axe Y }

  x := stepx;                                               { Gradue l'axe X }
  while ( x < MAXX ) do
    begin
      Line( x, 0, x, 5, cc );
      str( x, grad );
      if ( x < 100 ) then
        GrafPrint( x - 8 , 8, cc, cf, grad )
      else
        GrafPrint( x - 12, 8, cc, cf, grad );
      inc( x, stepx );
    end;

  y := stepy;                                               { Gradue l'axe Y }
  while ( y < MAXY ) do
    begin
      Line( 0, y, 5, y, cc );
      str( y:3, grad );
      GrafPrint( 8, y-4, cc, cf, grad );
      inc( y, stepy );
    end;
end;

{*****************************************************************
 * Demo : Démontre l'usage des différentes fonctions de ce module *
 *---------------------------------------------------------------*
 * Entrée : néant                                                *
 *****************************************************************}

procedure Demo;

const PAUSE = 100;                                { Pause en millisecondes }

var page  : byte;                                   { Compteur de pages }

begin
  for page := 1 to Pages do
    begin
      setpage( page-1 );                                { Traite une page }
      showpage( page-1 );
      ColorBox( 50+page*2, 40, MaxX-50+page*2, MaxY-40, 16 );
      DrawAxis( 30, 20, 15, 255 );                        { Trace les axes }
      GrafPrint( 46, MAXY-10, 15, 255,
                 'V16CDLP  -  (c) by MICHAEL TISCHER' );
    end;

  {-- Affiche les pages graphiques en alternance --------------------}
```

```
for page := 0 to 50 do                                  { 50 passages }
  begin
    showpage( page mod Pages );                         { Affiche une page }
    delay( PAUSE );                                     { Brève pause }
  end;
end;

{------------------------------------------------------------------------}
{--                        PROGRAMME PRINCIPAL                         ----}
{------------------------------------------------------------------------}

begin
 writeln( 'V16COLP.PAS - (c) 1990 by MICHAEL TISCHER'#13#10 );
 if ( MODUS = A640480 ) then                            { Mode VGA? }
   begin                                                { Oui}
     if ( IsEgaVga <> VGA ) then                        { Carte VGA ? }
       begin                                            { Non  }
         writeln( 'Attention ! Pour travailler dans le mode graphique' +
                  'de'#13#10' 640*480 pixels il faut une carte VGA!');
         exit;                                          { Terminé }
       end
     else               { Oui, initialise le mode et fixe les paramètres}
       begin                                            { 640*480 pixels }
         MaxX := 639;
         MaxY := 479;
         Pages := 1;
         init640480;
       end;
   end
 else                                           { doit être un mode EGA}
   begin
     if ( IsEgaVga = MINI ) then                { Pas de carte EGA ou VGA ? }
       begin                                            { Non }
         writeln( 'Attention: Pour faire tourner ce programme il faut' +
                  'au moins '#13#10'une carte EGA !');
         exit;                                          { Terminé }
       end
     else               { Oui, initialise le mode et fixe les paramètres }
       case MODUS of
         A320200 : begin                                { 320*200 pixels }
                     MaxX := 319;
                     MaxY := 199;
                     Pages := 8;
                     init320200;
                   end;
         A640200 : begin                                { 640*200 pixels }
                     MaxX := 639;
                     MaxY := 199;
                     Pages := 4;
                     init640200;
                   end;
         A640350 : begin                                { 640*350 pixels }
                     MaxX := 639;
                     MaxY := 349;
                     Pages := 2;
                     init640350;
                   end;
       end;
   end;

 Demo;                                          { Lance la démonstrations }
 repeat until keypressed;                       { Attend une frappe de touche }
 Textmode( C080 );                              { Revient au mode texte }
end.
```

Listing : V16COLCA.ASM

```
;*************************************************************************
;*                          V 1 6 C O L C A . A S M                      *
;*-----------------------------------------------------------------------*
;*   Fonction       : contient diverses routines pour travailler         *
;*                    dans les modes graphiques à 16 couleurs            *
;*                    des cartes EGA et VGA                              *
;*-----------------------------------------------------------------------*
;*   Auteur         : MICHAEL TISCHER                                    *
;*   Développé le   :  5.12.1990                                         *
;*   Dernière MAJ   : 14.02.1992                                         *
;*-----------------------------------------------------------------------*
;*   Modèle mémoire : SMALL                                              *
;*-----------------------------------------------------------------------*
;*   Assemblage     : MASM /mx V16COLCA     ou     TASM -mx V16COLCA     *
;*************************************************************************

IGROUP group _text                      ; Regroupe les segments de programme
DGROUP group _bss, _data

        assume CS:IGROUP, DS:DGROUP, ES:DGROUP, SS:DGROUP

_BSS    segment word public 'BSS'       ; Segment pour toutes les variables
_BSS    ends                                    ; statiques non initialisées

_DATA   segment word public 'DATA'      ; Segment pour toutes les variables
                                        ; statiques et globales initialisées

_DATA   ends

;-- Constantes ---------------------------------------------------------

SC_INDEX        = 3c4h  ; Registre d'index du contrôleur du séquenceur.
SC_MAP_MASK     = 2                     ; Numéro du registre Map-Mask
SC_MEM_MODE     = 4                     ; Numéro du registre de mode mémoire

GC_INDEX        = 3ceh          ; registre d'index du contrôleur graphique
GC_FN_SELECT    = 3             ; Numéro du registre de sélection de fonction
GC_READ_MAP     = 4                     ; Numéro du registre Read Map
GC_GRAPH_MODE   = 5                     ; Numéro du registre de mode graphique
GC_MISCELL      = 6                     ; Numéro du registre divers
GC_BIT_MASK     = 8             ; Numéro du registre de masquage binaire

CRTC_INDEX      = 3d4h          ; registre d'index du contrôleur d'écran
CC_MAX_SCAN     = 9             ; Numéro du registre de lignes balayées
CC_START_HI     = 0Ch                   ; Numéro du registre Hi_Start
CC_UNDERLINE    = 14h           ; Numéro du registre de soulignement
CC_MODE_CTRL    = 17h           ; Nummer du registre de contrôle de mode

DAC_WRITE_ADR   = 3C8h                  ; Adresse DAC Write
DAC_READ_ADR    = 3C7h                  ; Adresse DAC-Read
DAC_DATA        = 3C9h          ; Registres de données DAC

VERT_RETRACE    = 3DAh          ; Registre d'état des entrées #1

PIXX            = 640                   ; Résolution horizontale

;-- Données ------------------------------------------------------------

_DATA  segment word public 'DATA'

vio_seg     dw 0A000h                   ; Segment vidéo page courante
largeur1    dw 0            ; Largeur d'une ligne de pixels en octets
pageofs     dw 0                ; Offset de la page dans le segment

_DATA  ends

;-- Programme ----------------------------------------------------------

_TEXT  segment byte public 'CODE'               ; Segment de programme

;-- Déclarations publiques ---------------------------------------------

public  _init640350             ; initialise le mode 640*350 pixels
public  _init640480             ; initialise le mode 640*480 pixels
public  _init640200             ; initialise le mode 640*200 pixels
public  _init320200             ; initialise le mode 320*200 pixels
public  _setpix                         ; Dessine un pixel
public  _getpix                 ; Lit la couleur d'un pixel
public  _showpage                       ; Affiche la page 0 ou 1
public  _setpage            ; Fixe la page pour setpix et getpix
public  _getfontptr         ; retourne un pointeur sur le jeu 8*8

;-----------------------------------------------------------------------
;-- INIT640350  : Mode graphique EGA en 16 couleurs de 640*350 pixels
;-- Déclaration : void init640350( void );

_init640350 proc near

        mov   al,10h                    ; Fixe le mode 10h
        mov   cx,28000/16               ; Offset de la page

init16:
        mov   bx,640/8                  ; Largeur de la ligne
init:   mov   largeur1,bx               ; Mémorise la largeur
        mov   pageofs,cx        ; Mémorise l'offset de la page pour
                                        ; l'adresse de segment
```

```
        xor   ah,ah                     ; Appelle la fonction 00h pour fixer
        int   10h                                          ; le mode

        ret                                              ; Rend la main

_init640350 endp                                  ; Fin de la procédure

;---------------------------------------------------------------
;-- INIT640480: Initialise le mode graphique VGA 640*480 en 16 couleurs
;-- Déclaration : void init640480( void );

_init640480 proc near

        mov   al,12h                          ; Installe le mode 12hn

        ;-- L'offset de la page n'est pas intéressant car une seule page-
        ;-- est représentable ----------------------------------------
        jmp   init16

_init640480 endp                                  ; Fin de la procédure

;-- INIT640200: initialise le mode graphique EGA 640*200 en 16 couleurs
;-- Déclaration: void init640200 (void):

_init640200 proc near

        mov   al,0Eh                          ; Installe le mode 0Eh
        mov   bx,640/8                          ; Largeur des lignes
        mov   cx,(64000/4)/16                    ; Offset de la page
        jmp   init16

_init640200 endp                                  ; Fin de la procédure

;---------------------------------------------------------------
;-- INIT320200: initialise le mode graphique EGA 320*200 en 16 couleurs
;-- Déclaration : void init320200( void );

_init320200 proc near

        mov   al,0Dh                          ; Installe le mode 0Dh
        mov   bx,320/8                          ; Largeur des lignes
        mov   cx,(32000/4)/16                    ; Offset de la page
        jmp   init

_init320200 endp                                  ; Fin de la procédure

;-- SETPIX: Dessine un pixel dans une couleur donnée ----------------
;-- Déclaration : void setpix( int x, int y, unsigned char farbe );

_setpix   proc near

sframe    struc                           ; Structure d'accès à la pile
bp0       dw ?                                      ; Mémorise BP
ret_adr0  dw ?                        ; Adresse de retour à l'appelant
x0        dw ?                                      ; Abscisse X
y0        dw ?                                      ; Ordonnée Y
couleur   dw ?                                       ; Couleur
sframe    ends                                  ; Fin de la structure

frame  equ [ bp - bp0 ]           ; Adresse les éléments de la structure

        push  bp                        ; Prépare l'adressage des paramètres
        mov   bp,sp                                        ; par BP

        ;--Cacule d'abord l'offset en mémoire d'écran et le décalage ----

        mov   ax,frame.y0                        ; Charge l'ordonnée Y
        mov   dx,largeur1               ; Multiplie par la largeur de ligne
        mul   dx
        mov   bx,frame.x0                        ; Charge l'abscisse X
        mov   cl,bl                       ; Octet faible pour le décalage

        shr   bx,1                           ; Divise l'abscisse X par 8
        shr   bx,1
        shr   bx,1
        add   bx,ax                     ; + offset de la multiplication.

        and   cl,7                   ; Calcule masque binaire à partir de X
        xor   cl,7
        mov   ah,1
        shl   ah,cl

        mov   dx,GC_INDEX                ; Accède au contrôleur graphique
        mov   al,GC_BIT_MASK        ; Charge masque bin dans reg de masque
        out   dx,ax

        mov   ax,(02h shl 8)+GC_GRAPH_MODE
                                    ; Déclenche le mode Write 2 & le mode
        out   dx,ax                                        ; Read 0

        mov   ax,vio_seg                ; ES=segment de la mémoire d'écran
        mov   es,ax                                    ; avec la page
        mov   al,es:[bx]                      ; Charge le registre latch
        mov   al,byte ptr frame.couleur
                                                ; Fixe la couleur du point
        mov   es:[bx],al                ; Réécrit dans le registre latch

        ;-- Rétablit les valeurs par défaut dans les registres du -------
        ;-- contrôleur graphique qui ont été modifiés -------------------

        mov   ax,(0FFh shl 8 ) + GC_BIT_MASK
        out   dx,ax

        mov   ax,(00h shl 8) + GC_GRAPH_MODE
        out   dx,ax

        pop   bp
        ret                                  ; Retour au programme appelant

_setpix   endp                                    ; Fin de la procédure

;-- GETPIX: Détermine la couleur d'un pixel  ------------------------
;-- Déclaration : unsigned char getpix( int x, int y );

_getpix   proc near

sframe1   struc                           ; Structure d'accès à la pile
bp1       dw ?                                      ; Mémorise BP
ret_adr1  dw ?                        ; Adresse de retour à l'appelant
x1        dw ?                                      ; Abscisse X
y1        dw ?                                      ; Ordonnée Y
sframe1   ends                                  ; Fin de la structure

frame  equ [ bp - bp1 ]           ; adresse les éléments de la structure

        push  bp                        ; Prépare l'adressage des paramètres
        mov   bp,sp                                        ; par BP

        push  si

        ;--Calcule d'abord l'offset en mémoire d'écran et le décalage ---

        mov   ax,frame.y1                        ; Charge l'ordonnée Y
        mov   dx,largeur1               ; Multiplie par la largeur de ligne
        mul   dx
        mov   si,frame.x1                        ; Charge l'abscisse X
        mov   cx,si                           ; et prépare le décalage

        shr   si,1                                ; Divise X par huit
        shr   si,1
        shr   si,1
        add   si,ax             ; +offset en provenance de la multiplication

        and   cl,7                   ; Calcule le masque binaire à partir de X
        xor   cl,7
        mov   ch,1
        shl   ch,cl

        mov   ax,vio_seg             ; Charge en ES le segment de la mémoire
        mov   es,ax                                    ; d'écran avec la page

        mov   dx,GC_INDEX                ; Accède au contrôleur graphique
        mov   ax,(3 shl 8)+ GC_READ_MAP
                                                ; Lit d'abord le plan #3
        xor   bl,bl

gp1:    out   dx,ax                               ; Indique le plan à lire
        mov   bh,es:[si]                 ; Lit le contenu du registre latch
        and   bh,ch                       ; Ne conserve que le pixel concerné
        neg   bh                                  ; Fixe le bit 7 du pixel
        rol   bx,1                        ; Effectue une rotation du bit 7 de BH
                                                  ; dans le bit 1 de BL
        dec   ah                          ; Prépare le traitement du plan suivant
        jge   gp1                           ; sup ou égal à 0 ? --> on continue

        mov   al,bl                               ; Résultat de la fonction en AL

        pop   si
        pop   bp
        ret                                          ; Retourne à l'appelant

_getpix   endp                                    ; Fin de la procédure

;-- SETPAGE: Sélectionne la page concernée par les appels aux fonctions-
;--          setpix et  getpix
;-- Déclaration : void setpage( int page );

_setpage  proc near

        pop   cx                                ; Dépile l'adresse de retour
        pop   ax                                              ; et l'argument

        push  ax                                ; puis remet tout sur la pile
        push  cx
        mul   pageofs                     ; Multiplie le numéro de page par
                                                  ; l'offset de la page
```

```
        add   ax,0A000h                 ; Ajoute le segment de base
        mov   vio_seg,ax        ; Mémorise la nouvelle adresse de segment

        ret                             ; Retourne à l'appelant l'argument ayant
                                        ; déjà été retiré de la pile

_setpage   endp                                 ; Fin de la procédure

;-- SHOWPAGE: Affiche l'une des pages d'écran disponibles------
;-- Déclaration : void showpage( int page );

_showpage  proc near

        pop   cx                        ; Dépile l'adresse de retour
        pop   ax                            ; ainsi que l'argument

        push  ax                        ; Remet tout sur la pile
        push  cx

        mul   pageofs           ; Multiplie le numéro de la page par
                                        ; l'offset de la page
        mov   cl,4                          ; et le tout par 16
        shl   ax,cl

        mov   bl,al                     ; Mémorise l'octet faible

        mov   dx,CRTC_INDEX             ; Adresse le contrôleur d'écran
        mov   al,CC_START_HI                ; Emet d'abord l'octet fort
        out   dx,ax
        inc   al                            ; puis l'octet faible
        mov   ah,bl
        out   dx,ax

        ;--Attend le début d'un rafraîchissement d'écran ---------------

        mov   dx,VERT_RETRACE           ; Attend d'abord la fin du balayage
sp3:    in    al,dx                                       ; vertical
        test  al,8
        jne   sp3

sp4:    in    al,dx                 ; puis le début du retour du faisceau
        test  al,8
        je    sp4

        ret                                       ; Retourne à l'appelant

_showpage  endp                                 ; Fin de la procédure

;-- GETFONTPTR: Retourne un pointeur FAR sur le jeu 8*8 ---------------
;-- Déclaration : void far * getfontptr( void ) ------------------------

_getfontptr proc near

        push  bp                                        ; Sauvegarde BP

        mov   ax,1130h          ; Charge les registres pour l'interruption
        mov   bh,3
        int   10h                       ; Déclenche l'interruption vidéo

        mov   dx,es                         ; Transfère ES:BP en DX:AX
        mov   ax,bp

        pop   bp                                ; Reprend BP sur la pile
        ret                                      ; Retourne à l'appelant

_getfontptr endp                                ; Fin de la procédure

; == Fin ============================================================
_text   ends                                    ; Fin du segment de programme
        end                                 ; Fin de la source en assembleur
```

Listing : V16COLC.C

```
/*****************************************************************
*                        V 1 6 C O L C . C                       *
*****************************************************************
* Fonction        : Montre comment programmer les modes graphiques *
*                   à 16 couleurs des cartes EGA et VGA           *
*                   Ce programme utilise les routines en assembleur *
*                   du module V16COLCA.ASM .                      *
*****************************************************************
* Auteur          : MICHAEL TISCHER                               *
* Développé le    : 20.12.1990                                    *
* Dernière MAJ      14.02.1992                                    *
*****************************************************************
* Modèle mémoire  : SMALL                                         *
*****************************************************************
* (MICROSOFT C)                                                   *
* Compilation     : CL /AS v16colc.c v16colca                     *
*****************************************************************
* (BORLAND TURBO C)                                               *
* Compilation     : Utiliser un fichier de projet avec le contenu *
*                   suivant:                                      *
*                      v16colc.c                                  *
*                      v16colca.obj                               *
*****************************************************************
* Appel           : v16colc                                       *
*****************************************************************
* Info            : Le message "Unreachable code ... " émis par   *
*                   Turbo C ne signifie pas qu'il y a une erreur  *
*****************************************************************/

#include <dos.h>
#include <stdarg.h>
#include <stdlib.h>
#include <io.h>
#include <stdio.h>
#include <conio.h>

/*-- Déclarations de types ----------------------------------------*/

typedef unsigned char BYTE;

/*-- Références externes aux routines en assembleur ---------------*/

extern void init320200( void );
extern void init640480( void );
extern void init640350( void );
extern void init640200( void );
extern void setpix( int x, int y, unsigned char couleur);
extern BYTE getpix( int x, int y );
extern void setpage( int page );
extern void showpage( int page );
extern void far *getfontptr( void );

/*-- Constantes ---------------------------------------------------*/

#define A320200  1                  /* Résolutions et modes possibles */
#define A640200  2
#define A640350  3
#define A640480  4

#define MODE     A640350                /* Mettez ici la constante */
                                    /* exprimant le mode souhaité */

#define EGA      0                             /* Types de cartes */
#define VGA      1
#define NINI     2

/*-- Variables globales -------------------------------------------*/

int  MaxX,                                  /* Coordonnées maximales */
     MaxY;
BYTE Pages;                                 /* Nombre de pages d'écran */

/*****************************************************************
* IsEgaVga : Teste la présence d'une carte EGA ou VGA.           *
*****************************************************************
* Entrée : néant                                                 *
* Sortie : EGA, VGA ou NINI                                      *
*****************************************************************/

BYTE IsEgaVga( void )
{
  union REGS Regs;              /* Registres pour gérer les interruptions */

  Regs.x.ax = 0x1a00;             /* La fonction 1Ah n'existe qu'en VGA */
  int86( 0x10, &Regs, &Regs );
  if ( Regs.h.al == 0x1a )          /* La fonction est-elle disponible ? */
    return VGA;
  else
    {
    Regs.h.ah = 0x12;                           /* Appelle l'option 10h */
```

```
    Regs.h.bl = 0x10;                                   /* de la fonction 12h */
    int86(0x10, &Regs, &Regs );         /* Déclenche l'interruption vidéo */
    return (BYTE) (( Regs.h.bl != 0x10 ) ? EGA : MINI);
  }
}

/*****************************************************************
*  PrintChar : Ecrit un caractère en dehors de la zone visible   *
*              de la mémoire d'écran                             *
*----------------------------------------------------------------*
*  Entrée  :   caractere = caractère à afficher                  *
*              x, y      = Coordonnées du coin supérieur gauche  *
*              cc        = Couleur du caractère                  *
*              cf        = Couleur du fond                       *
*  Info    : Le caractère est dessiné dans une matrice de 8*8 pixels *
*            sur la base du jeu de caractères 8*8 en ROM         *
*****************************************************************/

void PrintChar( char caractere, int x, int y, BYTE cc, BYTE cf )
{
 typedef BYTE CARDEF[256][8];        /* Structure du jeu de caractères */
 typedef CARDEF far *CARPTR;         /* Pointe sur un jeu de caractères */

 BYTE        i, k,                           /* Compteur d'itérations */
       masque;              /* Masque binaire pour dessiner le caractère */

 static CARPTR fptr = (CARPTR) 0;          /* Jeu de caractères en ROM */

 if ( fptr == (CARPTR) 0 )      /* A-t-on déjà déterminé ce pointeur ? */
  fptr = getfontptr();     /* détermine avec la fonction en assembleur */

  /*- Dessine le caractère pixel par pixel ---------------------------*/

 if ( cf == 255 )                            /* Caractère transparent ? */
  for ( i = 0; i < 8; ++i ) /* dessine que les pixels du premier plan */
   {
    masque = (*fptr)[caractere][i];          /* Lit le motif bin/ligne */
    for ( k = 0; k < 8; ++k, masque <<= 1 )  /* Parcourt les colonnes */
     if ( masque & 128 )                       /* Pixel à dessiner ? */
      setpix( x+k, y+i, cc );                                /* Oui */
   }
 else                                      /* Non dessine chaque pixel */
  for ( i = 0; i < 8; ++i )                   /* Parcourt les lignes */
  {
   masque = (*fptr)[caractere][i];           /* Lit le motif bin/ligne */
   for ( k = 0; k < 8; ++k, masque <<= 1 )   /* Parcourt les colonnes */
    setpix( x+k, y+i, (BYTE) (( masque & 128 ) ? cc : cf) );
  }
}

/*****************************************************************
*  Line: Trace un segment dans la fenêtre graphique en appliquant *
*        l'algorithme de Bresenham                               *
*----------------------------------------------------------------*
*  Entrées : X1, Y1 = Coordonnées de l'origine                   *
*            X2, Y2 = Coordonnées de l'extrémité terminale       *
*            COULEUR = couleur du segment                        *
*****************************************************************/

/*-- Fonction accessoire pour échanger deux variables entières -------*/

void SwapInt( int *i1, int *i2 )
{
 int dummy;

 dummy = *i2;
 *i2   = *i1;
 *i1   = dummy;
}

/*-- Procédure principale ---------------------------------------------*/

void Line( int x1, int y1, int x2, int y2, BYTE couleur )
{
 int d, dx, dy,
     aincr, bincr,
     xincr, yincr,
     x, y;

 if ( abs(x2-x1) < abs(y2-y1) )           /* Parcours : axe X ou Y ? */
  {                                                    /* Par Y */
   if ( y1 > y2 )                          /* y1 plus grand que y2? */
    {
     SwapInt( &x1, &x2 );                  /* Oui échange X1 et X2, */
     SwapInt( &y1, &y2 );                                /* Y1 et Y2 */
    }

   xincr = ( x2 > x1 ) ? 1 : -1;          /* Fixe le pas horizontal */

   dy = y2 - y1;
   dx = abs( x2-x1 );
   d  = 2 * dx - dy;
   aincr = 2 * (dx - dy);
   bincr = 2 * dx;
   x = x1;
   y = y1;

   setpix( x, y, couleur );                    /* dessine le premier pixel */
   for (y=y1+1; y<= y2; ++y )                     /* Parcourt l'axe des Y */
    {
     if ( d >= 0 )
      {
       x += xincr;
       d += aincr;
      }
     else
      d += bincr;
     setpix( x, y, couleur );
    }
  }
 else                                                      /* par X */
  {
   if ( x1 > x2 )                              /* x1 plus grand que x2? */
    {
     SwapInt( &x1, &x2 );                       /* Oui, échange X1 et X2*/
     SwapInt( &y1, &y2 );                                 /* Y1 et Y2 */
    }

   yincr = ( y2 > y1 ) ? 1 : -1;              /* Fixe le pas vertical */

   dx = x2 - x1;
   dy = abs( y2-y1 );
   d  = 2 * dy - dx;
   aincr = 2 * (dy - dx);
   bincr = 2 * dy;
   x = x1;
   y = y1;

/*  SetPixel*/
   setpix( x, y, couleur );                    /* Dessine le premier pixel */
   for (x=x1+1; x<=x2; ++x )                      /* Parcourt l'axe des X */
    {
     if ( d >= 0 )
      {
       y += yincr;
       d += aincr;
      }
     else
      d += bincr;
     setpix( x, y, couleur );
    }
  }
}

/*****************************************************************
*  GrafPrintf: Affiche une chaîne formatée sur l'écran graphique *
*----------------------------------------------------------------*
*  Entrées : X, Y  = Coordonnées de départ (0 - ...)             *
*            CC    = Couleur des caractères                      *
*            CF    = Couleur du fond  (255 = transparent)        *
*            STRING = Chaîne avec indications de formatage       *
*            ...    = Expressions comme pour printf              *
*****************************************************************/

void GrafPrintf( int x, int y, BYTE cc, BYTE cf, char * string, ... )
{
 va_list parameter;       /* Liste de paramètres pour les macros VA_...*/
 char affichage[255],                 /* Buffer pour la chaîne formatée */
      *cp;

 va_start( parameter, string );             /* Convertit les paramètres */
 vsprintf( affichage, string, parameter );                  /* Formate  */
 for ( cp = affichage; *cp; ++cp, x+= 8 )  /* Affiche la chaîne         */
    PrintChar( *cp, x, y, cc, cf );          /* formatée par PrintChar */
}

/*****************************************************************
*  ColorBox: Dessine un rectangle et le remplit avec un motif composé *
*            de lignes                                           *
*----------------------------------------------------------------*
*  Entrées : X1, Y1 = Coordonnées du coin sup gauche de la fenêtre *
*            X2, Y2 = Coordonnées du coin inf droit de la fenêtre *
*            COULMAX= code couleur maximal                       *
*  Info    : Les couleurs des lignes sont répétées dans un cycle de 0 *
*            à COULMAX                                           *
*****************************************************************/

void ColorBox( int x1, int y1, int x2, int y2, int coulmax )
{
 int x, y,                                  /* Variables d'itération */
     sx, sy;                  /* Point de départ de la dernière boucle */

 Line( x1, y1, x1, y2, 15 );                    /* Encadre le rectangle */
 Line( x1, y2, x2, y2, 15 );
 Line( x2, y2, x2, y1, 15 );
 Line( x2, y1, x1, y1, 15 );

 for ( y = y2-1; y > y1; --y ) /* du coin inf gauche au bord droit    */
```

```
 Line( x1+1, y2-1, x2-1, y, (BYTE) (y % coulmax) );

 for ( y = y2-1; y > y1; --y )  /* du coin inf droit au bord gauche   */
  Line( x2-1, y2-1, x1+1, y, (BYTE) (y % coulmax) );

  /*-- Du milieu du rectangle au bord supérieur ----------------------*/

 for ( x=x1+1, sx=x1+(x2-x1)/2, sy=y1+(y2-y1)/ 2; x < x2; ++x )
  Line( sx, sy, x, y1+1, (BYTE) (x % coulmax) );
}

/*****************************************************************
*  DrawAxis: Dessine des axes à gauche et en haut de l'écran     *
*----------------------------------------------------------------*
*  Entrées: XSTEP = Pas selon l'axe X                            *
*           YSTEP = Pas selon l'axe Y                            *
*           CC    = Couleur des caractères                       *
*           CF    = Couleur de fond  (255 = transparent)         *
*****************************************************************/

void DrawAxis( int stepx, int stepy, BYTE cc, BYTE cf )
{
 int x, y;                                  /* Variables d'itération */

 Line( 0, 0, MaxX, 0, cc );                       /* Trace l'axe X */
 Line( 0, 0, 0, MaxY, cc );                       /* Trace l'axe Y */

 for ( x = stepx; x < MaxX; x += stepx )          /* Gradue l'axe X */
  {
   Line( x, 0, x, 5, cc );
   GrafPrintf( x < 100 ? x - 8 : x - 12, 8, cc, cf, "%d", x );
  }

 for ( y = stepy; y < MaxY; y += stepy )          /* Gradue l'axe Y */
  {
   Line( 0, y, 5, y, cc );
   GrafPrintf( 8, y-4, cc, cf, "%3d", y );
  }
}

/*****************************************************************
*  Demo: Démontre l'usage des différentes fonctions de ce module *
*****************************************************************/

void Demo( void )
{
 #define PAUSE 100000          /* Définit la pause, dépend du système */

 int x;                                     /* Compteur d'itérations */
 BYTE page;                                   /* Compteur de pages */
 long delay;                                  /* Compteur de pauses */

 for ( page = 1; page <= Pages; ++page )
  {
   setpage( page-1 );                           /* Traite une page */
   showpage( page-1 );
   ColorBox( 50+page*2, 40, MaxX-50+page*2, MaxY-40, 16 );
   DrawAxis( 30, 20, 15, 255 );                 /* Trace les axes */
   GrafPrintf( 46, MaxY-10, 15, 255,
               "V16COLC  -  (c) by MICHAEL TISCHER" );
  }

    /*-- Affiche les pages graphiques en alternance ----------------*/

 for ( x = 0; x < 50; ++x )                     /* 50 passages   */
  {
   showpage( x % Pages );                       /* Affiche une page */
   for ( delay = 1; delay < PAUSE; ++delay )          /* Pause  */
    ;
  }
}

/*----------------------------------------------------------------*/
/*--                      PROGRAMME PRINCIPAL                   --*/
/*----------------------------------------------------------------*/

void main( void )
{
 union REGS regs;

 printf( "V16COLC.C  - (c) 1990, 92 by MICHAEL TISCHER\n\n" );
 if ( MODE == A640480 )                                 /* Mode VGA ? */
  {                                                          /* Oui  */
   if ( IsEgaVga() != VGA )                     /* A-t-on une carte VGA */
    {                                                        /* Non */
     printf( "Attention ! Pour travailler dans le mode graphique "\
      "de 640*480 pixels " \
      "\nil faut une carte VGA !\n" );
     exit(1);                                              /* Terminé */
    }
   else              /* Oui initialise le mode et fixe les paramètres */
    {
     MaxX = 639;
     MaxY = 479;
     Pages = 1;
     init640480();
    }
  }
 else                                          /* doit être un mode EGA */
  {
   if ( IsEgaVga() == NINI )                          /* Ni EGA ni VGA ? */
    {                                                          /* Non */
     printf( "Attention: Pour faire tourner ce programme il faut "\
       "au moins une \ncarte EGA !\n" );
     exit(1);                                              /* Terminé  */
    }
   else              /* Oui initialise le mode et fixe les paramètres */
    switch( MODE )
     {
      case A320200 : {                                     /* 320*200 pixels */
                      MaxX = 319;
                      MaxY = 199;
                      Pages = 8;
                      init320200();
                      break;
                     }
      case A640200 : {                                     /* 640*200 pixels */
                      MaxX = 639;
                      MaxY = 199;
                      Pages = 4;
                      init640200();
                      break;
                     }
      case A640350 : {                                     /* 640*350 pixels */
                      MaxX = 639;
                      MaxY = 349;
                      Pages = 2;
                      init640350();
                     }
     }
  }

 Demo();
 getch();                                       /* Attend une frappe de touche */
 regs.x.ax = 0x0003;                                 /* Revient au mode texte */
 int86( 0x10, &regs, &regs );
}
```

4.8.6. Les modes graphiques avec 256 couleurs

En tant que l'une des principales innovations apparue au cours de l'ère EGA, les protagonistes des cartes VGA peuvent introduire leurs modes graphiques étendus où 256 couleurs variées peuvent être affichées simultanément à partir d'une palette de plus de 262000 couleurs.

Mais le seul et unique mode standard 256 couleurs ne reconnaît qu'une résolution de 320*200 points seulement, ce qui représente un net recul par rapport aux résolutions

des modes 16 couleurs pouvant atteindre 640*480 points. La résolution n'est toutefois pas le seul critère dont tient compte l'utilisateur pour apprécier la qualité d'une image vidéo. La variété des couleurs est tout aussi importante surtout dans des cas précis sinon le mode graphique 256 couleurs de la carte VGA n'aurait jamais remporté un tel succès.

Dans cette section, nous allons étudier comment activer ce mode, adresser les différents points et doubler la résolution avec quelques astuces. Voici un aperçu des thèmes évoqués :

- ✓ Activation du mode 256 couleurs et adressage des points
- ✓ Quatre pages graphiques au lieu d'une
- ✓ 320*400 points et toujours deux pages graphiques

Activation du mode 256 couleurs et adressage des points

Parmi les divers modes graphiques de la carte VGA, le mode 256 couleurs avec une résolution de 320*200 points est sans conteste le moins complexe aux yeux du programmeur. Cela s'explique surtout par le mode d'adressage des points écran qui ressemble davantage au mode texte que les autres modes graphiques.

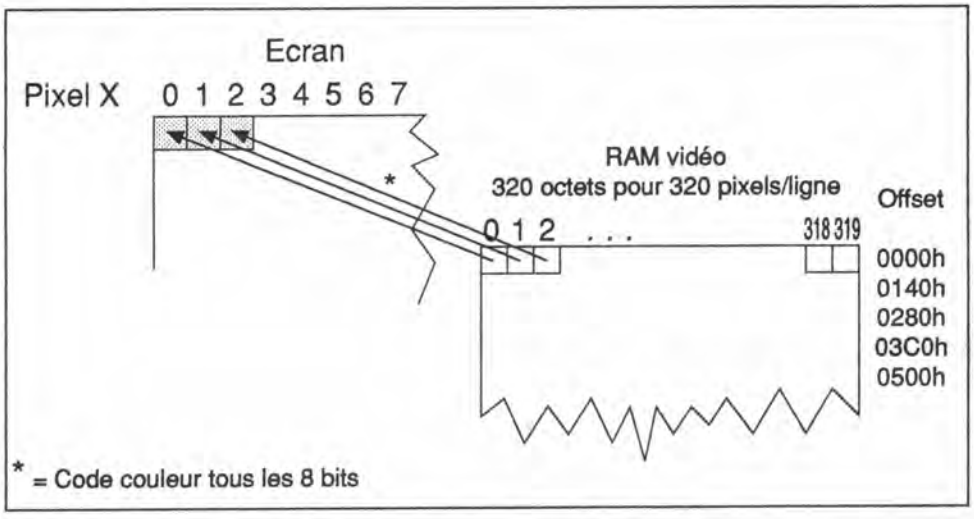
Avant d'en arriver à ce stade, il faut naturellement activer au préalable ce mode à l'aide de la fonction 00h de l'interruption vidéo du BIOS 10h. Pour spécifier ce mode, il faut transmettre la valeur 13h dans le registre AL ainsi que le numéro de fonction 00h dans le registre AH. C'est là tout ce qui est nécessaire pour initialiser ce mode. En procédant ainsi, vous initialisez non seulement ce mode graphique, mais aussi les modes Read et Write du contrôleur graphique ainsi que les registres appropriés pour que vous puissiez accéder aux 320*200 points sans être dérangé.

Vous pouvez oublier pour l'instant les quatre plans de bits qui viennent gêner en permanence lors du travail dans les modes graphiques 16 couleurs. En mode 256 couleurs, la RAM vidéo s'étend en effet au-delà de ces derniers selon une organisation très simple. Tout comme pour le mode texte, chaque point écran de la RAM vidéo est représenté exactement par un octet ce qui correspond à une couleur entre 0 et 255 pour ce point. La carte VGA considère ce code couleur comme un index direct de la table de couleurs DAC d'où est extraite la couleur finale du point concerné.

Sachant qu'un point représente un octet, une ligne écran se compose exactement de 320 octets dans la RAM vidéo. Ces derniers sont obligatoirement consécutifs et disposent les points d'une ligne depuis la gauche vers la droite. Les différentes lignes s'enchaînant automatiquement comme dans le mode texte, la formule suivante permet de calculer facilement l'adresse d'offset d'un point :

```
Offset = y * 320 + x
```

Du point de vue du programme, tous les points se trouvent à l'intérieur du segment 64 Ko. Dans ce mode, la RAM vidéo est configurée sur ce dernier à partir de l'adresse de segment A000h. Ainsi, les points peuvent être adressés facilement.



*Structure de la RAM vidéo en mode 256 couleurs avec une résolution de 320*200 points*

Nous ne vous proposons pas un programme d'exemple pour travailler dans ce mode car l'adressage des points est une tâche vraiment simplifiée grâce à la formule précédente. Il est donc inutile de démontrer cela en détail dans un listing. Mais ce n'est pas la seule et unique raison qui nous amène à ne pas examiner de plus près ce mode. Grâce à la structure d'organisation peu complexe de la RAM vidéo, les trois quarts de la RAM vidéo restent en effet inutilisés. Pour une page écran en 320*200 points, il nous faut 320*200 points soit 64000 octets en tout.

Théoriquement, on pourrait utiliser les 3*64 Ko restants pour afficher trois pages graphiques supplémentaires, mais la structure de la RAM vidéo telle qu'elle est définie interdit cette possibilité. Pourquoi IBM autorise cela, on ne peut que y spéculer. Si l'on considère la carte VGA du point de vue IBM, on est tenté immédiatement de la comparer avec la carte MCGA qui représente la petite soeur de la carte VGA pour IBM. Elle dispose du même mode 256 couleurs mais n'est équipée que de 64 Ko de RAM vidéo. Dans ce cas, on ne peut gérer qu'une seule page graphique. IBM souhaitait peut-être assurer la compatibilité entre ces deux cartes ce qui explique pourquoi la carte VGA ne reconnaît pas plusieurs pages graphiques.

Quelques astuces permettent cependant d'adresser les trois pages graphiques supplémentaires avec la carte VGA et les afficher sur l'écran comme le montre la section suivante.

Quatre pages graphiques au lieu d'une

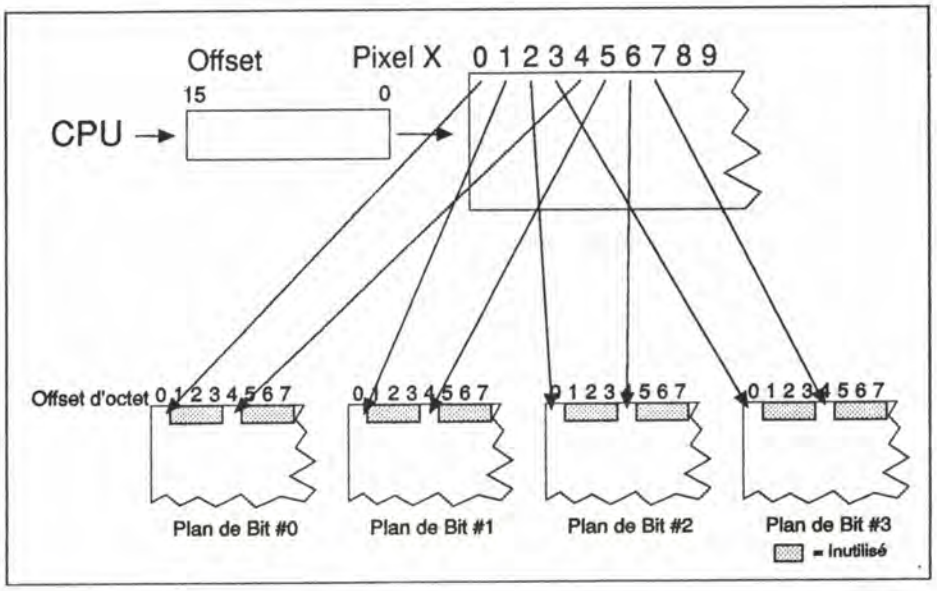
Il est nécessaire de comprendre l'organisation des informations de points dans les divers plans de bits pour trouver la clé permettant de gérer quatre pages graphiques en mode 256 couleurs. Tout comme dans le mode texte, la continuité de la RAM vidéo à partir de A000h censée aider le programmeur à adresser les points écran n'est qu'une illusion dans le mode 256 couleurs également.

En fait, les points graphiques sont gérés comme auparavant dans les quatre plans de bits où ils atterrissent à l'aide du mode Chain4. Il représente une extension du mode Odd/Even utilisé par la carte VGA en mode texte pour dévier les informations depuis la RAM vidéo vers les plans de bits 0 et 1.

Dans ce mode, les deux bits inférieurs de l'adresse d'offset déterminent, à chaque accès, le numéro du plan de bits atteint par l'octet spécifié. Sur le plan interne, ces deux bits sont ensuite mis sur zéro. Ils servent alors comme adresse d'offset pour l'accès au plan de bits sélectionné auparavant. Dans les divers plans de bits, trois octets restent ainsi inutilisés entre chaque octet occupé.

Par rapport aux modes graphiques 16 couleurs, l'information de couleur pour un point écran trouve suffisamment de place dans un octet à l'intérieur d'un plan de bits. Il est donc inutile de répartir sur quatre plans de bits comme à l'accoutumée. Quatre points occupent ainsi une adresse d'offset identique mais dans des plans de bits différents.

Considérons par exemple la première ligne écran. Les informations des quatre premiers points de cette ligne se trouvent réunies à l'adresse d'offset 0000h à l'intérieur de leurs plans de bits. Le plan de bits 0 est réservé au point de coordonnée X 0, le plan de bits 1 au point de coordonnée X 1, etc. D'après ce schéma, les points de coordonnées 4 à 7 se trouvent à nouveau à une adresse d'offset unique, notamment dans le quatrième octet du plan de bits concerné. Encore une fois, les points sont répartis sur les quatre plans de bits selon la méthode déjà décrite.



*Organisation interne des informations vidéo en mode 256 couleurs avec 320*200 points*

Les 64000 octets utilisés pour recevoir les 320*200 octets étant répartis sur quatre plans de bits différents, seuls les premiers 16 Ko d'un plan de bits sont effectivement occupés. Il ne reste pourtant plus de place pour d'autres pages écran parce que les octets sont répartis sur la totalité d'un plan de bits et même entre les interstices permanents constitués de trois octets.

Pour loger par conséquent plusieurs pages écran dans les divers plans de bits, il convient de rapprocher les octets des plans de bits pour que chaque octet ne soit pas suivi de trois autres. Pour obtenir ce résultat, il faut reprogrammer quelques registres VGA.

Mais cela oblige à renoncer au mode Chain4, c'est-à-dire que les informations de couleur ne peuvent plus être écrites manuellement dans les divers plans de bits. Le programme V3220 que nous proposons soutient les deux possibilités. La désignation codée du programme signifie "VGA 320*200 points". Le programme est écrit en Pascal et C, les deux versions disposent d'un module Assembleur spécifique. Les modules C s'appellent V3220C.C et V3220CA.ASM et les modules Pascal V3220P.PAS et V3220PA.ASM. Les modules Assembleur contiennent toutes sortes de routines pour initialiser le mode vidéo ainsi que pour accéder à des points individuels. Ils ont été spécialement développés dans un langage évolué eu égard à leur appel.

La routine init320200 du module Assembleur est utilisée dans les deux programmes pour installer le mode 320*200 points avec 256 couleurs de sorte que quatre pages graphiques différentes puissent être utilisées simultanément dans la RAM vidéo.

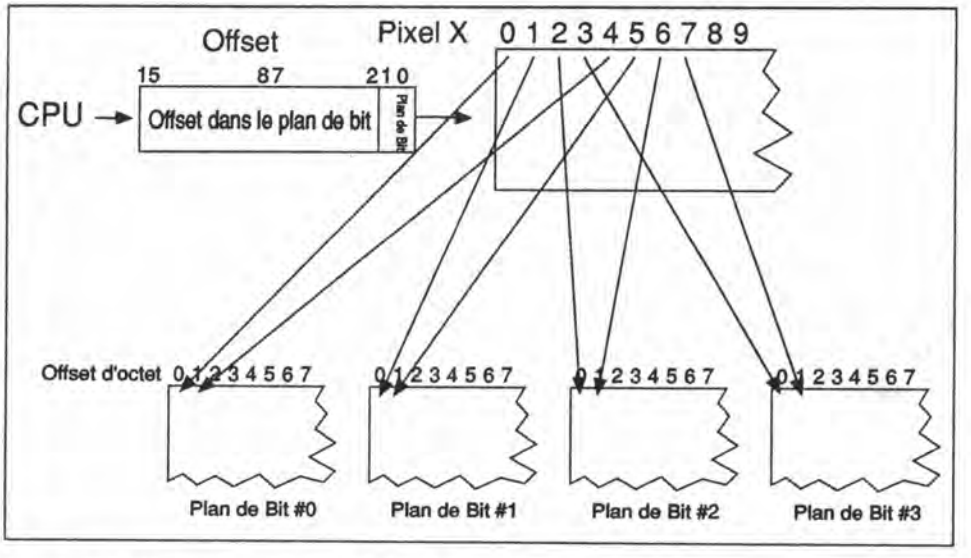
Pour commencer, le mode vidéo 13h est défini tout à fait normalement à travers le BIOS parce qu'il aurait été laborieux de programmer individuellement les divers registres.

Des modifications sont ensuite effectuées dans les registres pour que la structure de la RAM vidéo corresponde à nos souhaits.

On désactive à cet effet le mode Chain4 puis le mode Odd/Even. Cela provoque l'activation d'une sorte de mode linéaire pour que les adresses d'offset ne soient pas réparties et relogées dans les divers plans de bits lors d'un accès à la RAM vidéo. La seule conséquence de cette action se répercute sur les accès en lecture et écriture de la part de l'unité centrale dans la RAM vidéo. Du point de vue du contrôleur CRT qui est responsable de la structure de l'image vidéo cela ne change rien à la structure de la RAM vidéo.

A l'étape suivante, il faut informer le contrôleur CRT que les octets de couleur ne se suivent plus avec un intervalle de quatre octets à l'intérieur des plans de bits, mais se succèdent immédiatement les uns aux autres. Il convient donc de commuter du mode Doubleword au mode Byte et désactiver le mode Word. Pour obtenir la liste des registres impliqués par ces commutations, reportez-vous aux listings ou à la fin de cette section où les registres EGA et VGA sont décrits en détail.

Une fois l'opération achevée, la RAM vidéo ressemble à la figure suivante du point de vue du contrôleur CRT.



*Le mode 320*200 points modifié sous l'angle du contrôleur CRT*

Les 16 premiers Ko étant encore occupés dans chaque plan de bits, on peut placer trois pages écran supplémentaires dans la RAM vidéo dont un quart seulement sera stocké dans chacun des quatre plans de bits. Dans chaque plan de bits, la première page écran commence à l'adresse d'offset 0000h, la seconde à 4000h, la troisième à 8000h et la quatrième à C000h.

Mais l'adressage des points devient quelque peu complexe parce que tous les points ne peuvent plus être réclamés à travers les 64 Ko de la RAM vidéo en A000h. Il faut au contraire adresser individuellement chaque plan de bits. Une routine du module Assembleur se charge de réaliser cette tâche à votre place. Elle porte le nom SETPIX et attend les coordonnées X et Y du point réclamé ainsi que sa couleur en guise d'arguments.

SETPIX conçoit d'abord l'offset nécessaire pour adresser le point souhaité. En raison de la nouvelle organisation de la RAM vidéo, chaque ligne représentant seulement 80 points et non plus 320 à l'intérieur d'un plan de bits, elle multiplie la coordonnée Y par 80. Elle divise ensuite la coordonnée X par 4 parce que quatre points consécutifs occupent toujours la même et la seule adresse d'offset dans les divers plans de bits. La somme des deux calculs représente l'offset final pour l'accès au plan de bits concerné.

Le numéro du plan de bits à adresser est également fourni par la coordonnée X et notamment les deux bits inférieurs. Ils déterminent le nombre de bits par rapport auquel la valeur 1 doit être décalée vers la gauche permettant ainsi de créer les masques de bits pour le registre Map Mask. Une fois qu'ils ont été transmis dans le registre cité, il devient certain que seul le plan de bits souhaité sera effectivement adressé lors d'un prochain accès en écriture.

L'adresse d'offset préalablement calculée permet d'accéder à la RAM vidéo et d'affecter le point souhaité avec la couleur spécifiée. Vous vous demandez sans doute où se trouve la page écran dans tout cela.

Elle n'est pas configurée par un appel de SETPIX mais elle est déjà définie par un appel de SETPAGE. Par définition, SETPAGE configure la page écran adressée à chaque appel de SETPIX jusqu'à ce que SETPAGE définisse une autre page pour la sortie.

SETPIX adresse la page écran configurée en utilisant le contenu de la variable VIO_SEG comme adresse de segment lors de l'accès à la RAM vidéo. Compte tenu des appels de SETPAGE, elle reçoit notamment l'adresse de segment de la page concernée dans la RAM vidéo, soit A000h pour la première page, A400h pour la seconde, A800h pour la troisième et AC00h pour la quatrième. Comme le début de l'écran est déjà contenu dans l'adresse de segment, il n'est pas nécessaire d'en tenir compte dans SETPIX lors du calcul de l'adresse d'offset.

Pour spécifier non seulement des points mais aussi leurs couleurs, une autre routine appelée GETPIX est implantée dans les deux modules Assembleur. En guise d'arguments, elle attend une coordonnée X et Y et retourne la couleur du point spécifié comme résultat de la fonction.

Le fonctionnement de GETPIX ressemble considérablement à celui de SETPIX. Mais ici, il ne s'agit pas de programmer le registre Map Mask du séquenceur pour déterminer le plan de bits à adresser. C'est le registre Read Map du contrôleur graphique qui intervient ici. Lors d'un accès en lecture à la RAM vidéo en mode Read 0, il détermine le registre Latch et par conséquent le plan de bits dont le contenu est à transmettre à

l'unité centrale. Contrairement au registre Map Mask, il suffit de reporter la valeur du plan de bits à lire dans ce registre et non un masque de bits.

Une fois l'adresse d'offset du point souhaité configurée et le registre Read Map programmé en conséquence, la couleur du point peut être lue dans la RAM vidéo. A nouveau, le contenu des variables VIO_SEG sert d'adresse de segment pour tenir compte de la page écran définie lors du dernier appel de SETPAGE.

Pour pouvoir adresser et afficher les diverses pages, la routine Assembleur SHOWPAGE complète l'éventail des routines fonctionnant dans le mode 320*200 points. Elle attend le numéro de la page écran à afficher en tant qu'argument. Elle affiche la page sur l'écran en chargeant l'adresse d'offset 0000h, 4000h, 8000h ou C000h dans le registre Start du contrôleur CRT en fonction de la page écran.

Le premier objectif des deux programmes est de vous montrer le travail lié aux routines Assembleur. A cet effet, les quatre pages écran sont conçues à partir d'un modèle de points identique, notamment avec un repère de coordonnées, un message de copyright et un objet qui rappelle de loin le dos d'une enveloppe. Cet objet est construit à l'aide de plusieurs lignes affectées d'une couleur s'échelonnant entre 0 et n. Un tel objet est dessiné avec la procédure ou fonction COLORBOX.

Dans la page 0, n représentant la limite supérieure de la couleur est réglée sur 16. Elle est réglée sur 64 dans la page 1, sur 128 dans la page 2 et sur 256 dans la page 3. Pour dessiner les lignes, COLORBOX se sert d'une routine appelée LINE. Elle dessine une ligne sur la base de la routine SETPIX des modules Assembleur en respectant l'algorithme de Bresenham.

Comme les différentes pages écran sont identiques à un petit détail près, on obtient un effet optique intéressant si la commutation entre les pages s'effectue rapidement. Les deux programmes en langage évolué utilisent cette possibilité.

Listing : V3220C.C

```
/*****************************************************************
*                          V 3 2 2 0 C . C                       *
*----------------------------------------------------------------*
* Fonction          : Montre comment programmer le mode graphique *
*                     VGA 320*200 en 256 couleurs avec quatre     *
*                     pages d'écran . Le programme utilise les    *
*                     routines en assembleur du module V3220CA.ASM*
*----------------------------------------------------------------*
* Auteur            : MICHAEL TISCHER                            *
* Développé le      : 4.09.1990                                  *
* Dernière MAJ      : 14.02.1992                                 *
*----------------------------------------------------------------*
* Modèle mémoire    : SMALL                                      *
*----------------------------------------------------------------*
* (MICROSOFT C)                                                  *
* Compilation       : CL /AS v3220c.c v3220ca                    *
*----------------------------------------------------------------*
* (BORLAND TURBO C)                                              *
* Compilation       : Créer un fichier projet avec le contenu suivant *
*                       v3220c.c                                 *
*                       v3220ca.obj                              *
*----------------------------------------------------------------*
* Appel             : v3220c                                     *
*****************************************************************/

#include <dos.h>
#include <stdarg.h>
#include <stdlib.h>
#include <stdio.h>
#include <conio.h>

/*-- Déclarations de types ---------------------------------------*/

typedef unsigned char BYTE;

/*-- Références externes aux routines en assembleur -------------*/

extern void init320200( void );
extern void setpix( int x, int y, unsigned char couleur);
extern BYTE getpix( int x, int y );
extern void setpage( BYTE page );
extern void showpage( BYTE page );
extern void far * getfontptr( void );

/*-- Constantes -------------------------------------------------*/
```

```
#define MAXX 319                                /* Coordonnées maximales */
#define MAXY 199

/*****************************************************************************
*                                                                            *
*  IsVga: Teste la présence d'une carte VGA                                  *
*-----------------------------------------------------------------------------*
*  Entrée  : néant                                                           *
*  Sortie  : 0, si aucune carte VGA n'est branchée, sinon # 0                *
*                                                                            *
*****************************************************************************/

BYTE IsVga( void )
{
 union REGS  Regs;                      /* Registres pour gérer l'interruption */

 Regs.x.ax = 0x1a00;                    /* La fonction 1Ah n'existe qu'en VGA */
 int86( 0x10, &Regs, &Regs );
 return ( Regs.h.al == 0x1a );          /* Est-elle disponible ? */
}

/*****************************************************************************
*  PrintChar : Ecrit un caractère en dehors de la zone visible               *
*              de la mémoire d'écran                                         *
*-----------------------------------------------------------------------------*
*  Entrée  :  caractere = caractère à afficher                               *
*             x, y      = Coordonnées du coin supérieur gauche               *
*             cc        = Couleur du caractère                               *
*             cf        = Couleur du fond                                    *
*  Info    : Le caractère est dessiné dans une matrice de 8*8 pixels - il    *
*            sur la base du jeu de caractères 8*8 en ROM                     *
*****************************************************************************/

void PrintChar( char caractere, int x, int y, BYTE cc, BYTE cf )
{
 typedef BYTE   CARDEF[256][8];     /* Structure du jeu de caractères */
 typedef CARDEF far *CARPTR;        /* Pointe sur un jeu de caractères */

 BYTE     i, k,                             /* Compteur d'itérations */
          masque;           /* Masque binaire pour dessiner le caractère */

 static   CARPTR fptr = (CARPTR) 0;     /* Jeu de caractères en ROM */

 if( fptr == (CARPTR) 0 )       /* A-t-on déjà déterminé ce pointeur ? */
   fptr = getfontptr();         /*Non : détermine avec fonc. assembleur */

  /*- Dessine le caractère pixel par pixel ----------------------------*/

 if( cf == 255 )                            /* Caractère transparent ? */
   for( i = 0; i < 8; ++i )   /* Oui, ne dess. que pixels du 1er plan */
   {
     masque = (*fptr)[caractere][i];        /* Motif bin. pour ligne */
     for( k = 0; k < 8; ++k, masque <<= 1 ) /* Parcourt les colonnes */
     if( masque & 128 )                       /* Pixel à dessiner ? */
       setpix( x+k, y+i, cc );                              /* Oui */
   }
 else                                     /* Non dessine chaque pixel */
   for ( i = 0; i < 8; ++i )                  /* Parcourt les lignes */
   {
     masque = (*fptr)[caractere][i];        /* Motif bin. pour ligne */
     for( k = 0; k < 8; ++k, masque <<= 1 ) /* Parcourt les colonnes */
       setpix( x+k, y+i, (BYTE) (( masque & 128 ) ? cc : cf) );
   }
}

/*****************************************************************************
*  Line: Trace un segment dans la fenêtre graphique en appliquant            *
*        l'algorithme de Bresenham                                           *
*-----------------------------------------------------------------------------*
*  Entrées : X1, Y1 = Coordonnées de l'origine                               *
*            X2, Y2 = Coordonnées de l'extrémité terminale                   *
*            COULEUR = couleur du segment                                    *
*****************************************************************************/

/*-- Fonction accessoire pour échanger deux variables entières -------*/

void SwapInt( int *i1, int *i2 )
{
 int dummy;

 dummy = *i2;  *i2  = *i1;  *i1  = dummy;
}

/*-- Procédure principale ---------------------------------------------*/

void Line( int x1, int y1, int x2, int y2, BYTE couleur )
{
 int d, dx, dy,
     aincr, bincr,
     xincr, yincr,
     x, y;

 if( abs(x2-x1) < abs(y2-y1) )      /* Sens du parcours : axe X ou Y ? */
 {                                                          /* Par Y */
   if ( y1 > y2 )                                  /* y1 plus grand que y2? */
   {
     SwapInt( &x1, &x2 );                        /* Oui échange X1 et X2, */
     SwapInt( &y1, &y2 );                                   /* Y1 et Y2 */
   }

   xincr = ( x2 > x1 ) ? 1 : -1;                /* Fixe le pas horizontal */

   dy    = y2 - y1;
   dx    = abs( x2-x1 );
   d     = 2 * dx - dy;
   aincr = 2 * (dx - dy);
   bincr = 2 * dx;
   x     = x1;
   y     = y1;

   setpix( x, y, couleur );                   /* dessine le premier pixel */
   for( y=y1+1; y<= y2; ++y )                   /* Parcourt l'axe des Y */
   {
     if( d >= 0 )
     {
       x += xincr;
       d += aincr;
     }
     else
       d += bincr;
     setpix( x, y, couleur );
   }
 }
 else                                                       /* par X */
 {
   if( x1 > x2 )                                    /* x1 plus grand que x2? */
   {
     SwapInt( &x1, &x2 );                           /* Oui, échange X1 et X2 */
     SwapInt( &y1, &y2 );                                   /* Y1 et Y2 */
   }

   yincr = ( y2 > y1 ) ? 1 : -1;                /* Fixe le pas vertical */

   dx    = x2 - x1;
   dy    = abs( y2-y1 );
   d     = 2 * dy - dx;
   aincr = 2 * (dy - dx);
   bincr = 2 * dy;
   x     = x1;
   y     = y1;

   setpix( x, y, couleur );                   /* Dessine le premier pixel */
   for( x=x1+1; x<=x2; ++x )                    /* Parcourt l'axe des X */
   {
     if( d >= 0 )
     {
       y += yincr;
       d += aincr;
     }
     else
       d += bincr;
     setpix( x, y, couleur );
   }
 }
}

/*****************************************************************************
*  GrafPrintf: Affiche une chaine formatée sur l'écran graphique             *
*-----------------------------------------------------------------------------*
*  Entrées : X, Y   = Coordonnées de départ (0 - ...)                        *
*            CC     = Couleur des caractères                                 *
*            CF     = Couleur du fond  (255 = transparent)                   *
*            STRING = Chaine avec indications de formatage                   *
*            ...    = Expressions comme pour printf                          *
*****************************************************************************/

void GrafPrintf( int x, int y, BYTE cc, BYTE cf, char * string, ... )
{
 va_list  parameter;    /* Liste de paramètres pour les macros VA_... */
 char     affichage[255],           /* Buffer pour la chaine formatée */
          *cp;

 va_start( parameter, string );          /* Convertit les paramètres */
 vsprintf( affichage, string, parameter );                /* Formate */
 for( cp = affichage; *cp; ++cp, x+= 8 )       /* Affiche la chaine */
   PrintChar( *cp, x, y, cc, cf );          /* formatée par PrintChar */
}

/*****************************************************************************
*  ColorBox: Dessine un rectangle et le remplit avec un motif composé de     *
*            lignes                                                          *
*-----------------------------------------------------------------------------*
*  Entrées : X1, Y1 = Coordonnées du coin supérieur gauche de la fenêtre     *
*            X2, Y2 = Coordonnées du coin inférieur droit de la fenêtre      *
*            COULMAX= code couleur maximal                                   *
*  Info    : Les couleurs des lignes sont répétées dans un cycle de 0        *
*            à COULMAX                                                       *
```

```
/*************************************************************************/

void ColorBox( int x1, int y1, int x2, int y2, int coulmax )
{
 int x, y,                              /* Variables d'itération */
     sx, sy;                    /* Point de départ de la dernière boucle */

 Line( x1, y1, x1, y2, 15 );                    /* Encadre le rectangle */
 Line( x1, y2, x2, y2, 15 );
 Line( x2, y2, x2, y1, 15 );
 Line( x2, y1, x1, y1, 15 );

 for( y = y2-1; y > y1; --y )      /* du coin inf gauche au bord droit */
   Line( x1+1, y2-1, x2-1, y, (BYTE) (y % coulmax) );

 for( y = y2-1; y > y1; --y )      /* du coin inf droit au bord gauche */
   Line( x2-1, y2-1, x1+1, y, (BYTE) (y % coulmax) );

  /*-- Du milieu du rectangle au bord supérieur ----------------------*/

 for( x=x1+1, sx=x1+(x2-x1)/2, sy=y1+(y2-y1)/ 2; x < x2; ++x )
   Line( sx, sy, x, y1+1, (BYTE) (x % coulmax) );
}

/*************************************************************************
*  DrawAxis: Dessine des axes à gauche et en haut de l'écran            *
**-------------------------------------------------------------------**
*  Entrées  : XSTEP = Pas selon l'axe X                                 *
*             YSTEP = Pas selon l'axe Y                                 *
*             CC    = Couleur des caractères                            *
*             CF    = Couleur de fond  (255 = transparent)              *
*************************************************************************/

void DrawAxis( int stepx, int stepy, BYTE cc, BYTE cf )
{
 int  x, y;                             /* Variables d'itération */

 Line( 0, 0, MAXX, 0, cc );                     /* Trace l'axe X */
 Line( 0, 0, 0, MAXY, cc );                     /* Trace l'axe Y */

 for( x = stepx; x < MAXX; x += stepx )         /* Gradue l'axe X */
 {
   Line( x, 0, x, 5, cc );
   GrafPrintf( x < 100 ? x - 8 : x - 12, 8, cc, cf, "%d", x );
 }

 for( y = stepy; y < MAXY; y += stepy )         /* Gradue l'axe Y */
 {
   Line( 0, y, 5, y, cc );
   GrafPrintf( 8, y-4, cc, cf, "%3d", y );
 }
}
```

```
/*************************************************************************
*  Demo: Démontre l'usage des différentes fonctions de ce module        *
*************************************************************************/

void Demo( void )
{

#define PAUSE 100000            /* Définit la pause, dépend du système */

 int   x;                                   /* Compteur d'itérations */
 BYTE  page:                                  /* Compteur de pages */
 long  delay:                                 /* Compteur de pause */

 for( page = 0; page < 4; ++page )
 {
   setpage( page );                             /* Traite une page */
   ColorBox( 80, 25, 308, 175, ( page + 1 ) * 16 );  /* rect. couleur */
   DrawAxis( 30, 20, 15, 255 );                  /* Trace les axes */
   GrafPrintf( 46, MAXY-10, 15, 255,
               "V3220C  -  (c) by MICHAEL TISCHER" );
 }

     /*-- Affiche les pages graphiques en alternance ----------------*/

 for ( x = 0; x < 50; ++x )                         /* 50 passages */
 {
   showpage( (BYTE) (x % 4) );                  /* Affiche une page */
   for ( delay = 1; delay < PAUSE; ++delay );       /* Petite pause */
 }
}

/*-----------------------------------------------------------------*/
/*--                  PROGRAMME PRINCIPAL                        --*/
/*-----------------------------------------------------------------*/

void main( void )
{
 union REGS  regs;

 if( IsVga() )                              /* A-t-on une carte VGA ? */
 {                                                  /* Oui, c'est parti */
   init320200();                        /* Initialise le mode graphique */
   Demo();
   getch();                                   /* Attend une frappe */
   regs.x.ax = 0x0003;                      /* Revient au mode texte */
   int86( 0x10, &regs, &regs );
 }
 else
   printf( "V3220C  -  (c) 1990, 92 by MICHAEL TISCHER\n\nATTENTION "\
           "Ce programme exige une carte VGA ");

}
```

Listing : V3220CA.ASM

```
;*************************************************************************;
;*                          V 3 2 2 0 C A . A S M                        *;
;*-----------------------------------------------------------------------*;
;*   Fonction       : contient diverses routines pour travailler dans*;
;*                    le mode graphique 320*200 en 256 couleurs      *;
;*                    de la carte VGA                                *;
;*-----------------------------------------------------------------------*;
;*   Auteur         : MICHAEL TISCHER                                  *;
;*   Développé le   :  5.09.1990                                       *;
;*   Dernière MAJ   : 14.02.1992                                       *;
;*-----------------------------------------------------------------------*;
;*   Modèle mémoire : SMALL                                            *;
;*-----------------------------------------------------------------------*;
;*   Assemblage     : MASM /mx V3220CA;     ou     TASM -mx V3220CA    *;
;*                    ...puis lier à V3220C.C                          *;
;*************************************************************************;

IGROUP group _text                  ;Regroupe les segments de programme
DGROUP group _bss, _data            ;Regroupe les segments de données
       assume CS:IGROUP, DS:DGROUP, ES:DGROUP, SS:DGROUP

_BSS   segment word public 'BSS'        ;Segment des variables statiques
_BSS   ends                                          ;non initialisées

_DATA  segment word public 'DATA'                    ;Segment réservé aux
                                        ;variables globales et statiques
                                                     ;initialisées

_DATA  ends
```

```
;-- constantes ---------------------------------------------------------

SC_INDEX      = 3c4h      ;Registre d'index du contrôleur du séquenceur
SC_MAP_MASK   = 2                          ;Numéro du registre Map Mask
SC_MEM_MODE   = 4                  ;Numéro dy registre de mode mémoire

GC_INDEX      = 3ceh        ;Registre d'index du contrôleur graphique
GC_READ_MAP   = 4                        ;Numéro du registre Read Map
GC_GRAPH_MODE = 5                 ;Numéro du registre de mode graphique
GC_MISCELL    = 6                          ;Numéro du registre divers

CRTC_INDEX    = 3d4h         ;Registre d'index du contrôleur d'écran
CC_MAX_SCAN   = 9    ;Numéro du registre du maximum de lignes balayées
CC_START_HI   = 0Ch                      ;Numéro du registre HI_Start
CC_UNDERLINE  = 14h             ;Numéro du registre de soulignement
CC_MODE_CTRL  = 17h            ;Numéro du registre de contrôle de mode

DAC_WRITE_ADR = 3C8h                          ;Adresse DAC Write
DAC_READ_ADR  = 3C7h                           ;Adresse DAC Read
DAC_DATA      = 3C9h                    ;Registre de données DAC

VERT_RETRACE  = 3DAh                  ;registre d'état d'entrée #1

PIXX          = 320                        ;Résolution horizontale

;-- Données ------------------------------------------------------------

_DATA  segment word public 'DATA'
```

```
;
vio_seg    dw 0a000h            ;Segment mémoire écran avec page courante
;
_DATA  ends
;
;-- Programme ---------------------------------------------------------
;
_TEXT  segment byte public 'CODE'                    ;Segment de programme
;
;-- Déclarations publiques --------------------------------------------
;
public   _init320200                          ;Initialise le mode 320*200
public   _setpix                                  ;Dessine un pixel
public   _getpix                          ;Détermine la couleur d'un pixel
public   _showpage                                ;Affiche la page 0 ou 1
public   _setpage                    ;Fixe la page pour setpix ou getpix
public   _getfontptr                 ;Retourne un pointeur sur le jeu 8*8
;
;-- INIT320200: initialise le mode graphique 320*200us ----------------
;-- Déclaration : void init320200( void );
;
_init320200 proc near
;
        ;-- On commence par déclencher le mode 13h pour que le BIOS -----
        ;-- effectue la plus grande partie de l'initialisation ----------
        ;-- Puis on modifie les registres qui n'ont pas encore le -------
        ;-- contenu souhaité. -------------------------------------------
;
        mov   ax,0013h                              ;Appelle le mode 13h
        int   10h
;
        mov   dx,GC_INDEX                ;Désactive par le bit numéro 4
        mov   al,GC_GRAPH_MODE          ;la séparation des adresses mémoire
        out   dx,al                       ;dans le registre mode graphique
        inc   dx                                  ;du contrôleur graphique
        in    al,dx
        and   al,11101111b
        out   dx,al
        dec   dx
;
        mov   al,GC_MISCELL                 ;idem dans le registre divers
        out   dx,al                                 ;par le bit numéro 1
        inc   dx                                                        ;
        in    al,dx
        and   al,11111101b
        out   dx,al
;
        mov   dx,SC_INDEX              ;Modifie le registre de mode mémoire
        mov   al,SC_MEM_MODE                    ;du contrôleur de séquence
        out   dx,al                  ;de façon à mettre fin à la répartition
        inc   dx                    ;des adresses mémoire sur plusieurs plans
        in    al,dx                       ;de bits et à fixer le plan courant
        and   al,11110111b               ;par le registre de masquage binaire
        or    al,4                                                      ;
        out   dx,al
;
        mov   ax,vio_seg                 ;Remplit les quatre plans de bits
        mov   es,ax                     ;avec le code couleur 00h et efface
        xor   di,di                                          ;l'écran
        mov   ax,di
        mov   cx,8000h
        rep   stosw
;
        mov   dx,CRTC_INDEX                    ;Met fin au mode double -mot
        mov   al,CC_UNDERLINE            ; par le moyen du bit 6 du registre
        out   dx,al                   ; de soulignement du contrôleur d'écran
        inc   dx
        in    al,dx
        and   al,10111111b
        out   dx,al
        dec   dx
;
        mov   al,CC_MODE_CTRL          ;Passe du mode mot au mode octet par
        out   dx,al                ;le moyen du bit 6 du registre de contrôle
        inc   dx                                 ;de mode du contrôleur d'écran
        in    al,dx
        or    al,01000000b
        out   dx,al
;
        ret                                          ;retourne à l'appelant
;
_init320200 endp                                        ;Fin de la procédure
;
;-- SETPIX: Dessine un pixel dans une couleur donnée ------------------
;-- Déclaration : void setpix( int x, int y, unsigned char couleur );
;
_setpix    proc near
;
sframe     struc                                ;Structure d'accès à la pile
bp0        dw ?                                          ;Mémorise BP
ret_adr0   dw ?                          ;Adresse de retour à l'appelant
x0         dw ?                                                ;Abscisse X
y0         dw ?                                                ;Ordonnée Y
couleur    dw ?                                                  ;Couleur
sframe     ends                                        ;Fin de la structure
;
frame  equ [ bp - bp0 ]              ;adresse les éléments de la structure
;
        push  bp                          ;Prépare l'adressage des paramètres
        mov   bp,sp                                    ; par le registre BP
;
        push  di                                    ;Sauvegarde DI sur la pile
;
        mov   ax,PIXX / 4          ;Calcule l'offset dans la mémoire d'écran
        mul   frame.y0                                 ;et le charge en DI
        mov   cx,frame.x0
        mov   bx,cx
        shr   bx,1
        shr   bx,1
        add   ax,bx
        mov   di,ax
;
        and   cl,3                        ;Calcule en AH le masque binaire pour
        mov   ah,1                                      ; adresser le plan
        shl   ah,cl
        mov   al,SC_MAP_MASK                          ;Numéro du registre en AL
        mov   dx,SC_INDEX          ;Charge l'adresse d'index du séquenceur
        out   dx,ax                ;Charge le registre de masquage binaire
;
        mov   ax,vio_seg                ;ES pointe sur la mémoire d'écran
        mov   es,ax
        mov   al,byte ptr frame.couleur       ;Charge la couleur du pixel
        stosb                                  ; et la place dans le plan de bits
;
        pop   di                               ;Reprend le registre sur la pile
        pop   bp
;
        ret                                                ;Retourne à l'appelant
;
_setpix    endp                                            ;Fin de la procédure
;
;-- GETPIX: Détermine la couleur d'un pixel ---------------------------
;-- Déclaration : unsigned char getpix( int x, int y );
;
_getpix    proc near
;
sframe1    struc                                ;Structure d'accès à la pile
bp1        dw ?                                          ;Mémorise BP
ret_adr1   dw ?                          ;Adresse de retour à l'appelant
x1         dw ?                                                ;Abscisse X
y1         dw ?                                                ;Ordonnée Y
sframe1    ends                                        ;Fin de la structure
;
frame  equ [ bp - bp1 ]              ;adresse les éléments de la structure
;
        push  bp                          ;Prépare l'adressage des paramètres
        mov   bp,sp                                    ;par le registre BP
;
        push  si                                    ;Sauvegarde SI sur la pile
;
        mov   ax,PIXX / 4                ;calcule l'offset en mémoire d'écran
        mul   frame.y1                                 ; et le charge en SI
        mov   si,frame.x1
        mov   cx,si
        shr   si,1
        shr   si,1
        add   si,ax
;
        and   cl,3                       ;calcule en AH le masque binaire pour
        mov   ah,cl                                    ;adresser le plan
        mov   al,GC_READ_MAP                          ;Numéro du registre en AL
        mov   dx,GC_INDEX          ;Charge adresse index contrôleur graphique
        out   dx,ax                                ;Charge le registre Read Map
;
        mov   ax,vio_seg                ;ES pointe sur la mémoire d'écran
        mov   es,ax
        mov   al,es:[si]                          ;Charge la couleur du pixel
;
        pop   si                                ;reprend le registre sur la pile
        pop   bp
;
        ret                                                ;retourne à l'appelant
;
_getpix    endp                                            ;Fin de la procédure
;
;-- SETPAGE: Sélectionne la page concernée par les appels aux ----------
;--          fonctions setpix et getpix
;-- Déclaration : void setpage( unsigend char page );
;
_setpage   proc near
;
        pop   bx                                  ;Dépile l'adresse de retour
        pop   cx                                                ; et l'argument
;
        push  cx                                  ; puis les remet sur la pile
        push  bx
```

```
;
        mov   al,4                  ;Octet fort du segment = page * 4 + A0h
        mul   cl
        or    al,0A0h
;
        mov   byte ptr vio_seg + 1,al  ;Mémorise nouvelle adresse segment
;
        ret                                          ;retourne à l'appelant
;
_setpage   endp                                      ;Fin de la procédure
;
;-- SHOWPAGE: Affiche l'une des deux pages d'écran------
;-- Déclaration : void showpage( unsigend char page );
;
_showpage  proc near
;
        pop   bx                                  ;Dépile l'adresse de retour
        pop   cx                                        ; et l'argument
;
        push  cx                                  ;puis les remet sur la pile
        push  bx
;
        mov   al,64                         ;Octet fort de l'offset = page * 64
        mul   cl
        mov   ah,al                               ;Octet fort de l'offset en AH
;
        ;-- Charge la nouvelle adresse de début-------------------------
;
        mov   dx,CRTC_INDEX                     ;Adresse du contrôleur d'écran
        mov   al,CC_START_HI                      ;Numéro du registre en AL
        out   dx,ax                                     ;Effectue la sortie
;
        ;-- Attend un début de rafraîchissement d'écran ----------------
        mov   dx,VERT_RETRACE                   ;Attend la fin du balayage
sp3:    in    al,dx                                             ;vertical
        test  al,8
        jne   sp3
;
sp4:    in    al,dx                       ;puis le début du retour du faisceau
        test  al,8
        je    sp4
;
        ret                                                ;Retourne à l'appelant
;
_showpage  endp                                          ;Fin de la procédure
;
;-- GETFONTPTR: Renvoie un pointeur FAR sur le jeu de caractères 8*8
;-- Déclaration : void far * getfontptr( void )
;
_getfontptr proc near
;
        push  bp                                                ;Sauvegarde BP
;
        mov   ax,1130h                                  ;Charge les registres
        mov   bh,3
        int   10h                    ; puis déclenche l'interruption vidéo
;
        mov   dx,es                                  ;Transfère ES:BP en DX:AX
        mov   ax,bp
;
        pop   bp                                        ;reprend BP sur la pile
        ret                                              ;Retourne à l'appelant
;
_getfontptr endp                                         ;Fin de la procédure
;
;-- Fin ---------------------------------------------------------------
;
_text   ends                                          ;Fin du segment de programme
        end                                      ;Fin de la source en assembleur
```

Listing : V3220P.PAS

```
{*********************************************************************}
{*                        V 3 2 2 0 P . P A S                        *}
{**-----------------------------------------------------------------**}
{*  Fonction : Montre comment programmer le mode graphique VGA 320*200 *}
{*             en 256 couleurs avec quatre pages d'écran. Le programme *}
{*             utilise les routines en assembleur du module V3220PA.ASM *}
{**-----------------------------------------------------------------**}
{*  Auteur       : MICHAEL TISCHER                                     *}
{*  Développé le : 08.09.1990                                          *}
{*  Dernière MAJ : 14.01.1991                                          *}
{*********************************************************************}

program V3220P;

uses dos, crt;

{-- Déclarations de types ------------------------------------------}

type BPTR = ^byte;

{-- Références externes aux routines en assembleur ------------------}

{$L v3220pa} { Inclut les module en assembleur }

procedure init320200; external;
procedure setpix( x, y : integer; farbe : byte ); external;
function  getpix( x, y: integer ) : byte ; external;
procedure setpage( page : byte ); external;
procedure showpage( page : byte ); external;

{-- Constantes ------------------------------------------------------}

const MAXX = 319;                                  { Coordonnées maximales }
      MAXY = 199;

{*********************************************************************}
{*  IsVga : Teste la présence d'une carte VGA.                       *}
{**-----------------------------------------------------------------**}
{*  Entrée  : néant                                                  *}
{*  Sortie  : TRUE ou FALSE                                          *}
{*********************************************************************}

function IsVga : boolean;

var Regs : Registers;                      { Registres pour l'interruption }

begin
  Regs.AX := $1a00;                 { La fonction 1Ah n'existe qu'en VGA }
  Intr( $10, Regs );
  IsVga := ( Regs.AL = $1a );
end;

{*********************************************************************}
{*  PrintChar : Affiche un caractère en mode graphique               *}
{**-----------------------------------------------------------------**}
{*  Entrée :  caractere = le caractère à afficher                    *}
{*            x, y      = Coordonnées du coin sup gauche             *}
{*            cc        = Couleur du caractère                       *}
{*            cf        = Couleur du fond                            *}
{*  Info   :  Le caractère est dessiné dans une matrice de 8*8 pixels *}
{*            sur la base du jeu de caractères 8*8 en ROM            *}
{*********************************************************************}

procedure PrintChar( caractere : char; x, y : integer; cc, cf : byte );

type CARADEF = array[0..255,0..7] of byte; { Struct. jeu de caractères}
     CARAPTR = ^CARADEF;                   { Pointe sur le jeu de caractères }

var  Regs  : Registers;          { Registres pour gérer les interruptions }
     ch    : char;                                    { Pixel du caractère }
     i, k,                                          { Compteur d'itérations }
     Masque : byte;              { Masque binaire pour dessiner le caractère}

const fptr : CARAPTR = NIL;   { Pointe sur le jeu de caractères en ROM }

begin
  if fptr = NIL then                  { A-t-on déjà déterminé ce pointeur ? }
    begin                                                          { Non}
      Regs.AH := $11;                          { Appelle l'option $1130 de la }
      Regs.AL := $30;                                { fonction vidéo du BIOS }
      Regs.BH := 3;                  { pour obtenir un pointeur sur le jeu 8*8 }
      intr( $10, Regs );
      fptr := ptr( Regs.ES, Regs.BP );                { Compose le pointeur }
    end;

  if ( cf = 255 ) then                              { Caractère transparent ? }
    for i := 0 to 7 do  { Oui ne dessine que les pixels du premier plan)
      begin
        Masque := fptr^[ord(caractere),i];{Motif binaire pour une ligne}
        for k := 0 to 7 do
          begin
            if ( Masque and 128 <> 0 ) then          { Pixel à dessiner? }
              setpix( x+k, y+i, cc );                                { Oui }
            Masque := Masque shl 1;
```

```
        end;
    end
  else                                        ( Non, tient compte du fond )
    for i := 0 to 7 do                          ( Parcourt les lignes )
      begin
        Masque := fptr^[ord(caractere),i];(Lit motif binaire pour ligne)
        for k := 0 to 7 do
          begin
            if ( Masque and 128 <> 0 ) then       ( Premier plan ? )
              setpix( x+k, y+i, cc )                ( Oui )
            else
              setpix( x+k, y+i, cf );             ( Non, fond)
            Masque := Masque shl 1;
          end;
      end;
end;

{(*****************************************************************
*  Line : Trace un segment dans la fenêtre graphique en appliquant  *
*         l'algorithme de Bresenham                                 *
******************************************************************
*  Entrée : X1, Y1  = Coordonnées de l'origine (0 - ...)            *
*           X2, Y2  = Coordonnées de l'extrémité terminale          *
*           COULEUR = couleur du segment                            *
*****************************************************************)}

procedure Line( x1, y1, x2, y2 : integer; couleur : byte );

var d, dx, dy,
    aincr, bincr,
    xincr, yincr,
    x, y                  : integer;

{-- Procédure accessoire pour échanger deux variables entières --------}

procedure SwapInt( var i1, i2: integer );

var dummy : integer;

begin
  dummy := i2;
  i2    := i1;
  i1    := dummy;
end;

{-- Procédure principale ----------------------------------------------}

begin
  if ( abs(x2-x1) < abs(y2-y1) ) then     (Parcours : par axe X ou Y ? )
    begin                                             ( par l'axe des Y )
      if ( y1 > y2 ) then                           ( y1 supérieur à y2? )
        begin
      SwapInt( x1, x2 );                        ( Oui, échange X1 et X2, )
      SwapInt( y1, y2 );                                ( Y1 et Y2 )
        end;

      if ( x2 > x1 ) then xincr := 1          ( Fixe le pas horizontal )
                     else xincr := -1;

      dy := y2 - y1;
      dx := abs( x2-x1 );
      d  := 2 * dx - dy;
      aincr := 2 * (dx - dy);
      bincr := 2 * dx;
      x := x1;
      y := y1;

      Setpix( x, y, couleur );                ( Dessine le premier point )
      for y:=y1+1 to y2 do                      ( Parcourt l'axe des Y )
        begin
          if ( d >= 0 ) then
            begin
              inc( x, xincr );
              inc( d, aincr );
            end
          else
            inc( d, bincr );
          Setpix( x, y, couleur );
        end;
    end
  else                                                ( par l'axe des X )
    begin
      if ( x1 > x2 ) then                           ( x1 plus grand que x2? )
        begin
      SwapInt( x1, x2 );                        ( Oui, échange X1 et X2 )
      SwapInt( y1, y2 );                                ( Y1 et Y2 )
        end;

      if ( y2 > y1 ) then yincr := 1          ( Fixe le pas vertical )
                     else yincr := -1;

      dx := x2 - x1;
      dy := abs( y2-y1 );
      d  := 2 * dy - dx;
      aincr := 2 * (dy - dx);
      bincr := 2 * dy;
      x := x1;
      y := y1;

      Setpix( x, y, couleur );                ( Dessine le premier point )
      for x:=x1+1 to x2 do                      ( Parcourt l'axe des X )
        begin
          if ( d >= 0 ) then
            begin
              inc( y, yincr );
              inc( d, aincr );
            end
          else
            inc( d, bincr );
          Setpix( x, y, couleur );
        end;
    end;
end;

{(*****************************************************************
*  GrafPrint: Affiche une chaine formatée sur l'écran graphique     *
******************************************************************
*  Entrées : X, Y   = Coordonnées de départ (0-...)                 *
*            CC     = Couleur des caractères                        *
*            CF     = Couleur du fond (255 = transparent)           *
*            STRING = Chaine avec indications de formatage          *
*****************************************************************)}

procedure GrafPrint( x, y : integer; cc, cf : byte; strt : string );

var i : integer;                                  ( Compteur d'itérations )

begin
  for i:=1 to length( strt ) do
    begin
      printchar( strt[i], x, y, cc, cf );       ( Affiche par printchar )
      inc( x, 8 );                          ( x à la position du caractère suivant )
    end;
end;

{(*****************************************************************
*  ColorBox : Dessine un rectangle et le remplit avec un motif composé *
*             de lignes.                                            *
******************************************************************
*  Entrées : X1, Y1  = Coordonnées du coin sup gauche de la fenêtre *
*            X2, Y2  = Coordonnées du coin inf droit de la fenêtre  *
*            COULMAX = Code de couleur maximal                      *
*  Info    : Les couleurs des lignes sont répétées dans un cycle de 0 *
*            à COULMAX                                              *
*****************************************************************)}

procedure ColorBox( x1, y1, x2, y2 : integer; coulmax : byte );

var x, y,                                         ( Compteur d'itérations )
    sx, sy : integer;           ( Point de départ de la dernière boucle )

begin
  Line( x1, y1, x1, y2, 15 );                   ( Cadre autour du rectangle )
  Line( x1, y2, x2, y2, 15 );
  Line( x2, y2, x2, y1, 15 );
  Line( x2, y1, x1, y1, 15 );

  for y := y2-1 downto y1+1 do       ( Du coin inf gauche au bord droit )
    Line( x1+1, y2-1, x2-1, y, y mod coulmax );

  for y := y2-1 downto y1+1 do       ( Du coin inf droit au bord gauche )
    Line( x2-1, y2-1, x1+1, y, y mod coulmax );

  {-- Du milieu du rectangle au bord supérieur ------------------------}

  sx := x1+ (x2-x1) div 2;
  sy := y1+ (y2-y1) div 2;
  for x := x1+1 to x2-1 do
    Line( sx, sy, x, y1+1, x mod coulmax );
end;

{(*****************************************************************
*  DrawAxis : Dessine des axes en haut et à gauche de l'écran       *
******************************************************************
*  Entrées : XSTEP = Pas selon l'axe X                              *
*            YSTEP = Pas selon l'axe Y                              *
*            CC    = Couleur des caractères                         *
*            CF    = Couleur de fond (255 = transparent)            *
*****************************************************************)}

procedure DrawAxis( stepx, stepy : integer; cc, cf : byte );

var x, y     : integer;                           ( Variables d'itération )
    grad : string[3];

begin
```

```
  Line( 0, 0, MAXX, 0, cc );                    ( Trace l'axe X )
  Line( 0, 0, 0, MAXY, cc );                    ( Trace l'axe Y )

  x := stepx;                                   ( Gradue l'axe X )
  while ( x < MAXX ) do
    begin
      Line( x, 0, x, 5, cc );
      str( x, grad );
      if ( x < 100 ) then
        GrafPrint( x - 8 , 8, cc, cf, grad )
      else
        GrafPrint( x - 12, 8, cc, cf, grad );
      inc( x, stepx );
    end;

  y := stepy;                                   ( Gradue l'axe Y )
  while ( y < MAXY ) do
    begin
      Line( 0, y, 5, y, cc );
      str( y:3, grad );
      GrafPrint( 8, y-4, cc, cf, grad );
      inc( y, stepy );
    end;
end;


{*********************************************************************
*  Demo : Démontre l'usage des différentes fonctions de ce module     *
**--------------------------------------------------------------------**
*  Entrée: néant                                                      *
*********************************************************************}

procedure Demo;

const PAUSE = 100;                  { Compteur de pause en millisecondes}

var page : byte;                                { Compteur de pages }

begin
 for page := 0 to 4 do
   begin
     setpage( page );                                  { Traite une page }
      ColorBox(80, 25, 308, 175, (page + 1)*16);{ Dessine un rectangle }
     DrawAxis( 30, 20, 15, 255 );                      { Trace les axes }
     GrafPrint( 46, MAXY-10, 15, 255,
                'V3220P  -  (c) by MICHAEL TISCHER' );
   end;

  {-- Affiche alternativement les quatre pages graphiques ------------}

 for page := 0 to 50 do                                  { 50 passages }
   begin
     showpage( page mod 4 );                         { Affiche une page }
     delay( PAUSE );                                   { Petite pause }
   end;
end;


{-------------------------------------------------------------------}
{--                      PROGRAMME PRINCIPAL                      ----}
{-------------------------------------------------------------------}

begin
 if IsVga then                                  { A-t-on une carte VGA ? }
   begin                                                { Oui, c'est parti }
     init320200;                              { Initialise le mode graphique }
     Demo;
     repeat until keypressed;
     Textmode( C080 );                                { Restaure le mode texte }
   end
 else
   writeln( 'V3220P  -  (c) 1990 by MICHAEL TISCHER'#13#10#10 +
            'Attention! Ce programme exige une carte VGA !');
end.
```

Listing : V3220PA.ASM

```
;*****************************************************************;
;*                      V 3 2 2 0 P A . A S M                    *;
;*---------------------------------------------------------------*;
;*   Fonction : contient diverses routines pour travailler dans le *;
;*              mode graphique 320*200 256 couleurs de la carte VGA *;
;*---------------------------------------------------------------*;
;*   Auteur       : MICHAEL TISCHER                              *;
;*   Développé le :  5.09.1990                                   *;
;*   Dernière MAJ : 14.01.1991                                   *;
;*---------------------------------------------------------------*;
;*   Assemblage : MASM /mx V3220PA;     ou TASM -mx V3220PA     *;
;*                ... puis inclure dans V3220P.PAS              *;
;*****************************************************************;

;-- Constantes ---------------------------------------------------

SC_INDEX        = 3c4h    ;Registre d'index du contrôleur du séquenceur
SC_MAP_MASK     = 2                     ;Numéro du registre Map-Mask
SC_MEM_MODE     = 4                   ;Numéro du registre de mode mémoire

GC_INDEX        = 3ceh          ;registre d'index du contrôleur graphique
GC_READ_MAP     = 4                     ;Numéro du registre Read Map
GC_GRAPH_MODE   = 5                 ;Numéro du registre de mode graphique
GC_MISCELL      = 6                       ;Numéro du registre divers
GC_BIT_MASK     = 8                 ;Numéro du registre de masquage binaire

CRTC_INDEX      = 3d4h            ;registre d'index du contrôleur d'écran
CC_MAX_SCAN     = 9     ;Numéro du registre du maximum de lignes balayées
CC_START_HI     = 0Ch                   ;Numéro du registre Hi_Start
CC_UNDERLINE    = 14h               ;Numéro du registre de soulignement
CC_MODE_CTRL    = 17h             ;Nummer du registre de contrôle de mode

DAC_WRITE_ADR   = 3C8h                          ;Adresse DAC Write
DAC_READ_ADR    = 3C7h                          ;Adresse DAC Read
DAC_DATA        = 3C9h                  ;Registres de données DAC

VERT_RETRACE    = 30Ah                  ;Registre d'état d'entrée #1

PIXX            = 320                       ;Résolution horizontale

;-- Segment de données -------------------------------------------

DATA    segment word public

vio_seg     dw (?)                  ;Segment mémoire d'écran avec la page
                                    ;courante, à initialiser à l'exécution

DATA    ends

;-- Programme ----------------------------------------------------

CODE    segment byte public                     ;Segment de programme

        assume cs:code, ds:data

;-- Déclarations publiques ---------------------------------------

public    init320200                    ;Initialise le mode 320*200
public    setpix                                  ;Dessine un pixel
public    getpix                  ;Détermine la couleur d'un pixel
public    showpage                        ;Affiche la page 0 ou 1
public    setpage               ;Fixe la page pour setpix et getpix

;-----------------------------------------------------------------
;-- INIT320200: initialise le mode graphique 320*200
;-- Appel depuis TP: init320200;

init320200 proc near

              ;-- On commence par demander le mode 13h pour que le BIOS
              ;-- effectue la plus grande partie de l'initialisation .
                ;-- Puis on modifie les registres qui n'ont pas encore
                                          ;-- le contenu souhaité
        mov   ax,0013h                  ;Appelle le mode ordinaire 13h
        int   10h

        mov   dx,GC_INDEX         ;Désactive la séparation des adresses
        mov   al,GC_GRAPH_MODE  ;mémoire dans le registre mode graphique
        out   dx,al                             ;du contrôleur graphique
        inc   dx                          ;en manipulant le bit numéro 4
        in    al,dx
        and   al,11101111b
        out   dx,al
        dec   dx

        mov   al,GC_MISCELL               ;idem dans le registre divers
        out   dx,al                       ;en manipulant le bit numéro 1
```

```
        inc   dx                                              ;
        in    al,dx
        and   al,11111101b
        out   dx,al

        mov   dx,SC_INDEX           ;Modifie le registre de mode mémoire
        mov   al,SC_MEM_MODE          ;dans le contrôleur du séquenceur
        out   dx,al                 ;de façon à mettre fin à la répartition
        inc   dx                  ;des adresses sur plusieurs plans de bits
        in    al,dx               ;et à fixer le plan courant par le registre
        and   al,11110111b                    ;de masquage binaire
        or    al,4                                            ;
        out   dx,al

        mov   ax,0A000h               ;Remplit les quatre plans de bits
        mov   vio_seg,ax            ;avec le code couleur 00h et efface
        mov   es,ax                                   ;ainsi l'écran
        xor   di,di
        mov   ax,di
        mov   cx,8000h
        rep   stosw

        mov   dx,CRTC_INDEX       ;Arrête le mode double-mot par le bit 6
        mov   al,CC_UNDERLINE       ;du registre de soulignement du
        out   dx,al                             ;contrôleur d'écran
        inc   dx
        in    al,dx
        and   al,10111111b
        out   dx,al
        dec   dx

        mov   al,CC_MODE_CTRL     ;Par le bit 6 du registre de contrôle
        out   dx,al                   ;de mode du contrôleur d'écran
        inc   dx                    ;passe du mode mot au mode octet
        in    al,dx
        or    al,01000000b
        out   dx,al

        ret                                           ;retourne à l'appelant

Init320200 endp                                       ;Fin de la procédure


;---------------------------------------------------------------------------
;-- SETPIX: Dessine un pixel dans une couleur donnée
;-- Appel depuis TP: setpix( x , y : integer; couleur : byte );

setpix    proc near

sframe    struc                             ;Structure d'accès à la pile
bp0       dw ?                                          ;Mémorise BP
ret_adr0  dw ?                          ;Adresse de retour à l'appelant
couleur   dw ?                                          ;Couleur
y0        dw ?                                          ;Ordonnée Y
x0        dw ?                                          ;Abscisse X
sframe    ends                                  ;Fin de la structure

frame  equ [ bp - bp0 ]           ;Adresse les éléments de la structure

        push  bp                    ;Prépare l'adressage des paramètres
        mov   bp,sp                                 ; par le registre BP

        mov   ax,PIXX / 4           ;Calcule l'offset en mémoire d'écran
        mul   frame.y0                              ; et en charge DI
        mov   cx,frame.x0
        mov   bx,cx
        shr   bx,1
        shr   bx,1
        add   ax,bx
        mov   di,ax

        and   cl,3                        ;Calcule en AH le masque binaire
        mov   ah,1                              ; pour adresser le plan
        shl   ah,cl
        mov   al,SC_MAP_MASK                    ;Numéro du registre en AL
        mov   dx,SC_INDEX         ;Charge l'adresse d'index du séquenceur
        out   dx,ax               ;Charge le registre de masquage binaire

        mov   ax,vio_seg                ;ES pointe sur la mémoire d'écran
        mov   es,ax
        mov   al,byte ptr frame.couleur       ;Charge la couleur du pixel
        stosb                           ;et la place dans le plan de bits

        pop   bp                          ;Reprend le registre sur la pile

        ret   6                                       ;Retourne à l'appelant
                                  ;en retirant les paramètres de la pile

setpix endp                                           ;Fin de la procédure

;---------------------------------------------------------------------------
;-- GETPIX: Détermine la couleur d'un pixel
;-- Appel depuis TP: x := getpix( x , y : integer );

getpix proc near

sframe1   struc                             ;Structure d'accès à la pile
bp1       dw ?                                          ;Mémorise BP
ret_adr1  dw ?                          ;Adresse de retour à l'appelant
y1        dw ?                                          ;Ordonnée Y
x1        dw ?                                          ;Abscisse X
sframe1   ends                                  ;Fin de la structure

frame  equ [ bp - bp1 ]           ;Adresse les éléments de la structure

        push  bp                    ;Prépare l'adressage des paramètres
        mov   bp,sp                                 ; par le registre BP

        mov   ax,PIXX / 4           ;Calcule l'offset en mémoire d'écran
        mul   frame.y1                              ; et le transfère en SI
        mov   si,frame.x1
        mov   cx,si
        shr   si,1
        shr   si,1
        add   si,ax

        and   cl,3                        ;Calcule en AH le masque binaire pour
        mov   ah,cl                                 ; adresser le plan
        mov   al,GC_READ_MAP                    ;Numéro du registre en AL
        mov   dx,GC_INDEX ;Charge adresse d'index du contrôleur graphique
        out   dx,ax                             ;Charge le registre Read-Map

        mov   ax,vio_seg                ;ES pointe sur la mémoire d'écran
        mov   es,ax
        mov   al,es:[si]                        ;Charge la couleur du pixel

        pop   bp                          ;Reprend le registre sur la pile

        ret   4                           ;Retourne à l'appelant en retirant
                                                  ;les arguments de la pile

getpix endp                                           ;Fin de la procédure

;---------------------------------------------------------------------------
;-- SETPAGE: Sélectionne la page concernée par les appels
;--  aux fonctions setpix et getpix
;-- Appel depuis TP: setpage( page : byte );

setpage   proc near

        pop   bx                                ;Dépile l'adresse de retour
        pop   cx                                        ; et l'argument

        push  cx                                ; puis les remet sur la pile
        push  bx

        mov   al,4                        ;Octet fort du segmt = page * 4 + A0h
        mul   cl
        or    al,0A0h

        mov   byte ptr vio_seg + 1,al;Mémorise la nouvelle adr de segment

        ret   2                                       ;Retourne à l'appelant
                                          ;en enlevant l'argument de la pile

setpage   endp                                        ;Fin de la procédure

;---------------------------------------------------------------------------
;-- SHOWPAGE: Affiche l'une des deux pages d'écran
;-- Appel depuis TP: showpage( page : byte );

showpage  proc near

        pop   bx                                ;Dépile l'adresse de retour
        pop   cx                                        ; et l'argument

        push  cx                                ; puis les remet sur la pile
        push  bx

        mov   al,64                       ;Octet fort de l'offset= page * 64
        mul   cl
        mov   ah,al                             ;Octet fort de l'offset en AH

        ;-- Charge la nouvelle adresse de début ----------------------

        mov   dx,CRTC_INDEX               ;Adresse du contrôleur d'écran
        mov   al,CC_START_HI            ;Charge en AL le numéro du registre
        out   dx,ax                                   ; effectue la sortie

                ;-- Attend un début de rafraîchissement d'écran-----

        mov   dx,VERT_RETRACE                   ;Attend la fin du balayage
sp3:    in    al,dx                                           ;vertical
        test  al,8
        jne   sp3
```

```
sp4:    in    al,dx           ;puis le début du retour du faisceau
        test  al,8
        je    sp4

        ret   2               ;Retourne à l'appelant en retirant
; l'argument de la pile
```

```
showpage    endp              ;Fin de la procédure

;--- Fin ---------------------------------------------------------

CODE    ends                  ;Fin du segment de code
        end                   ;Fin du programme
```

320*400 points graphiques et toujours deux pages graphiques

Bien que les 256 couleurs soient si bien imprégnées dans le mode 320*200 points, ce mode fait une piètre figure par rapport aux autres modes vidéo de la carte VGA. Dans le mode VGA standard à haute résolution, le mode 12h, le nombre de points affichés sur l'écran est cinq fois plus important. On peut toutefois considérer le mode 320*200 points comme parfait. Grâce à la multiplicité des couleurs, il procure une impression de haute résolution comparable à celle du téléviseur, mais on souhaite malgré tout atteindre une résolution encore plus élevée.

Cela est d'autant plus valable au vu du dernier paragraphe qui montre qu'une page écran nécessite 16 Ko dans ce mode et qu'il reste encore suffisamment de place pour recevoir des pages plus volumineuses. Ajoutons à cela que la carte VGA affiche sans problème 400 lignes de points sur l'écran et non 200 parce qu'un mode 200 points est absolument inexistant sur cette carte.

A vrai dire, la carte VGA configure 400 lignes sur l'écran en mode 320*200 points où 200 lignes seulement sont créées puis dédoublées. Que se passerait-il si on désactivait la duplication des lignes de points ? La carte VGA affiche tout de même 400 lignes de points différentes sur l'écran.

Les registres CRT responsables du Timing horizontal et vertical doivent être reprogrammés pour créer un mode 320*400 points à partir d'un mode 320*200 points. On obtient ainsi un comportement de pages inhabituel entre les axes horizontaux et verticaux mais le doublement de la résolution est tellement appréciable que les arguments en sa faveur relèvent le plateau de la balance.

Selon la méthode traditionnelle, les 128000 points ne peuvent plus être adressés dans ce mode parce qu'il faut une RAM vidéo avec une longueur de 128 Ko et le mode Chain4 ne convient plus. Comme on est de toute manière obligé de renoncer à la seconde page écran, il est peut-être plus raisonnable de définir le mode d'adressage qui nous a déjà permis de créer quatre pages écran dans la dernière section. En mode 320*400 points chaque page écran nécessitant 128 Ko et non 64, deux autres pages écran restent encore disponibles dans ce mode. Cela suffit amplement à satisfaire les besoins de la plupart des applications.

Nous allons illustrer la programmation graphique dans ce mode à l'aide de deux exemples portant les noms V3240C.C et V3240P.PAS. Ils résultent directement des deux programmes présentés dans la section précédente. Deux modules Assembleur sont à nouveau inclus ici. Ils s'appellent V3240CA.ASM et V3240PA.ASM et ont été

légèrement modifiés par rapport à leurs homologues de la section précédente. Ces modifications montrent les différences entre les modes 320*200 et 320*400.

Une première différence saute aux yeux dans les deux listings Assembleur : la routine init320200 porte maintenant le nom init320400. A part cela, cette routine a subi très peu de modifications. Comme précédemment, le mode vidéo 13h est d'abord initialisé à travers le BIOS, puis l'adressage linéaire de la RAM vidéo est activé et le mode Doubleword ignoré au profit du mode Byte.

L'accès au registre Maximum Scan Line du contrôleur CRT est toutefois une nouveauté. Deux informations y sont modifiées, d'une part le bit responsable du dédoublement des lignes de points et d'autre part, le hauteur des différents "caractères". Elle doit passer de 1 à 0 puisqu'il s'agit d'afficher seulement une ligne de points par parcours de lignes et non deux. Il faut en outre désactiver le bit 200 lignes pour que les lignes de points ne soient plus dédoublées.

Ces deux manipulations suffisent pour commuter du mode 320*200 vers le mode 320*400 points. Rien de plus simple pour doubler la résolution de l'écran !

La modification de la résolution écran se répercute sur les routines Assembleur SETPAGE et SHOWPAGE. Elles doivent en effet être ajustées par rapport à la taille modifiée d'une page écran. En fait, la seconde page écran - et la dernière - ne commence plus en 4000h mais en 8000h.

Il n'est pas nécessaire de modifier les routines SETPIX et GETPIX puisque la largeur des lignes et leur disposition dans la RAM vidéo n'ont pas changé. Seul leur nombre a doublé mais cela est déjà pris en compte par la transmission de coordonnées Y appropriées.

Les deux programmes en langage évolué démontrent que la commutation fonctionne réellement dans ces modes. Dans les deux pages écran, ils dessinent un repère ainsi qu'une boîte remplie avec des lignes de couleur et écrivent en outre un message de copyright dans la RAM vidéo.

La croix formée par le repère indique que vous êtes en présence de 400 lignes de points et il n'est pas étonnant que l'axe vertical s'étire jusqu'au point 399. Et comme vous disposez en outre de plusieurs pages écran, une commutation rapide entre les deux pages crée un effet de couleur intéressant.

A la fin de cette section, n'oublions pas de signaler qu'il existe un autre mode 256 couleurs au-dessus du mode 320*400 points. Mais il n'est pas reconnu par toutes les cartes VGA. Il dispose d'une résolution de 360*480 points affiche un nombre de points nettement plus important sur l'écran que le mode 320*400 points. L'inconvénient est que la taille magique de 128 Ko par page écran est franchie ce qui ne permet plus de gérer simultanément deux pages écran dans la RAM vidéo.

A notre avis, ces arguments parlent en faveur du mode 320*400 points. Comparé au mode ordinaire 256 couleurs, il affiche de toute manière pas moins de 100 % de points sur l'écran. Mais si vous souhaitez augmenter le nombre de points en mode 256 couleurs, il ne vous reste plus qu'à faire appel aux cartes Super VGA et s'en tenir à leur programmation puisqu'elles sont capables de fournir des résolutions encore plus élevées.

Listing : V3240C.C

```
/*****************************************************************
*                        V 3 2 4 0 C . C                         *
******************************************************************
* Fonction        : Montre comment programmer la carte VGA       *
*                   en mode graphique 320*400 en 256 couleurs avec*
*                   deux pages d'écran. Le programme utilise les *
*                   routines en assembleur du module V3240CA.ASM *
******************************************************************
* Auteur          : MICHAEL TISCHER                              *
* Développé le    : 4.09.1990                                    *
* Dernière MAJ    : 14.02.1992                                   *
******************************************************************
* Modèle mémoire : SMALL                                         *
******************************************************************
* (MICROSOFT C)                                                  *
* Compilation     : CL /AS v3240c.c v3240ca                      *
******************************************************************
* (BORLAND TURBO C)                                              *
* Compilation     : Utiliser un fichier de projet avec le contenu*
*                   suivant                                      *
*                     v3240c.c                                   *
*                     v3240ca.obj                                *
******************************************************************
* Appel           : v3240c                                       *
*****************************************************************/

#include <dos.h>
#include <stdarg.h>
#include <stdlib.h>
#include <stdio.h>
#include <conio.h>

/*-- Déclarations de types ------------------------------------*/

typedef unsigned char BYTE;

/*-- Références externes aux routines en assembleur-----------*/

extern void init320400( void );
extern void setpix( int x, int y, unsigned char couleur);
extern BYTE getpix( int x, int y );
extern void setpage( BYTE page );
extern void showpage( BYTE page );
extern void far * getfontptr( void );

/*-- constantes ----------------------------------------------*/

#define MAXX 319                          /* Coordonnées maximales */
#define MAXY 399

/*****************************************************************
* IsVga: Teste la présence d'une carte VGA.                      *
******************************************************************
* Entrée : néant                                                 *
* Sortie : 0  si pas de carte VGA, sinon # 0                     *
*****************************************************************/

BYTE IsVga( void )
{
 union REGS Regs;                 /* Registres pour gérer l'interruption */

 Regs.x.ax = 0x1a00;              /* La fonction 1AH n'existe qu'en VGA */
 int86( 0x10, &Regs, &Regs );
 return ( Regs.h.al == 0x1a );                /* Est-elle disponible ? */
}
/*****************************************************************
* PrintChar : Ecrit un caractère en dehors de la zone visible    *
*             de la mémoire d'écran                              *
******************************************************************
* Entrée  :   caractere = caractère à afficher                   *
*             x, y      = Coordonnées du coin supérieur gauche   *
*             cc        = Couleur du caractère                   *
*             cf        = Couleur du fond                        *
* Info    : Le caractère est dessiné dans une matrice de 8*8 pixels -*
*           sur la base du jeu de caractères 8*8 en ROM          *
*****************************************************************/

void PrintChar( char caractere, int x, int y, BYTE cc, BYTE cf )
{
 typedef BYTE   CARDEF[256][8];     /* Structure du jeu de caractères */
 typedef CARDEF far *CARPTR;        /* Pointe sur un jeu de caractères */

 BYTE     i, k,                                /* Compteur d'itérations */
          masque;          /* Masque binaire pour dessiner le caractère */

 static   CARPTR fptr = (CARPTR) 0;       /* Jeu de caractères en ROM */

 if( fptr == (CARPTR) 0 )       /* A-t-on déjà déterminé ce pointeur ? */
   fptr = getfontptr();     /* Non, détermine avec fonction assembleur */

  /*- Dessine le caractère pixel par pixel ----------------------*/

 if( cf == 255 )                              /* Caractère transparent ? */
   for ( i = 0; i < 8; ++i )  /* Oui, ne dess. que pixels du 1er plan */
   {
     masque = (*fptr)[caractere][i];  /* Motif binaire pour une ligne */
     for( k = 0; k < 8; ++k, masque <<= 1 )  /* Parcourt les colonnes */
       if( masque & 128 )                        /* Pixel à dessiner ? */
         setpix( x+k, y+i, cc );                           /* Oui */
   }
 else                                    /* Non dessine chaque pixel */
   for( i = 0; i < 8; ++i )                   /* Parcourt les lignes */
   {
     masque = (*fptr)[caractere][i];  /* Motif binaire pour une ligne */
     for( k = 0; k < 8; ++k, masque <<= 1 )  /* Parcourt les colonnes */
       setpix( x+k, y+i, (BYTE) (( masque & 128 ) ? cc : cf) );
   }
}

/*****************************************************************
* Line: Trace un segment dans la fenêtre graphique en appliquant *
*       l'algorithme de Bresenham                                *
******************************************************************
* Entrées : X1, Y1 = Coordonnées de l'origine                    *
*           X2, Y2 = Coordonnées de l'extrémité terminale        *
*           COULEUR = couleur du segment                         *
*****************************************************************/

/*-- Fonction accessoire pour échanger deux variables entières -------*/

void SwapInt( int *i1, int *i2 )
{
 int dummy;

 dummy = *i2;  *i2  = *i1;  *i1  = dummy;
}

/*-- Procédure principale --------------------------------------*/

void Line( int x1, int y1, int x2, int y2, BYTE couleur )
{
 int d, dx, dy,
     aincr, bincr,
     xincr, yincr,
     x, y;

 if( abs(x2-x1) < abs(y2-y1) )     /* Sens du parcours : axe X ou Y ? */
 {                                                      /* Par Y */
   if ( y1 > y2 )                          /* y1 plus grand que y2 ? */
   {
     SwapInt( &x1, &x2 );                   /* Oui échange X1 et X2, */
     SwapInt( &y1, &y2 );                                /* Y1 et Y2 */
   }

   xincr = ( x2 > x1 ) ?  1 : -1;          /* Fixe le pas horizontal */

   dy    = y2 - y1;
   dx    = abs( x2-x1 );
```

```
    d     = 2 * dx - dy;
    aincr = 2 * (dx - dy);
    bincr = 2 * dx;
    x     = x1;
    y     = y1;

    setpix( x, y, couleur );                    /* dessine le premier pixel */
    for( y=y1+1; y<= y2; ++y )                  /* Parcourt l'axe des Y */
    {
      if( d >= 0 )
      {
        x += xincr;
        d += aincr;
      }
      else
        d += bincr;
      setpix( x, y, couleur );
    }
  }
  else                                                      /* par X */
  {
    if( x1 > x2 )                           /* x1 plus grand que x2? */
    {
      SwapInt( &x1, &x2 );                  /* Oui, échange X1 et X2 */
      SwapInt( &y1, &y2 );                                  /* Y1 et Y2 */
    }

    yincr = ( y2 > y1 ) ? 1 : -1;           /* Fixe le pas vertical */

    dx    = x2 - x1;
    dy    = abs( y2-y1 );
    d     = 2 * dy - dx;
    aincr = 2 * (dy - dx);
    bincr = 2 * dy;
    x     = x1;
    y     = y1;

    setpix( x, y, couleur );                    /* Dessine le premier pixel */
    for( x=x1+1; x<=x2; ++x )                   /* Parcourt l'axe des X */
    {
      if( d >= 0 )
      {
        y += yincr;
        d += aincr;
      }
      else
        d += bincr;
      setpix( x, y, couleur );
    }
  }
}

/*****************************************************************
*  GrafPrintf: Affiche une chaîne formatée sur l'écran graphique  *
*****************************************************************
*  Entrées : X, Y   = Coordonnées de départ (0 - ...)            *
*            CC     = Couleur des caractères                     *
*            CF     = Couleur du fond  (255 = transparent)       *
*            STRING = Chaîne avec indications de formatage       *
*            ...    = Expressions comme pour printf              *
*****************************************************************/

void GrafPrintf( int x, int y, BYTE cc, BYTE cf, char * string, ... )
{
  va_list  parameter;     /* Liste de paramètres pour les macros VA_... */
  char     affichage[255].          /* Buffer pour la chaîne formatée */
           *cp;

  va_start( parameter, string );            /* Convertit les paramètres */
  vsprintf( affichage, string, parameter );                 /* Formate */
  for( cp = affichage; *cp; ++cp, x+= 8 )           /* Affiche la chaîne */
    PrintChar( *cp, x, y, cc, cf );         /* formatée par PrintChar */
}

/*****************************************************************
*  ColorBox: Dessine un rect. et le remplit avec un motif composé de  *
*            lignes                                              *
*****************************************************************
*  Entrées : X1, Y1 = Coordonnées du coin sup. gauche de la fenêtre   *
*            X2, Y2 = Coordonnées du coin inf. droit de la fenêtre    *
*            COULMAX= code couleur maximal                        *
*  Info    : Les couleurs des lignes sont répétées dans un cycle de 0 *
*            à COULMAX                                            *
*****************************************************************/

void ColorBox( int x1, int y1, int x2, int y2, int coulmax )
{
  int  x, y,                                    /* Variables d'itération */
       sx, sy;                  /* Point de départ de la dernière boucle */

  Line( x1, y1, x1, y2, 15 );                   /* Encadre le rectangle */
  Line( x1, y2, x2, y2, 15 );
  Line( x2, y2, x2, y1, 15 );
  Line( x2, y1, x1, y1, 15 );

  for( y = y2-1; y > y1; --y )      /* du coin inf gauche au bord droit */
    Line( x1+1, y2-1, x2-1, y, (BYTE) (y % coulmax) );

  for( y = y2-1; y > y1; --y )      /* du coin inf droit au bord gauche */
    Line( x2-1, y2-1, x1+1, y, (BYTE) (y % coulmax) );

  /*-- Du milieu du rectangle au bord supérieur ----------------------*/

  for( x=x1+1, sx=x1+(x2-x1)/2, sy=y1+(y2-y1)/ 2; x < x2; ++x )
    Line( sx, sy, x, y1+1, (BYTE) (x % coulmax) );
}

/*****************************************************************
*  DrawAxis: Dessine des axes à gauche et en haut de l'écran      *
*****************************************************************
*  Entrées : XSTEP = Pas selon l'axe X                            *
*            YSTEP = Pas selon l'axe Y                            *
*            CC    = Couleur des caractères                       *
*            CF    = Couleur de fond  (255 = transparent)         *
*****************************************************************/

void DrawAxis( int stepx, int stepy, BYTE cc, BYTE cf )
{
  int x, y;                                 /* Variables d'itération */

  Line( 0, 0, MAXX, 0, cc );                        /* Trace l'axe X */
  Line( 0, 0, 0, MAXY, cc );                        /* Trace l'axe Y */

  for( x = stepx; x < MAXX; x += stepx )            /* Gradue l'axe X */
  {
    Line( x, 0, x, 5, cc );
    GrafPrintf( x < 100 ? x - 8 : x - 12, 8, cc, cf, "%d", x );
  }

  for( y = stepy; y < MAXY; y += stepy )            /* Gradue l'axe Y */
  {
    Line( 0, y, 5, y, cc );
    GrafPrintf( 8, y-4, cc, cf, "%3d", y );
  }
}


/*****************************************************************
*  Demo: Démontre l'usage des différentes fonctions de ce module  *
*****************************************************************/

void Demo( void )
{
#define PAUSE 100000            /* Durée de pause, dépend du système */

  int   x;                                  /* Compteur d'itérations */
  long  delay;                              /* Compteur de pause */

  ColorBox( 80, 50, 308, 350, 16 );     /* Dessine un rectangle coloré */
  DrawAxis( 30, 40, 15, 255 );                      /* Trace des axes */
  GrafPrintf( 46, MAXY-10, 15, 255, "V3240C  -  (c) by MICHAEL TISCHER");

  setpage( 1 );                             /* Active la page 1 */
  ColorBox( 80, 50, 308, 350, 255 );    /* Dessine un rectangl coloré */
  DrawAxis( 30, 40, 15, 255 );                      /* Trace des axes */
  GrafPrintf( 46, MAXY-10, 15, 255, "V3240C  -  (c) by MICHAEL TISCHER");

  /*-- Affiche alternativement les deux pages graphiques --------------*/

  for( x = 0; x < 50; ++x )                         /* 50 passages */
  {
    showpage( (BYTE) (x % 2) );                     /* Affiche une page */
    for( delay = 1; delay < PAUSE; ++delay );       /* Petite pause */
  }
}

/*-----------------------------------------------------------------*/
/*--                    PROGRAMME PRINCIPAL                      --*/
/*-----------------------------------------------------------------*/

void main( void )
{
  union REGS  regs;

  if( IsVga() )                             /* A-t-on une carte VGA ? */
  {                                                 /* Oui, c'est parti */
    init320400();                       /* Initialise le mode graphique */
    Demo();
    getch();                                    /* Attend une frappe*/
    regs.x.ax = 0x0003;                         /* retourne au mode texte */
    int86( 0x10, &regs, &regs );
  }
  else
    printf( "V3240C - (c) 1990, 92 by MICHAEL TISCHER\n\nATTENTION "\
            "Ce programme exige une carte VGA .\n\n" );
}
```

Listing : V3240CA.ASM

```
;*****************************************************************;
;*                        V3240CA.ASM                            *;
;*---------------------------------------------------------------*;
;*   Fonction       : contient diverses routines pour travailler *;
;*                    dans le mode graphique 320*400 en 256 couleurs *;
;*                    de la carte VGA                            *;
;*---------------------------------------------------------------*;
;*   Auteur         : MICHAEL TISCHER                            *;
;*   Développé le   :  5.09.1990                                 *;
;*   Dernière MAJ   : 14.01.1991                                 *;
;*---------------------------------------------------------------*;
;*   Assemblage     : MASM /mx V3240CA;    ou   TASM -mx V3240CA *;
;*                    ... puis lier à V3240C.C                   *;
;*****************************************************************;

IGROUP group _text              ; Regroupe les segments de programme
DGROUP group const, _bss, _data ;Regroupe les segments de données
        assume CS:IGROUP, DS:DGROUP, ES:DGROUP, SS:DGROUP

CONST   segment word public 'CONST'
CONST   ends

_BSS    segment word public 'BSS'       ;Segment des variables statiques
_BSS    ends                                    ;non initialisées

_DATA   segment word public 'DATA'      ;Segment des variables globales
                                        ;et statiques non initialisées


_DATA   ends

;== Constantes ================================================

SC_INDEX        = 3c4h    ;Registre d'index du contrôleur du séquenceur .
SC_MAP_MASK     = 2                     ;Numéro du registre Map-Mask
SC_MEM_MODE     = 4                 ;Numéro du registre de mode mémoire

GC_INDEX        = 3ceh          ;registre d'index du contrôleur graphique
GC_READ_MAP     = 4                     ;Numéro du registre Read Map
GC_GRAPH_MODE   = 5                 ;Numéro du registre de mode graphique
GC_MISCELL      = 6                     ;Numéro du registre divers

CRTC_INDEX      = 3d4h          ;registre d'index du contrôleur d'écran
CC_MAX_SCAN     = 9     ;Numéro du registre du maximum de lignes balayées
CC_START_HI     = 0Ch                   ;Numéro du registre HI_Start
CC_UNDERLINE    = 14h               ;Numéro du registre de soulignement
CC_MODE_CTRL    = 17h           ;Nummer du registre de contrôle de mode

VERT_RETRACE    = 3DAh                  ;Registre d'état d'entrée #1

PIXX            = 320                       ;Résolution horizontale

;== Données ===================================================

_DATA   segment word public 'DATA'

vio_seg     dw 0a000h                   ;Segment de la mémoire d'écran
                                        ;avec la page courante

_DATA   ends

;== Programme =================================================

_TEXT   segment byte public 'CODE'              ;Segment de programme

;-- Public-Déklarationen ------------------------------------------

public    _init320400                   ;Initialise le mode 320*400
public    _setpix                               ;Dessine un point
public    _getpix                   ;Détermine la couleur d'un pixel
public    _showpage                         ;Affiche la page 0 ou 1
public    _setpage                  ;Fixe la page pour setpix et getpix
public    _getfontptr ;Retourne un pointeur sur le jeu de caractères 8*8

;-- INIT320400: initialise le mode 320*400 ------------------------
;-- Déclaration : void init320400( void ); ------------------------

_init320400 proc near

        ;-- Commence par installer le mode 13h pour que le BIOS ---------
        ;-- effectue la plus grande partie de l'initialisation ----------
        ;-- Puis modifie les registres qui n'ont pas encore le ----------
        ;-- contenu souhaité ---------------------------------------

        mov   ax,0013h                          ;Déclenche le mode 13h
        int   10h

        mov   dx,GC_INDEX           ;Désactive la séparation des adresses
        mov   al,GC_GRAPH_MODE      ;mémoire dans registre mode graphique
        out   dx,al                             ;du contrôleur graphique
        inc   dx                          ;en manipulant le bit numéro 4
        in    al,dx
        and   al,11101111b
        out   dx,al
        dec   dx

        mov   al,GC_MISCELL                 ;idem dans le registre divers
        out   dx,al                       ;en manipulant le bit numéro 1
        inc   dx                                                        ;
        in    al,dx
        and   al,11111101b
        out   dx,al

        mov   dx,SC_INDEX           ;Modifie le registre de mode mémoire
        mov   al,SC_MEM_MODE          ;dans le contrôleur du séquenceur
        out   dx,al                 ;de façon à mettre fin à la répartition
        inc   dx                    ;des adresses sur plusieurs plans de bits
        in    al,dx                 ;et à fixer le plan courant par le registre
        and   al,11110111b                          ;de masquage binaire
        or    al,4                                                      ;
        out   dx,al

        mov   ax,vio_seg                    ;Remplit les quatre plans bits
        mov   es,ax                           ;avec le code couleur 00h
        xor   di,di                                   ;et efface l'écran
        mov   ax,di


        mov   cx,8000h
        rep   stosw

        mov   dx,CRTC_INDEX ;Met fin au dédoublement des lignes de pixels
        mov   al,CC_MAX_SCAN          ;en manipulant le bit 7 du registre

        out   dx,al           ;du maximum de lignes balayées du contrôleur
        inc   dx        ;fixe en même temps à 1 la hauteur des caractères
        in    al,dx                                   ;par les bits 0-4
        and   al,01110000b                                              ;
        out   dx,al
        dec   dx              ;DX = registre d'index du contrôleur d'écran

        mov   al,CC_UNDERLINE                 ;Arrête le mode double mot
        out   dx,al                             ;en manipulant le bit 6
        inc   dx                          ;du registre de soulignement
        in    al,dx
        and   al,10111111b
        out   dx,al
        dec   dx

        mov   al,CC_MODE_CTRL           ;Passe du mode mot au mode octet
        out   dx,al                   ;par manipulation du bit 6 du registre
        inc   dx                                    ;de contrôle de mode
        in    al,dx
        or    al,01000000b
        out   dx,al

        ret                                         ;Rtourne à l'appelant

_init320400 endp                                    ;Fin de la procédure

;-- SETPIX: Dessine un pixel dans une couleur donnée ------------------
;-- Déclaration : void setpix( int x, int y, unsigned char couleur );

_setpix    proc near

sframe     struc                            ;Structure d'accès à la pile
bp0        dw ?                                          ;Mémorise BP
ret_adr0   dw ?                       ;Adresse de retour à l'appelant
x0         dw ?                                          ;Abscisse X
y0         dw ?                                          ;Ordonnée Y
couleur    dw ?                                             ;Couleur
sframe     ends                                 ;Fin de la structure

frame  equ [ bp - bp0 ]           ;adresse les éléments de la structure

        push  bp                    ;Prépare l'adressage des paramètres
        mov   bp,sp                                 ; par le registre BP

        push  di                              ;Sauvegarde DI sur la pile

        mov   ax,PIXX / 4           ;Calcule l'offset en mémoire d'écran
        mul   frame.y0                                  ; et le charge en DI
        mov   cx,frame.x0
```

```
        mov   bx,cx
        shr   bx,1
        shr   bx,1
        add   ax,bx
        mov   di,ax

        and   cl,3                      ;Calcule en AH le masque binaire
        mov   ah,1                              ; pour adresser le plan
        shl   ah,cl
        mov   al,SC_MAP_MASK                    ;Numéro du registre en AL
        mov   dx,SC_INDEX           ;Charge l'adresse d'index du séquenceur
        out   dx,ax                 ;Charge le registre de masquage binaire

        mov   ax,vio_seg            ;ES doit pointer sur la mémoire d'écran
        mov   es,ax
        mov   al,byte ptr frame.couleur       ;Charge la couleur du pixel
        stosb                           ; et la place dans le plan de bits
        pop di                                  ; Dépile le registre DI
        pop   bp                                        ; ainsi que BP

        ret                                             ;Retourne à l'appelant

_setpix   endp                                          ;Fin de la procédure

;-- GETPIX: détermine la couleur d'un pixel  -------------------------
;-- Déclaration : unsigned char getpix( int x, int y ); ---------------

_getpix   proc near

sframel   struc                                 ;Structure d'accès à la pile
bpl       dw ?                                          ;Mémorise BP
ret_adrl  dw ?                          ;Adresse de retour à l'appelant
xl        dw ?                                          ;Abscisse X
yl        dw ?                                          ;Ordonnée Y
sframel   ends                                  ;Fin de la structure

frame  equ [ bp - bpl ]             ;Adresse les éléments de la structure

        push  bp                    ;Prépare l'adressage des paramètres
        mov   bp,sp                             ; par le registre BP

        push  si                                ;Sauvegarde SI sur la pile

        mov   ax,PIXX / 4           ;Calcule l'offset en mémoire et le
        mul   frame.yl                          ; charge en SI
        mov   si,frame.xl
        mov   cx,si
        shr   si,1
        shr   si,1
        add   si,ax

        and   cl,3                  ;Calcule en AH le masque binaire pour
        mov   ah,cl                             ;adresser le plan
        mov   al,GC_READ_MAP                    ;Numéro du registre en AL
        mov   dx,GC_INDEX       ;Adresse d'index du contrôleur graphique
        out   dx,ax                     ;Charge le registre Read Map

        mov   ax,vio_seg            ;ES doit pointer sur la mémoire d'écran
        mov   es,ax
        mov   al,es:[si]                ;Charge la couleur du pixel
        pop   si                    ;Reprend les registres sur la pile
        pop   bp

        ret                                     ;Retourne à l'appelant

_getpix   endp                                  ;Fin de la procédure

;-- SETPAGE: Sélectionne la page concernée par les appels aux fonctions-
;--          setpix et getpix
;-- Déclaration : void setpage( unsigend char page );

_setpage  proc near

        pop   bx                        ;Dépile l'adresse de retour
        pop   ax                                ; et l'argument

        push  ax                                ; puis le replace sur la pile
        push  bx

        mov   bl,0a0h                           ;
        or    al,al                                     ;Est-ce la page 0?
        je    sp1                               ;Oui, mémorise le segment
        mov   bl,0a8h                             ;Non c'est la page 1

sp1:    mov   byte ptr vio_seg + 1,bl       ;Nouvelle adresse segment

        ret                                     ;retourne à l'appelant

_setpage  endp                                  ;Fin de la procédure

;-- SHOWPAGE: Affiche l'une des deux pages d'écran --------------------
;-- Déclaration : void showpage( unsigend char page ); ----------------

_showpage proc near

        pop   bx                        ;Dépile l'adresse de retour
        pop   ax                                ; et l'argument

        push  ax                        ;Puis les replace sur la pile
        push  bx

        or    al,al                             ;Est-ce la page 0?
        je    sp2    ;Oui, son numéro est égal à l'octet fort de l'offset

        mov   al,80h                    ;Non, page 1, avec offset 8000h

sp2:    mov   dx,CRTC_INDEX             ;Adresse le contrôleur d'écran
        mov   ah,al             ;Charge en AH l'octet fort de l'offset
        mov   al,CC_START_HI                    ;Numéro du registre en AL
        out   dx,ax                             ;Effectue la sortie

        ;-- Attend un début de rafraîchissement d'écran ---------------

        mov   dx,VERT_RETRACE       ;Attend la fin du retour de balayage
sp3:    in    al,dx                                       ;vertical
        test  al,8
        jne   sp3

sp4:    in    al,dx                 ;puis le début du retour du faisceau
        test  al,8
        je    sp4

        ret                                     ;Retourne à l'appelant

_showpage endp                                  ;Fin de la procédure

;-- GETFONTPTR: Renvoie un pointeur FAR sur le jeu de caractères 8*8 ---
;-- Déclaration : void far * getfontptr( void )

_getfontptr proc near

        push  bp                                        ;Sauve BP

        mov   ax,1130h                  ;Charge les registres puis
        mov   bh,3
        int   10h                   ; déclenche l'interruption vidéo

        mov   dx,es                     ;Transfère ES:BP dans DX:AX
        mov   ax,bp

        pop   bp                                ;Reprend BP sur la pile
        ret                                     ;Retourne à l'appelant

_getfontptr endp                                ;Fin de la procédure

;-- Fin ---------------------------------------------------------------

_text   ends                                    ;Fin du segment de programme
        end                             ;Fin de la source en assembleur
```

Listing : V3240P.PAS

```
{*****************************************************************
*                          V 3 2 4 0 P . P A S                   *
*****************************************************************
*  Fonction        : Montre comment programmer la carte VGA dans le  *
*                    mode graphique 320*400 en 256 couleurs avec deux *
*                    pages d'écran. Le programme utilise les routines *
*                    en assembleur du module V3240PA.ASM           *
*****************************************************************
*  Auteur          : MICHAEL TISCHER                               *
*  Développé le    : 08.09.1990                                    *
*  Dernière MAJ    : 14.01.1991                                    *
*****************************************************************}

program V3240P;

uses dos, crt;

{-- Déclarations de types ---------------------------------------}

type BPTR = ^byte;

{-- Références externes aux routines en assembleur -------------}

{$L v3240pa}          { Intègre le module en assembleur }

procedure init320400; external;
procedure setpix( x, y : integer; couleur : byte ); external;
function  getpix( x, y: integer ) : byte ; external;
procedure setpage( page : byte ); external;
procedure showpage( page : byte ); external;

{-- Constantes -------------------------------------------------}

const MAXX = 319;                              { Coordonnées maximales }
      MAXY = 399;

{*****************************************************************
*  IsVga : Teste la présence d'une carte VGA.                    *
*****************************************************************
*  Entrée  : néant                                               *
*  Sortie  : TRUE ou FALSE                                       *
*****************************************************************}

function IsVga : boolean;

var Regs : Registers;                  { Registres pour l'interruption}

begin
  Regs.AX := $1a00;                { La fonction 1Ah n'existe qu'en VGA }
  Intr( $10, Regs );
  IsVga := ( Regs.AL = $1a );
end;

{*****************************************************************
*  PrintChar : Affiche un caractère en mode graphique            *
*****************************************************************
*  Entrée : caractere = le caractère à afficher                  *
*           x, y      = Coordonnées du coin sup gauche           *
*           cc        = Couleur du caractère                     *
*           cf        = Couleur du fond                          *
*  Info   : Le caractère est dessiné dans une matrice de 8*8 pixels *
*           sur la base du jeu de caractères 8*8 en ROM           *
*****************************************************************}

procedure PrintChar( caractere : char; x, y : integer; cc, cf : byte );

type CARADEF = array[0..255,0..7] of byte; {Structure jeu de caractères}
     CARAPTR = ^CARADEF;             { Pointe sur le jeu de caractères }

var  Regs  : Registers;       { Registres pour gérer les interruptions }
     ch    : char;                                  { Pixel du caractère }
     i, k,                                  { Compteur d'itérations }
     Masque : byte;          { Masque binaire pour dessiner le caractère}

const fptr : CARAPTR = NIL;   { Pointe sur le jeu de caractères en ROM }

begin
  if fptr = NIL then              { A-t-on déjà déterminé ce pointeur ? }
    begin                                                  { Non}
      Regs.AH := $11;                    { Appelle l'option $1130 de la }
      Regs.AL := $30;                          { fonction vidéo du BIOS }
      Regs.BH := 3;              { pour obtenir un pointeur sur le jeu 8*8 }
      intr( $10, Regs );
      fptr := ptr( Regs.ES, Regs.BP );             { Compose le pointeur}
    end;

  if ( cf = 255 ) then                          { Caractère transparent ? }
    for i := 0 to 7 do  { Ont ne dessine que les pixels du premier plan}
      begin
        Masque := fptr^[ord(caractere),i];{Motif binaire pour une ligne}
        for k := 0 to 7 do
          begin
            if ( Masque and 128 <> 0 ) then        { Pixel à dessiner? }
              setpix( x+k, y+i, cc );                          { Oui }
            Masque := Masque shl 1;
          end;
      end
  else                                          { Non, tient compte du fond }
    for i := 0 to 7 do                            { Parcourt les lignes }
      begin
        Masque := fptr^[ord(caractere),i];{Motif binaire pour une ligne}
        for k := 0 to 7 do
          begin
            if ( Masque and 128 <> 0 ) then          { Premier plan ? }
              setpix( x+k, y+i, cc )                               { Oui }
            else
              setpix( x+k, y+i, cf );                        { Non, fond}
            Masque := Masque shl 1;
          end;
      end;
end;

{*****************************************************************
*  Line : Trace un segment dans la fenêtre graphique en appliquant *
*         l'algorithme de Bresenham                              *
*****************************************************************
*  Entrée : X1, Y1  = Coordonnées de l'origine (0 - ...)         *
*           X2, Y2  = Coordonnées de l'extrémité terminale       *
*           COULEUR = couleur du segment                         *
*****************************************************************}

procedure Line( x1, y1, x2, y2 : integer; couleur : byte );

var d, dx, dy,
    aincr, bincr,
    xincr, yincr,
    x, y                  : integer;

{-- Procédure accessoire pour échanger deux variables entières --------}

procedure SwapInt( var i1, i2: integer );

var dummy : integer;

begin
  dummy := i2;
  i2    := i1;
  i1    := dummy;
end;

{-- Procédure principale ----------------------------------------}

begin
  if ( abs(x2-x1) < abs(y2-y1) ) then    { Parcours : axe des X ou Y ? }
    begin                                                { par l'axe des Y }
      if ( y1 > y2 ) then                                { y1 supérieur à y2? }
        begin
      SwapInt( x1, x2 );                          { Oui, échange X1 et X2, }
      SwapInt( y1, y2 );                                    { Y1 et Y2 }
        end;

      if ( x2 > x1 ) then xincr := 1          { Fixe le pas horizontal }
                     else xincr := -1;

      dy := y2 - y1;
      dx := abs( x2-x1 );
      d  := 2 * dx - dy;
      aincr := 2 * (dx - dy);
      bincr := 2 * dx;
      x := x1;
      y := y1;

      Setpix( x, y, couleur );                { Dessine le premier point }
      for y:=y1+1 to y2 do                      { Parcourt l'axe des Y }
        begin
          if ( d >= 0 ) then
            begin
              inc( x, xincr );
              inc( d, aincr );
            end
          else
            inc( d, bincr );
          Setpix( x, y, couleur );
        end;
```

```
    end
  else                                              { par l'axe des X }
    begin
      if ( x1 > x2 ) then                          { x1 plus grand que x2? }
        begin
      SwapInt( x1, x2 );                           { Oui, échange X1 et X2 }
      SwapInt( y1, y2 );                                  { Y1 et Y2 }
        end;

      if ( y2 > y1 ) then yincr := 1               { Fixe le pas vertical }
                     else yincr := -1;

      dx := x2 - x1;
      dy := abs( y2-y1 );
      d  := 2 * dy - dx;
      aincr := 2 * (dy - dx);
      bincr := 2 * dy;
      x := x1;
      y := y1;

      Setpix( x, y, couleur );                     { Dessine le premier point }
      for x:=x1+1 to x2 do                           { Parcourt l'axe des X }
        begin
          if ( d >= 0 ) then
          begin
            inc( y, yincr );
            inc( d, aincr );
          end
        else
          inc( d, bincr );
        Setpix( x, y, couleur );
      end;
    end;
end;

{*********************************************************************
* GrafPrint: Affiche une chaîne formatée sur l'écran graphique       *
**-----------------------------------------------------------------**
* Entrées : X, Y   = Coordonnées de départ (0-...)                   *
*           CC     = Couleur des caractères                          *
*           CF     = Couleur du fond    (255 = transparent)          *
*           STRING = Chaîne avec indications de formatage            *
*********************************************************************}

procedure GrafPrint( x, y : integer; cc, cf : byte; strt : string );

var i : integer;                                   { Compteur d'itérations }

begin
  for i:=1 to length( strt ) do
    begin
      printchar( strt[i], x, y, cc, cf );        { Affiche par printchar }
      inc( x, 8 );                      { x à la position du caractère suivant }
    end;
end;

{*********************************************************************
* ColorBox : Dessine un rectangle et le remplit avec un motif composé *
*            de lignes.                                              *
**-----------------------------------------------------------------**
* Entrées  : X1, Y1 = Coordonnées du coin sup gauche de la fenêtre   *
*            X2, Y2 = Coordonnées du coin inf droit de la fenêtre    *
*            COULMAX= Code de couleur maximal                        *
* Info     : Les couleurs des lignes sont répétées dans un cycle de 0 *
*            à COULMAX                                               *
*********************************************************************}

procedure ColorBox( x1, y1, x2, y2 : integer; coulmax : byte );

var x, y,                                          { Compteur d'itérations }
    sx, sy : integer;             { Point de départ de la dernière boucle }

begin
  Line( x1, y1, x1, y2, 15 );                  { Cadre autour du rectangle}
  Line( x1, y2, x2, y2, 15 );
  Line( x2, y2, x2, y1, 15 );
  Line( x2, y1, x1, y1, 15 );

  for y := y2-1 downto y1+1 do          { Du coin inf gauche au bord droit }
    Line( x1+1, y2-1, x2-1, y, y mod coulmax );

  for y := y2-1 downto y1+1 do          { Du coin inf droit au bord gauche }
    Line( x2-1, y2-1, x1+1, y, y mod coulmax );

  {-- Du milieu du rectangle au bord supérieur ----------------------}

  sx := x1+ (x2-x1) div 2;
  sy := y1+ (y2-y1) div 2;
  for x := x1+1 to x2-1 do
    Line( sx, sy, x, y1+1, x mod coulmax );
end;

{*********************************************************************
* DrawAxis: Dessine des axes en haut et à gauche de l'écran          *
**-----------------------------------------------------------------**
* Entrées : XSTEP = Pas selon l'axe X                                *
*           YSTEP = Pas selon l'axe Y                                *
*           CC    = Couleur des caractères                           *
*           CF    = Couleur de fond (255 = transparent)              *
*********************************************************************}

procedure DrawAxis( stepx, stepy : integer; cc, cf : byte );

var x, y     : integer;                          { Variables d'itération }
    grad : string[3];

begin
  Line( 0, 0, MAXX, 0, cc );                              { Trace l'axe X }
  Line( 0, 0, 0, MAXY, cc );                              { Trace l'axe Y }

  x := stepx;                                             { Gradue l'axe X }
  while ( x < MAXX ) do
    begin
      Line( x, 0, x, 5, cc );
      str( x, grad );
      if ( x < 100 ) then
        GrafPrint( x - 8 , 8, cc, cf, grad )
      else
        GrafPrint( x - 12, 8, cc, cf, grad );
      inc( x, stepx );
    end;

  y := stepy;                                             { Gradue l'axe Y }
  while ( y < MAXY ) do
    begin
      Line( 0, y, 5, y, cc );
      str( y:3, grad );
      GrafPrint( 8, y-4, cc, cf, grad );
      inc( y, stepy );
    end;
end;

{*********************************************************************
* Demo : Démontre l'usage des différentes fonxtions de ce module     *
**-----------------------------------------------------------------**
* Entrées : néant                                                    *
*********************************************************************}

procedure Demo;

const PAUSE = 100;                          { Compteur de millisecondes }

var page : byte;                                  { Compteur de pages }

begin
  ColorBox( 80, 50, 308, 350, 16 );       { Dessine un rectangle coloré }
  DrawAxis( 30, 40, 15, 255 );                          { Trace des axes }
  GrafPrint( 46, MAXY-10, 15, 255, 'V3240P  -  (c) by MICHAEL TISCHER');

  setpage( 1 );                                         { Passe en page 1 }
  ColorBox( 80, 50, 308, 350, 255 );      { Dessine un rectangle coloré }
  DrawAxis( 30, 40, 15, 255 );                          { Trace des axes }
  GrafPrint( 46, MAXY-10, 15, 255, 'V3240P  -  (c) by MICHAEL TISCHER');

  {-- Affiche alternativement les deux pages graphiques----------------}

  for page := 0 to 50 do                                  { 50 passages }
    begin
      showpage( page mod 2 );                       { Affiche une page }
      delay( PAUSE );                                    { Petite pause}
    end;

end;

{---------------------------------------------------------------------}
{--                        PROGRAMME PRINCIPAL                      --}
{---------------------------------------------------------------------}

begin
  if IsVga then                                  { A-t-on une carte VGA? }
    begin                                              { Oui, c'est parti }
      init320400;                              { Initialise le mode graphique }
      Demo;
      repeat until keypressed;
      Textmode( C080 );                            { Retourne au mode texte }
    end
  else
    writeln( 'V3240P  -  (c) 1990 by MICHAEL TISCHER'#13#10#10 +
             'Attention: ce programme exige une carte VGA' +
             ''#13#10 );
end.
```

Listing : V3240PA.ASM

```
;*****************************************************************;
;*                      V3240PA.ASM                              *;
;*---------------------------------------------------------------*;
;*   Fonction : contient diverses routines pour travailler dans le *;
;*              mode graphique 320*400 de la carte VGA           *;
;*---------------------------------------------------------------*;
;*   Auteur        : MICHAEL TISCHER                             *;
;*   Développé le  : 05.09.1990                                  *;
;*   Dernière MAJ  : 14.01.1991                                  *;
;*---------------------------------------------------------------*;
;*   Assemblage    : MASM /mx V3240PA;    ou    TASM -mx V3240PA *;
;*                   ...puis inclure dans V3240P.PAS           *;
;*****************************************************************;

;-- Constantes -----------------------------------------------------

SC_INDEX      = 3c4h    ;Registre d'index du contrôleur du séquenceur .
SC_MAP_MASK   = 2                    ;Numéro du registre Map-Mask
SC_MEM_MODE   = 4                    ;Numéro du registre de mode mémoire

GC_INDEX      = 3ceh          ;registre d'index du contrôleur graphique
GC_READ_MAP   = 4                    ;Numéro du registre Read Map
GC_GRAPH_MODE = 5                    ;Numéro du registre de mode graphique
GC_MISCELL    = 6                    ;Numéro du registre divers

CRTC_INDEX    = 3d4h          ;registre d'index du contrôleur d'écran
CC_MAX_SCAN   = 9     ;Numéro du registre du maximum de lignes balayées
CC_START_HI   = 0Ch                  ;Numéro du registre Hi_Start
CC_UNDERLINE  = 14h                  ;Numéro du registre de soulignement
CC_MODE_CTRL  = 17h             ;Nummer du registre de contrôle de mode

DAC_WRITE_ADR = 3C8h                 ;Adresse DAC Write
DAC_READ_ADR  = 3C7h                 ;Adresse DAC Read
DAC_DATA      = 3C9h              ;Registres de données DAC

VERT_RETRACE  = 3DAh             ;Registre d'état d'entrée #1

PIXX          = 320                  ;Résolution horizontale

;-- Segment de données ---------------------------------------------

DATA   segment word public

vio_seg   dw (?)                ;Segment de la mémoire d'écran, à
                                 ;initialiser à l'exécution

DATA   ends

;-- Programme ------------------------------------------------------

CODE   segment byte public                  ;Segment de programme

       assume cs:code, ds:data

;-- Déclarations publiques -----------------------------------------

public   init320400                  ;Initialise le mode 320*400
public   setpix                             ;Dessine un pixel
public   getpix                 ;Détermine la couleur d'un pixel
public   showpage                      ;Affiche la page 0 ou 1
public   setpage             ;Fixe la page pour setpix et getpix

;-------------------------------------------------------------------
;-- INIT320400: initialise le mode graphique 320*400
;-- Appel depuis TP: init320400;

init320400 proc near

               ;-- Commence par installer le mode 13h pour que le BIOS
               ;-- effectue la plus grande partie de l'initialisation
               ;-- Puis modifie les registres qui n'ont pas encore le
                                          ;-- contenu souhaité

       mov   ax,0013h                     ;Appelle le mode 13h
       int   10h

       mov   dx,GC_INDEX          ;Désactive la séparation des adresses
       mov   al,GC_GRAPH_MODE;mémoire dans le registre de mode graphique
       out   dx,al                           ;du contrôleur graphique
       inc   dx                         ;en manipulant le bit numéro 4
       in    al,dx
       and   al,11101111b
       out   dx,al
       dec   dx

       mov   al,GC_MISCELL                ;idem dans le registre divers
       out   dx,al                        ;en manipulant le bit numéro 1
       inc   dx                                                      ;
       in    al,dx
       and   al,11111101b
       out   dx,al

       mov   dx,SC_INDEX          ;Modifie le registre de mode mémoire
       mov   al,SC_MEM_MODE             ;dans le contrôleur du séquenceur
       out   dx,al                 ;de façon à mettre fin à la répartition
       inc   dx                  ;des adresses sur plusieurs plans de bits
       in    al,dx               ;et à fixer le plan courant par le registre
       and   al,11110111b                          ;de masquage binaire
       or    al,4                                                    ;
       out   dx,al

       mov   ax,0A000h                 ;Remplit les quatre plans de bits
       mov   vio_seg,ax               ;avec le code couleur 00h et efface
       mov   es,ax                                            ;l'écran
       xor   di,di
       mov   ax,di
       mov   cx,8000h
       rep   stosw

       mov   dx,CRTC_INDEX ;Met fin au dédoublement des lignes de pixels
       mov   al,CC_MAX_SCAN          ;en manipulant le bit 7 du registre

       out   dx,al          ;du maximum de lignes balayées du contrôleur
       inc   dx       ;fixe en même temps à 1 la hauteur des caractères
       in    al,dx                                        ;par les bits 0-4
       and   al,01110000b                                            ;
       out   dx,al
       dec   dx ;DX repointe sur le registre d'index du contrôleur écran

       mov   al,CC_UNDERLINE                 ;Arrête le mode double mot
       out   dx,al                                 ;en manipulant le bit 6
       inc   dx                              ;du registre de soulignement
       in    al,dx
       and   al,10111111b
       out   dx,al
       dec   dx

       mov   al,CC_MODE_CTRL          ;Passe du mode mot au mode octet
       out   dx,al                ;par manipulation du bit 6 du registre
       inc   dx                                      ;de contrôle de mode
       in    al,dx
       or    al,01000000b
       out   dx,al

       ret                                          ;Retourne à l'appelant

init320400 endp                                      ;Fin de la procédure

;-------------------------------------------------------------------
;-- SETPIX: Dessine un pixel dans une couleur donnée
;-- Appel depuis TP: setpix( x , y : integer; couleur : byte );

setpix     proc near

sframe     struc                        ;Structure d'accès à la pile
bp0        dw ?                                    ;Mémorise BP
ret_adr0   dw ?                  ;Adresse de retour à l'appelant
couleur    dw ?                                       ;Couleur
y0         dw ?                                     ;Ordonnée Y
x0         dw ?                                     ;Abscisse X
sframe     ends                                ;Fin de la structure

frame      equ [ bp - bp0 ]       ;Adresse les éléments de la structure

       push  bp                        ;Prépare l'adressage des paramètres
       mov   bp,sp                                  ;par le registre BP

       mov   ax,PIXX / 4          ;Calcule l'offset en mémoire d'écran
       mul   frame.y0                                ; et le charge en DI
       mov   cx,frame.x0
       mov   bx,cx
       shr   bx,1
       shr   bx,1
       add   ax,bx
       mov   di,ax

       and   cl,3                        ;Calcule en AH le masque binaire
       mov   ah,1                                 ; pour adresser le plan
       shl   ah,cl
       mov   al,SC_MAP_MASK                    ;Numéro du registre en AL
       mov   dx,SC_INDEX          ;Charge l'adresse d'index du séquenceur
       out   dx,ax                ;Charge le registre de masquage binaire

       mov   ax,vio_seg           ;ES doit pointer sur la mémoire d'écran
```

```
        mov   es,ax
        mov   al,byte ptr frame.couleur      ;Charge la couleur du pixel
        stosb                                ; et la place dans le plan de bits

        pop   bp                                    ;Dépile le registre BP

        ret   6                          ;retourne à l'appelant en retirant les
                                                    ;arguments de la pile
setpix  endp                                        ;Fin de la procédure

;---------------------------------------------------------------------------
;-- GETPIX: détermine la couleur d'un pixel
;-- Appel depuis TP: x := getpix( x , y : integer );

getpix  proc near

sframe1 struc                                ;Structure d'accès à la pile
bp1     dw ?                                            ;Mémorise BP
ret_adr1 dw ?                           ;Adresse de retour à l'appelant
y1      dw ?                                            ;Ordonnées Y
x1      dw ?                                             ;Abscisse X
sframe1 ends                                        ;Fin de la structure

frame   equ [ bp - bp1 ]            ;adresse les éléments de la structure

        push  bp                      ;Prépare l'adressage des paramètres
        mov   bp,sp                             ; par le registre BP

        mov   ax,PIXX / 4             ;Calcule l'offset en mémoire d'écran
        mul   frame.y1                          ; et le charge en SI
        mov   si,frame.x1
        mov   cx,si
        shr   si,1
        shr   si,1
        add   si,ax

        and   cl,3                    ;Calcule en AH le masque binaire pour
        mov   ah,cl                             ;adresser le plan
        mov   al,GC_READ_MAP                    ;Numéro du registre en AL
        mov   dx,GC_INDEX ;Charge adresse d'index du contrôleur graphique
        out   dx,ax                             ;Charge le registre Read Map

        mov   ax,vio_seg              ;ES doit pointer sur la mémoire d'écran
        mov   es,ax
        mov   al,es:[si]                        ;Charge la couleur du pixel

        pop   bp                                    ;Reprend BP sur la pile

        ret   4                          ;Retourne à l'appelant en retirant les
                                                    ;arguments de la pile
getpix endp                                         ;Fin de la procédure

;---------------------------------------------------------------------------
;-- SETPAGE: Sélectionne la page concernée par les appels aux fonctions
;--          setpix et getpix
;-- Appel depuis TP: setpage( page : byte );

setpage proc near

        pop   ax                                    ;Dépile l'adresse de retour
        pop   cx                                            ; et l'argument

        push  ax                     ;ne remet que la seule adresse de retour

        mov   bl,0a0h                                               ;
        or    cl,cl                                         ;Est-ce la page 0?
        je    sp1                                   ;Oui, mémorise le segment
        mov   bl,0a8h                                     ;Non c'est la page 1

sp1:    mov   byte ptr vio_seg + 1,bl  ;Mémorise la nouvelle adr. de seg.

        ret   ;retourne à l'appelant, le paramètre a été retiré de la pile

setpage endp                                        ;Fin de la procédure

;---------------------------------------------------------------------------
;-- SHOWPAGE: Affiche l'une des deux pages d'écran
;-- Appel depuis TP: showpage( page : byte );

showpage proc near

        pop   bx                                    ;Dépile l'adresse de retour
        pop   ax                                            ; ainsi l'argument

        push  bx                          ;Remet l'adresse de retour sur la pile

        or    al,al                                         ;Est-ce la page 0?
        je    sp2     ;Oui, son numéro est égal à l'octet fort de l'offset

        mov   al,80h                              ;Non, page 1, avec offset 8000h

sp2:    mov   dx,CRTC_INDEX                     ;Adresse le contrôleur d'écran
        mov   ah,al                   ;Charge en AH l'octet fort de l'offset
        mov   al,CC_START_HI                        ;Numéro du registre en AL
        out   dx,ax                                         ;Effectue la sortie

              ;-- Attend un début de rafraîchissement d'écran -----

        mov   dx,VERT_RETRACE           ;Attend la fin du retour de balayage
sp3:    in    al,dx                                                 ;vertical
        test  al,8
        jne   sp3

sp4:    in    al,dx                     ;puis le début du retour du faisceau
        test  al,8
        je    sp4

        ret                           ;Retourne à l'appelant, l'argument a déjà
                                                        ;été retiré de la pile
showpage endp                                       ;Fin de la procédure

;--- Fin ---------------------------------------------------------------------

CODE    ends                                        ;Fin du segment de code
        end                                                 ;Fin du programme
```

4.8.7. Libre sélection de couleurs

La principale différence marquant la génération des cartes EGA et VGA par rapport à leurs prédécesseurs est leur faculté à afficher plus de couleurs que les 16 habituelles. Si le programmeur dispose de 64 couleurs différentes avec une carte EGA, l'éventail de couleurs des cartes VGA couvre plus de 262000 couleurs. Mais toutes les couleurs disponibles ne peuvent pas être affichées simultanément sur l'écran : selon qu'il s'agit d'un mode texte ou graphique, vous pouvez reproduire 16 ou 256 variétés de couleurs sur l'écran.

Ce chapitre montre comment sélectionner les couleurs et les définir à l'aide du BIOS. Les thèmes suivants sont abordés :

✓ Fonction et programmation des registres de palette

- ✓ Fonction et programmation de la table de couleurs DAC
- ✓ Affichage de couleurs en mode texte
- ✓ Affichage de couleurs dans les modes graphiques 16 et 256 couleurs
- ✓ Définition de la couleur du cadre de l'écran
- ✓ Transformation des niveaux de gris avec une carte VGA

Fonction des registres de palette

Les 16 registres de palette, faisant partie du contrôleur d'attributs, sont déterminants pour l'affichage des couleurs dans tous les modes texte et graphiques avec 16 couleurs ou moins des cartes EGA et VGA. Si le contrôleur CRT rencontre la couleur de premier plan ou de fond d'un caractère de texte ou la couleur d'un point graphique lors de la configuration de l'écran, il utilise alors cette valeur entre 0 et 15 comme index dans la table des registres de palette. A partir du registre de palette ainsi indexé, il extrait la couleur souhaitée qu'il transmet directement au moniteur à travers les diverses lignes du câble dans le cas d'une carte EGA.

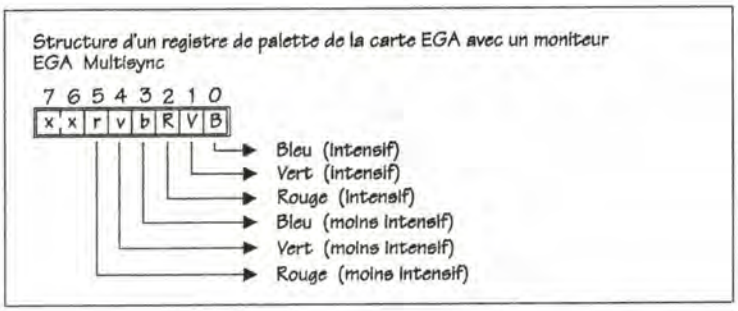
Cette manière de procéder dans les cartes EGA et VGA tend à briser le lien étroit existant entre les codes couleur et les couleurs affichées sur l'écran qui était la caractéristique de toutes les anciennes cartes vidéo. D'une part, cela permet au programmeur de sélectionner les couleurs qui lui conviennent. D'autre part, il peut effectuer des modifications d'ordre général sans être obligé de changer le contenu de la RAM vidéo. Par exemple, si tous les points du mode graphique 640*350 points qui étaient jusqu'à présent noirs doivent apparaître subitement avec une autre couleur, il suffit de changer le contenu du registre de palette 0.

Tant qu'on laisse intacts les registres de palette, on ne risque d'apercevoir aucune différence entre les cartes EGA/VGA et les cartes CGA en ce qui concerne l'affectation des couleurs. Cela s'explique tout simplement par le fait que les registres de palette sont chargés avec des valeurs correspondant aux couleurs de la carte CGA lors de l'initialisation d'un mode vidéo à travers la fonction 00h de l'interruption vidéo du BIOS. On peut interdire cette action à l'aide d'une sous-fonction de la fonction 12h pour que les registres de palette (ainsi que le registre DAC d'une carte VGA) restent inchangés lors de l'initialisation d'un mode vidéo à travers la fonction 00h.

Pour l'appel de cette fonction, il faut charger le numéro de fonction 12h dans le registre AH et le numéro de sous-fonction 31h dans le registre BL. Le registre AL recevant normalement le numéro de sous-fonction décide du verrouillage de l'initialisation. S'il est chargé avec la valeur 1 avant l'appel de la fonction, l'initialisation des registres de palette reste interdite lors de tous les appels ultérieurs de la fonction 00h. En revanche, on peut remettre en route ce mécanisme en chargeant la valeur 0 dans le registre AL.

Si l'initialisation automatique des registres de palette est activée, les techniques de couleur étendues des cartes EGA et VGA peuvent être pleinement exploitées une fois les registres de palette programmés. Mais à cet effet, il faut respecter attentivement la

structure des divers registres de palette. En considérant que la carte EGA est liée à un moniteur EGA ou Multisync, les différents bits d'un registre de palette correspondent directement avec la signification des lignes du moniteur servant à coder les couleurs. Pour les trois couleurs primaires rouge, vert et bleu (RVB), il existe respectivement deux lignes. L'une symbolise un affichage avec une couleur intense, l'autre avec une couleur pastel. Six bits sont en tout impliqués dans le codage des couleurs créant un maximum de 64 ($2^6 = 64$) couleurs dont seulement 16 sont visibles simultanément à travers les 16 registres de palette.



Le registre Color Plane Enable du contrôleur d'attributs joue également un rôle prépondérant dans l'affectation des couleurs. Avant chaque accès à l'un des 16 registres de palette, le contrôleur vidéo des cartes EGA et VGA relie l'index de couleur et les quatre bits inférieurs de ce registre par un ET logique. Ce mécanisme reste généralement transparent parce que le registre Color Plane Enable contient la valeur 1111b dans les quatre bits inférieurs en tant que valeur par défaut. Par conséquent, la combinaison ET avec l'index de couleur ne change pas la valeur si bien que l'écran affiche la couleur souhaitée comme à l'accoutumée.

Le résultat est différent lorsqu'on change la valeur du registre Color Plane Enable et on règle sur 0 certains bits qui y sont contenus. La valeur de l'index de couleur étant modifiée, on n'obtient plus la même structure entre l'index de couleur et les divers registres de palette. Si la valeur du registre Color Plane Enable vaut par exemple 0111b au lieu de 1111b, le bit de poids fort masque alors l'index de couleur et affecte les couleurs tirées des registres de palette 0 à 7 à tous les points écran (ou caractères) dotés d'un code couleur compris entre 8 et 15. Cette technique est rarement utilisée en pratique et la valeur par défaut dans le registre Color Plane Enable reste inchangée pendant toute la durée de l'exécution du programme.

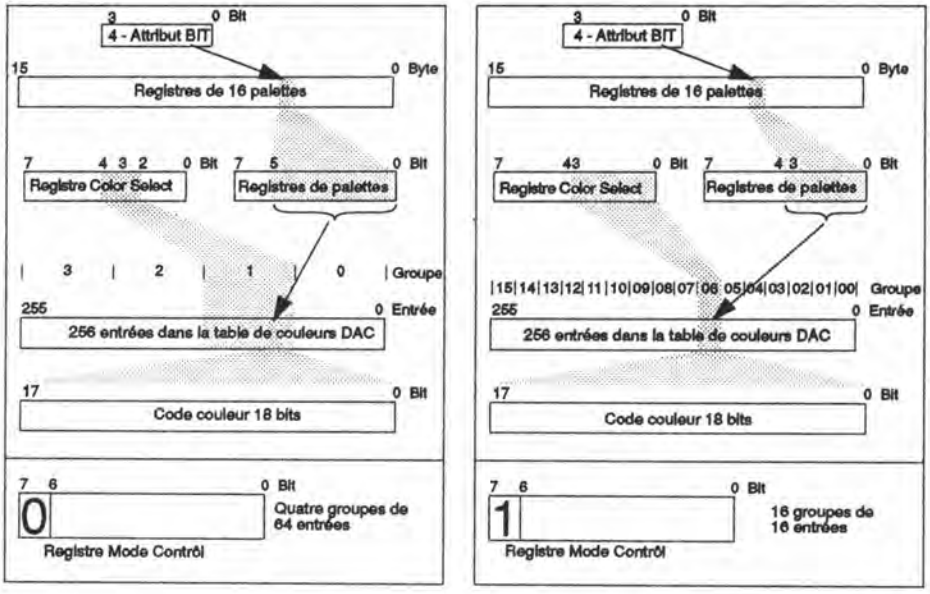
Fonction de la table de couleurs DAC

La carte VGA ne permet pas de transmettre au moniteur le contenu des registres de palette à travers les six lignes de couleur du câble du moniteur comme c'est le cas avec une carte EGA. En fait le standard VGA se distingue du standard EGA par l'utilisation des signaux analogiques qui ne peuvent pas être créés de cette manière.

Cela explique pourquoi il faut activer le table de couleurs DAC composée de 256 registres entre les 16 registres de palette, existant également sur la carte VGA, et le moniteur. La désignation DAC (Digital to Analog Converter en anglais) indique la fonction réalisée par cette table, c'est-à-dire convertir les codes couleurs digitaux en signaux analogiques.

Chaque registre de cette table de couleurs DAC représente une couleur parmi les 256 à sélectionner dans une palette de plus de 262 000 couleurs. Ce nombre astronomique résulte du codage de couleurs sous forme de 18 bits (2^{18} = 262 144), chaque registre de la table DAC étant composé de trois entrées de 6 bits chacune correspondant à la proportion de rouge, vert et bleu de la couleur concernée.

Un registre de la table DAC est sélectionné lorsque le contrôleur vidéo interprète le contenu des divers registres de palette d'une carte VGA comme un index et non comme un code couleur. Comme le montre la figure suivante, toutes sortes de registres interviennent dans ce contexte pour répartir les registres DAC en groupes distincts.



Construction des codes couleur sur une carte VGA

Le registre Mode Control du contrôleur vidéo assure une tâche importante. S'il contient la valeur 0, l'index de la table DAC est construit avec les bits 0 à 5 du registre de palette correspondant et les bits 2 et 3 du registre Color Select. La conséquence pratique de cette manipulation est de provoquer le sectionnement de la table DAC en 4 groupes de 64 registres consécutifs chacun. La valeur du registre de palette représente l'index de

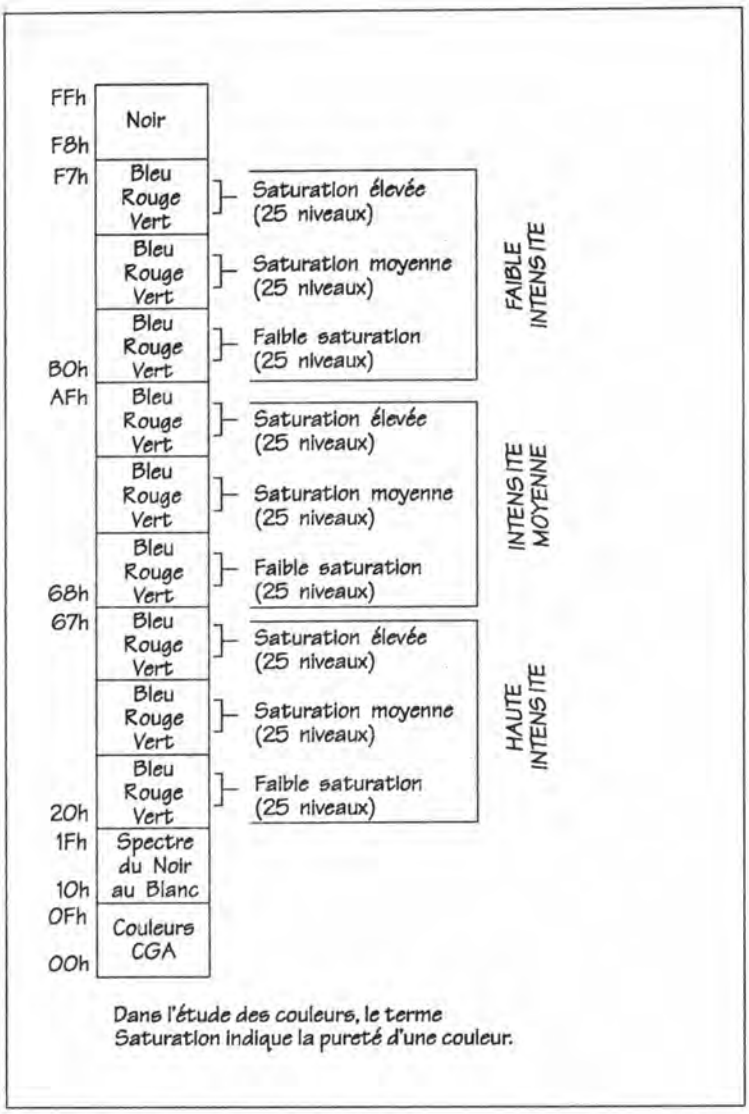
ce groupe où le groupe actif est sélectionné en fonction du contenu des bits 2 et 3 du registre Color Select.

Le résultat est différent lorsque le bit 7 du registre Mode Control contient la valeur 1. Dans ce cas, la table DAC est subdivisée en 16 groupes de 16 registres consécutifs. L'index de cette table est construit avec les bits 0 à 3 du registre de palette correspondant et les bits 0 à 3 du registre Color Select. Ce registre sélectionne par conséquent le groupe de couleurs actif à l'intérieur de la table DAC et le contenu du registre de palette représente l'index de ce groupe.

On peut par exemple utiliser ce type de codage pour créer des couleurs défilant rapidement et en permanence lorsqu'il s'agit de mettre en valeur sur l'écran un groupe de caractères et non un caractère unique grâce à une sorte de clignotement. Pour obtenir un tel effet, il suffit de stocker des séries de couleurs identiques avec une intensité croissante ou décroissante à l'intérieur des groupes de la table DAC et commuter ensuite rapidement vers le groupe de couleurs actif à travers le registre Color Select.

Eu égard au principe de la carte EGA lors de l'émulation de couleurs de la carte CGA, les 16 registres de palette de la carte VGA sont initialisés de telle manière qu'ils renvoient vers les 16 premiers registres de la table DAC. A leur tour, ces derniers sont définis de telle manière qu'ils reproduisent les couleurs CGA normales depuis le noir (0) jusqu'au blanc (15). Les autres registres de la table DAC ne sont pas définis à travers la fonction 00h du BIOS vidéo lors de l'initialisation d'un mode texte. En revanche, lors de l'initialisation d'un mode graphique, les 256 registres de la table DAC sont initialisés dans la mesure où cette initialisation n'a pas été désactivée par la sous-fonction 31h de la fonction 12h.

La figure suivante montre le schéma selon lequel les registres DAC sont initialisés. Pour que vous puissiez mettre cette manipulation en pratique, nous vous proposons un petit programme à la fin de cette section. Il montre comment installer les registres DAC sur l'écran et permet de modifier individuellement les différents registres.



Configuration des 256 registres de la table de couleurs DAC dans les modes graphiques de la carte VGA

Couleurs des modes graphiques 256 couleurs de la carte VGA

Dans les divers modes texte et modes graphiques 16 couleurs, les registres de palette constituent le point de départ pour l'affectation des couleurs. Mais ils ne jouent aucun rôle dans les modes graphiques 256 couleurs parce qu'on aurait besoin de 256 registres de palette différents qui ne sont pourtant pas disponibles. On évite donc de passer par

les registres de palette et on interprète directement le code couleur des divers points graphiques comme un index dans la table DAC.

En mode graphique 256 couleurs, les 256 entrées de cette table réflètent inévitablement les 256 couleurs affichables simultanément sur l'écran dans ce mode. Dans ce mode, la programmation des registres de palette ne provoque aucune réaction.

Définition de couleurs à travers le BIOS

Le BIOS EGA/VGA étendu contient de nombreuses fonctions permettant de définir le contenu des registres de palette et registres DAC ainsi que celui des autres registres du contrôleur d'attributs. Ces services sont implantés sous forme de sous-fonctions de la fonction 10h de l'interruption vidéo du BIOS. Pour les appeler, il faut que le numéro de fonction 10h soit toujours disponible dans le registre AH et le numéro de sous-fonction dans le registre AL.

La première sous-fonction 00h permet de charger l'un des 16 registres de palette avec une valeur de votre choix. En dehors du numéro de fonction dans le registre AH et du numéro de sous-fonction dans le registre AL, cette fonction attend le numéro du registre de palette à adresser (0 à 15) dans le registre BH et la nouveau code couleur de ce registre dans le registre BL.

Cette sous-fonction ne contrôlant pas le numéro de registre transmis, elle permet de lire un registre de palette ainsi que le registre Overscan qui s'enchaîne directement au dernier registre de palette. Comme ce registre reproduit la couleur du cadre de l'écran et la couleur du fond dans les modes graphiques compatibles CGA, il existe une sous-fonction destinée à cet usage. Elle porte le numéro 01h.

Mais il n'y a aucun intérêt à définir une autre couleur de fond que le noir dans les modes texte de la carte EGA. Ici, l'affichage du texte s'effectue sur la totalité de l'écran et il ne reste plus que deux ou trois lignes de grille pour la sortie d'une couleur de cadre. En cas de connexion à un moniteur monochrome, le contenu du registre Overscan ne joue par ailleurs aucun rôle.

Pour appeler la fonction permettant l'accès à un registre Overscan, il faut d'abord charger le numéro de fonction 10h dans le registre AH et le numéro de sous-fonction 01h dans le registre AL. Le registre BH reçoit la couleur de cadre chargée dans le registre Overscan lors de l'appel de la fonction.

Si les divers registres de palette y compris le registre Overscan doivent être chargés en une seule passe, la sous-fonction 02h offre des services intéressants. En dehors des numéros de fonction et sous-fonction, il faut lui transmettre l'adresse d'une table dans la paire de registres ES:DX. Cette table contient les valeurs pour les 17 registres (16 registres de palette plus le registre Overscan qui suit) sous la forme de 17 octets. Dès

que cette fonction est exécutée, le contenu de cette table est entièrement copié dans les 17 registres, ce qui fait immédiatement transposer toutes les couleurs.

Alors qu'il existe deux fonctions pour agir sur le contenu des registres de palette, le BIOS EGA étendu ne possède aucune fonction pour lire ces registres. Cela n'est d'ailleurs pas possible avec la carte EGA parce que pratiquement tous les registres de la carte EGA et non seulement ceux du contrôleur d'attributs sacrifient leur contenu après la description. Cela est particulièrement gênant pour les programmes TSR qui remettent les registres de palette sur leur valeur par défaut lors de l'appel. Mais lorsque le programme interrompu auparavant est réactivé, ils ne sont plus en mesure de placer le contenu des registres de palette sur leur état initial.

De tels programmes font alors dévier les sous-fonctions 00h et 02h de la fonction 10h sur une routine qui lit d'abord les codes couleurs spécifiés avant de les écrire dans les registres de palette. Ce processus ne fonctionne pas lorsqu'un programme accède directement aux registres de palette en évitant ainsi les fonctions BIOS destinées à cet usage. Nous vous conseillons donc d'avoir toujours recours à ces fonctions même si vous pouvez manipuler directement ces registres comme nous l'avons expliqué à la section 4.8.10.

La dernière sous-fonction de la fonction 10h du BIOS EGA étendu définit la signification du bit 7 dans l'octet d'attribut du caractère à l'intérieur des modes texte. Lorsqu'il est réglé, ce bit affiche les caractères avec une couleur de fond intense ou les fait clignoter en présence d'une carte EGA/VGA ou CGA/MDA. Avec des cartes CGA/MDA, la signification de ce bit ne peut être influencée que par la programmation directe de l'électronique vidéo. En revanche, le BIOS EGA/VGA propose spécialement la sous-fonction 03h de la fonction 10h pour réaliser cette manipulation.

En dehors des numéros de fonction et sous-fonction dans les registres AH et AL, le contenu du registre BL est particulièrement utile pour l'appel de cette sous-fonction. La valeur 0 instaure la couleur de fond intense et la valeur 1 fait clignoter tous les caractères sur l'écran dont le bit 7 est réglé dans l'octet d'attribut.

Autres sous-fonctions du BIOS VGA étendu

La fonction 10h de l'interruption vidéo du BIOS a été augmentée dans le BIOS VGA par rapport au BIOS EGA. Il contient de nouvelles fonctions liées au contexte de la table de couleurs DAC et la lecture des registres de palette. Avec la carte VGA, il devient plus commode de lire d'autres registres VGA qui étaient dotés de l'attribut "read only" dans le standard EGA.

Le contenu des différents registres DAC peut être modifié à l'aide de la sous-fonction 10h. En dehors des numéros de fonction et sous-fonction, la fonction attend le numéro du registre DAC réclamé (0 à 255) dans le registre BX et le code couleur à charger dans les registres CH, CL et DH. Ils reçoivent respectivement six bits parmi les 18 qui

constituent le code couleur. Comme les autres sous-fonctions décrites ci-après, la sous-fonction 10h attend la proportion de rouge dans le registre DH, la proportion de vert en CH et la proportion de bleu en DL où seuls les bits 0 à 5 sont à prendre en compte.

Les registres DH, DL et CL servent à la sous-fonction 15h pour retourner le contenu d'un registre DAC. Avant l'appel de la fonction, il suffit de charger le registre BX avec le numéro du registre DAC et les registres AH et AL avec les numéros de fonction et sous-fonction.

La sous-fonction 12h permet de charger simultanément plusieurs registres DAC. Précisez tout simplement le numéro du premier registre DAC à charger en BX et le nombre de registres à charger en CX. Le nouveau contenu des registres DAC est transmis dans un buffer (adresse en ES:DX) et non dans les registres du processeur. A l'intérieur de ce buffer, chaque registre DAC occupe trois octets consécutifs où le premier contient la proportion de vert, le second celle de rouge et le troisième celle de bleu du code couleur.

La sous-fonction 17h est l'inverse de la sous-fonction 12h. Elle lit le contenu d'une série de registres DAC. Ici aussi, entrez le numéro du premier registre DAC à adresser en BX et le nombre de registres en CX. Le BIOS VGA copie le contenu de ces registres dans un buffer dont les adresses de segment et d'offset sont attendues dans la paire de registres ES:DX. La structure du buffer est identique à celle rencontrée dans la sous-fonction 12h. Notez toutefois que chaque registre de la table DAC a besoin de trois octets et non d'un seul et pensez à prévoir un buffer suffisamment grand.

Vous pouvez utiliser la sous-fonction 13h pour sélectionner le sectionnement et le groupe de couleurs actif. Celle-ci dispose de deux sous-fonctions (à vrai dire sous-sous-fonctions). Si vous lui transmettez la valeur 0 dans le registre BL, elle copie le bit 0 du registre BH dans le bit 7 du registre Mode Control du contrôleur VGA et subdivise la table DAC en 4 ou 16 groupes. Si le registre BL contient la valeur 1 lors de l'appel de cette sous-fonction, elle copie alors le contenu du registre BH dans le registre Color Select et sélectionne ainsi le groupe de couleurs actif.

Vous pouvez consulter le contenu des deux registres en appelant la sous-fonction 1Ah. Après appel de cette sous-fonction, le registre BL reçoit le contenu du bit 7 dans le registre Mode Control et le registre BH obtient le contenu du registre Color Select.

La sous-fonction 1Bh sert à convertir les codes couleur de la table DAC en niveaux de gris lorsqu'il faut afficher une image grisée sur un moniteur couleur VGA. Une telle manipulation reste toutefois sans effet sur un moniteur monochrome VGA puisque la conversion des couleurs en niveaux de gris s'effectue automatiquement dans le moniteur.

La sous-fonction 1Bh attend le numéro du premier registre à convertir en BX et le nombre de registres à traiter en CX. La conversion, désignée par "gray scale summing", résulte de la prise en compte des différents éléments entrant dans la composition des couleurs pour obtenir un code couleur compris entre 0 (noir) et 1 (blanc). Selon l'intensité

de l'affichage sur l'écran, la proportion de rouge correspond à 30 % du niveau de gris, celle du vert à 59 % et celle du bleu à 11 %.

Outre la conversion sélective des différents registres DAC en niveaux de gris, vous pouvez également définir une conversion générale qui s'exécute automatiquement à chaque accès à la table DAC à travers le BIOS. Utilisez à cet effet la sous-fonction 33h de la fonction 12h. Exceptionnellement, le numéro de sous-fonction doit être chargé dans le registre BL et non AL avant l'appel de l'interruption vidéo du BIOS. Le numéro de fonction est à placer comme d'habitude dans le registre AH. Ici, le registre AL reçoit le véritable argument de fonction au lieu du numéro de sous-fonction. Cet argument décide si la conversion du registre DAC doit s'effectuer automatiquement ou non. La valeur 0 commande au BIOS VGA de produire un éclairage vert alors que la valeur 1 interdit ce mécanisme.

Sous la fonction 10h, le BIOS VGA regroupe d'autres sous-fonctions en plus des fonctions d'accès à la table DAC. Ces fonctions permettent un accès en lecture aux registres de palette. Ainsi, la sous-fonction 07h sert à lire le contenu d'un registre de palette quelconque. En dehors du numéro de fonction, il faut lui transmettre le numéro du registre de palette à adresser en BL pour qu'elle retourne son contenu en BH. Cette méthode permet également d'obtenir le contenu du registre Overscan (couleur du cadre dans le registre de palette 16), mais il existe une sous-fonction spécialement conçue à cet effet et c'est la sous-fonction 08h. Comme la sous-fonction 07h, elle retourne son résultat dans le registre BH.

La sous-fonction 09h permet d'obtenir la copie intégrale de la table de palette, c'est-à-dire les 16 registres de palette y compris le registre Overscan. Elle écrit le contenu de cette table dans un buffer de 17 octets dont l'adresse de segment doit lui être transmis dans le registre ES et l'adresse d'offset dans le registre DS.

Programmes d'exemple

La sélection des couleurs est une tâche plus ardue à gérer que la définition qui se réalise assez facilement grâce aux fonctions BIOS prédéfinies. Il est vrai qu'avec les 64 couleurs de la carte EGA on arrive mieux à se retrouver qu'avec la gamme étendue des couleurs de la carte VGA. Nous vous proposons donc un programme d'exemple à la fin de ce chapitre. Il porte le nom VDAC et existe dans une version Pascal sous le nom VDACP.PAS et une version C sous le nom VDACC.C.

Les deux programmes fonctionnent dans le mode graphique 256 couleurs de la carte VGA avec une résolution de 320*400 points. Ils collaborent avec les modules Assembleur V3240PA.ASM et V3240CA.ASM décrits en 4.8.7. Vous rencontrez en outre de nombreuses routines telles que ISVGA, LINE et GRAFPRINT qui vous ont déjà été présentées dans les sections précédentes.

Les routines SETDAC, GETDAC et DEMO sont les principales pièces des deux programmes. SETDAC et GETDAC servent d'interfaces aux sous-fonctions 12h et 17h de la fonction 10h. Elles permettent de lire et écrire un nombre quelconque de registres DAC. DEMO s'avère particulièrement utile car elle charge non seulement le contenu de tous les registres DAC dans la mémoire mais vous permet de consulter et modifier le contenu des différents registres.

Au centre de l'écran, vous apercevez 256 blocs de couleurs disposés dans un carré. Chaque bloc se compose de points dotés d'une couleur bien précise, le premier se trouvant au coin supérieur gauche avec le code couleur 0. Le bloc situé immédiatement à côté porte le code couleur 1, son voisin le code couleur 2 et ainsi de suite jusqu'au dernier bloc occupant le coin inférieur droit dont les points correspondent au code couleur 255.

Ce mode d'organisation provoque l'affichage des 256 couleurs sur l'écran définies automatiquement par le BIOS dans les divers registres DAC lors de l'initialisation d'un mode graphique. Pour ne pas vous donner uniquement un aperçu optique de la couleur, vous pouvez lire la correspondance numérique de la couleur concernée dans la ligne d'état au bas de l'écran sous forme de proportions de rouge, vert et bleu. Ces indications concernent d'abord les couleurs affichées en haut à gauche mais vous pouvez utilisez les touches du curseur pour afficher d'autres couleurs.

Vous constatez que le champ de couleur en cours est mis en évidence à l'aide d'un cadre blanc qui agit comme une sorte de curseur. Vous apercevez également le message de copyright en haut de l'écran dans la couleur du champ de couleur en cours. Vous ne pouvez pas voir ce message au début parce que vous vous trouvez dans un champ noir.

Pour que le message de copyright soit toujours adapté à la couleur en cours, il est doté du code couleur 255 qui correspond au noir. A chaque déplacement du curseur, la couleur du champ en cours est copiée pour la couleur 255 dans le registre DAC, ce qui fait apparaître le message de copyright dans cette couleur ainsi que le champ dans le coin inférieur droit du bloc de couleurs.

Le programme permet de traiter séparément les couleurs et modifier également la composition du champ en cours c'est-à-dire changer les proportions de rouge, vert et bleu. Les touches R, V et B assurent cette fonction. Utilisées en minuscules, sans appuyer sur la touche Shift, elles augmentent la proportion de la couleur concernée ce qui se remarque immédiatement dans le champ de couleur, le message de copyright et le contenu de la ligne d'état. L'appui simultané de l'une des touches R, V et B avec Shift provoque la diminution des différentes composantes. L'appui sur la barre d'espace remet un champ de couleur sur son état initial.

L'appui sur Return met fin au programme, réinstalle la table de couleurs initiale et réactive le mode texte.

Listing : VDACC.C

```
/*****************************************************************
*                          V D A C C . C                         *
******************************************************************
* Fonction        : Montre comment programmer les registres DAC  *
*                   dans les 256 couleurs du mode graphique de la*
*                   carte VGA. Le programme utilise les routines en*
*                   assembleur du module V3240CA.ASM             *
******************************************************************
* Auteur          : MICHAEL TISCHER                              *
* Développé le    :  2.01.1991                                   *
* Dernière MAJ    : 14.02.1992                                   *
******************************************************************
* Modèle mémoire  : SMALL                                        *
******************************************************************
* (MICROSOFT C)                                                  *
* Compilation     : CL /AS vdacc.c v3240ca                       *
******************************************************************
* (BORLAND TURBO C)                                              *
* Compilation     : Utilise un projet avec le contenu suivant    *
*                     vdacc.c                                    *
*                     v3240ca.obj                                *
******************************************************************
* Appel           : vdacc                                        *
******************************************************************
* Info            : Le message "Structure passed by value ..."   *
*                   est normal et n'indique pas une erreur       *
*****************************************************************/

#include <dos.h>
#include <stdarg.h>
#include <stdlib.h>
#include <stdio.h>
#include <conio.h>

/*-- Déclarations de types ------------------------------------*/

typedef unsigned char BYTE;                 /* Bricolade d'un type BYTE */

typedef union {                             /* Décrit un registre DAC */
              struct { BYTE Rouge, Vert, Bleu; } b;
              BYTE RGB[3];
             } DACREG;
typedef DACREG DACARRAY[256];               /* Table DAC complète */

/*-- Références externes aux routines en assembleur-----------*/

extern void init320400( void );
extern void setpix( int x, int y, unsigned char couleur);
extern BYTE getpix( int x, int y );
extern void setpage( BYTE page );
extern void showpage( BYTE page );
extern void far * getfontptr( void );

/*-- Constantes -----------------------------------------------*/

#define MAXX      319                       /* Coordonnées maximales */
#define MAXY      399

#define LARGEUR   10        /* Largeur d'un bloc de couleur en pixels */
#define HAUTEUR   20        /* Hauteur d'un bloc de couleur en pixels */
#define DISTANCE  2                       /* Distance entre les blocs */
#define LARGEURT  (16 * LARGEUR + ( 15 * DISTANCE ))    /* Larg. tot. */
#define HAUTEURT  (16 * HAUTEUR + ( 15 * DISTANCE ))    /* Haut tot. */
#define STARTX    (MAXX - LARGEURT ) / 2      /* Coin bloc sup gauche */
#define STARTY    (MAXY - HAUTEURT ) / 2


/*****************************************************************
* IsVga: Teste la présence d'une carte VGA.                      *
******************************************************************
* Entrée : néant                                                 *
* Sortie : 0  si pas de carte VGA, sinon -1                      *
*****************************************************************/

BYTE IsVga( void )
{
 union REGS Regs;                   /* Registres pour gérer l'interruption */

 Regs.x.ax = 0x1a00;                /* La fonction 1AH n'existe qu'en VGA */
 int86( 0x10, &Regs, &Regs );
 return ( Regs.h.al == 0x1a );                  /* Est-elle disponible ? */
}
/*****************************************************************
* PrintChar : Ecrit un caractère en dehors de la zone visible    *
*             de la mémoire d'écran                              *
******************************************************************
* Entrée  :   caractere = caractère à afficher                   *
*             x, y      = Coordonnées du coin supérieur gauche   *
*             cc        = Couleur du caractère                   *
*             cf        = Couleur du fond                        *
* Info    : Le caractère est dessiné dans une matrice de 8*8 pixels - *
*           sur la base du jeu de caractères 8*8 en ROM          *
*****************************************************************/

void PrintChar( char caractere, int x, int y, BYTE cc, BYTE cf )
{
 typedef BYTE   CARDEF[256][8];     /* Structure du jeu de caractères */
 typedef CARDEF far *CARPTR;        /* Pointe sur un jeu de caractères */

 BYTE    i, k,                                  /* Compteur d'itérations */
         masque;        /* Masque binaire pour dessiner le caractère */

 static  CARPTR fptr = (CARPTR) 0;      /* Jeu de caractères en ROM */

 if( fptr == (CARPTR) 0 )       /* A-t-on déjà déterminé ce pointeur ? */
   fptr = getfontptr();         /* Non, détermine par fonct. assembleur */

 /*- Dessine le caractère pixel par pixel ---------------------*/

 if( cf == 255 )                                /* Caractère transparent ? */
   for( i = 0; i < 8; ++i )       /* Oui, dessine que pixels 1er plan */
   {
     masque = (*fptr)[caractere][i];          /* Motif bin. pour ligne */
     for( k = 0; k < 8; ++k, masque <<= 1 )  /* Parcourt les colonnes */
       if( masque & 128 )                      /* Pixel à dessiner ? */
         setpix( x+k, y+i, cc );                          /* Oui */
   }
 else                                       /* Non dessine chaque pixel */
   for( i = 0; i < 8; ++i )                   /* Parcourt les lignes */
   {
     masque = (*fptr)[caractere][i];          /* Motif bin. pour ligne */
     for( k = 0; k < 8; ++k, masque <<= 1 ) /* Parcourt les colonnes */
       setpix( x+k, y+i, (BYTE) (( masque & 128 ) ? cc : cf) );
   }
}

/*****************************************************************
* Line: Trace un segment dans la fenêtre graphique en appliquant *
*       l'algorithme de Bresenham                                *
******************************************************************
* Entrées : X1, Y1 = Coordonnées de l'origine                    *
*           X2, Y2 = Coordonnées de l'extrémité terminale        *
*           COULEUR = couleur du segment                         *
*****************************************************************/

/*-- Fonction accessoire pour échanger deux variables entières -------*/

void SwapInt( int *i1, int *i2 )
{
 int dummy;

 dummy = *i2;  *i2  = *i1;  *i1  = dummy;
}

/*-- Procédure principale -------------------------------------*/

void Line( int x1, int y1, int x2, int y2, BYTE couleur )
{
 int d, dx, dy,
     aincr, bincr,
     xincr, yincr,
     x, y;

 if( abs(x2-x1) < abs(y2-y1) )     /* Sens du parcours : axe X ou Y ? */
 {                                                          /* Par Y */
   if( y1 > y2 )                                 /* y1 plus grand que y2? */
   {
     SwapInt( &x1, &x2 );                        /* Oui échange X1 et X2, */
     SwapInt( &y1, &y2 );                                  /* Y1 et Y2 */
   }

   xincr = ( x2 > x1 ) ?  1 : -1;               /* Fixe le pas horizontal */

   dy    = y2 - y1;
   dx    = abs( x2-x1 );
   d     = 2 * dx - dy;
   aincr = 2 * (dx - dy);
   bincr = 2 * dx;
   x     = x1;
   y     = y1;

   setpix( x, y, couleur );                  /* dessine le premier pixel */
   for( y=y1+1; y<= y2; ++y )                   /* Parcourt l'axe des Y */
   {
```

```
    if( d >= 0 )
    {
      x += xincr;
      d += aincr;
    }
    else
      d += bincr;
    setpix( x, y, couleur );
  }
 }
 else                                                    /* par X */
 {
  if( x1 > x2 )                              /* x1 plus grand que x2? */
  {
    SwapInt( &x1, &x2 );                     /* Oui, échange X1 et X2 */
    SwapInt( &y1, &y2 );                                /* Y1 et Y2 */
  }

  yincr = ( y2 > y1 ) ? 1 : -1;               /* Fixe le pas vertical */

  dx    = x2 - x1;
  dy    = abs( y2-y1 );
  d     = 2 * dy - dx;
  aincr = 2 * (dy - dx);
  bincr = 2 * dy;
  x     = x1;
  y     = y1;

  setpix( x, y, couleur );                  /* Dessine le premier pixel */
  for( x=x1+1; x<=x2; ++x )                    /* Parcourt l'axe des X */
  {
    if( d >= 0 )
    {
      y += yincr;
      d += aincr;
    }
    else
      d += bincr;
    setpix( x, y, couleur );
  }
 }
}

/*****************************************************************
*  GrafPrintf: Affiche une chaine formatée sur l'écran graphique *
*----------------------------------------------------------------*
*  Entrées : X, Y   = Coordonnées de départ (0 - ...)            *
*            CC     = Couleur des caractères                     *
*            CF     = Couleur du fond  (255 = transparent)       *
*            STRING = Chaine avec indications de formatage       *
*            ...    = Expressions comme pour printf              *
*****************************************************************/

void GrafPrintf( int x, int y, BYTE cc, BYTE cf, char * string, ... )
{
 va_list  parameter;      /* Liste de paramètres pour les macros VA_...*/
 char     affichage[255],           /* Buffer pour la chaine formatée */
          *cp;

 va_start( parameter, string );           /* Convertit les paramètres */
 vsprintf( affichage, string, parameter );                /* Formate */
 for( cp = affichage; *cp; ++cp, x += 8 )          /* Affiche la chaine */
   PrintChar( *cp, x, y, cc, cf );          /* formatée par PrintChar */
}

/*****************************************************************
*  GetDac: Détermine les contenus d'un certain nombre de registres DAC *
*----------------------------------------------------------------*
*  Entrées : FIRST = Numéro du premier registre (0-255)          *
*            NBR   = Nombre de registres DAC                     *
*            BUFP  = Pointeur sur le buffer qui doit recevoir    *
*                    les contenus des registres DAC. Doit être une *
*                    variable du type DACREG ou un tableau de variables*
*                    de ce type                                  *
*  Info    : Le buffer transmis doit comporter trois octets par registre*
*            DAC à lire (pour la composante rouge, verte et bleue *
*            de chaque couleur)                                  *
*****************************************************************/

void GetDac( int First, int Nbr, void far *BufP )
{
 union REGS    Regs;       /* Registres proc. pour gérer interruption */
 struct SREGS  SRegs;                          /* Registres de segment */

 Regs.x.ax = 0x1017;          /* Numéro de la fonction et de l'option */
 Regs.x.bx = First;                /* Numéro du premier registre DAC */
 Regs.x.cx = Nbr;                   /* Nombre de registres à charger */
 Regs.x.dx = FP_OFF( BufP );
 SRegs.es  = FP_SEG( BufP );                    /* Pointeur sur le buffer */
 int86x( 0x10, &Regs, &Regs, &SRegs );           /* Déclenche int. BIOS */
}

/*****************************************************************
*  SetDac: Charge un certain nombre de registres DAC             *
*----------------------------------------------------------------*
*  Entrées : FIRST = Numéro du premier registre DAC  (0-255)     *
*            NBR   = Nombre de registres DAC                     *
*            BUFP  = Pointeur sur le buffer d'où seront tirées les *
*                    valeurs à transférer dans les registres DAC.  *
*                    Il doit s'agir d'une variable de type DACREG ou *
*                    d'un tableau de variables de ce type.       *
*  Info    : cf GetDac()                                         *
*****************************************************************/

void SetDac( int First, int Nbr, void far *BufP )
{
 union REGS    Regs;                 /* Registres proc. pour gérer int. */
 struct SREGS  SRegs;                          /* Registres de segment */

 Regs.x.ax = 0x1012;            /* Numéro de la fonction et de l'option */
 Regs.x.bx = First;                /* Numéro du premier registre DAC */
 Regs.x.cx = Nbr;                   /* Nombre de registres à charger */
 Regs.x.dx = FP_OFF( BufP );
 SRegs.es  = FP_SEG( BufP );                    /* Pointeur sur le buffer*/
 int86x( 0x10, &Regs, &Regs, &SRegs );           /* Déclenche int. BIOS */
}

/*****************************************************************
*  PrintDac: Affiche le contenu d'un registre DAC et règle la couleur *
*            dans le registre DAC 255                            *
*----------------------------------------------------------------*
*  Entrées : DREG  = Registre DAC                                *
*            NUM   = Numéro de ce registre                       *
*            COULEUR = Couleur d'affichage                       *
*****************************************************************/

void PrintDac( DACREG DReg, BYTE Num, BYTE Couleur )
{
  SetDac( 255, 1, &DReg );          /* Couleur dans le registre DAC 255 */
  GrafPrintf( 60, MAXY-10, Couleur, 0,
              "DAC:%3d  R:%2d  V:%2d  B:%2d",
              Num, DReg.b.Rouge, DReg.b.Vert, DReg.b.Bleu);
}

/*****************************************************************
*  Encadre : Trace un cadre autour d'un champ de couleur         *
*----------------------------------------------------------------*
*  Entrées : X     = Abscisse X du champd de couleur (0-15)      *
*            Y     = ordonnée Y du champ de couleur (0-15)       *
*            COULEUR = Couleur du cadre                          *
*  Info    : L'épaisseur du cadre est de 1 pixel indépendamment  *
*            de la distance qui sépare les champs                *
*****************************************************************/

void Encadre( int x, int y, BYTE Couleur)
{
 int sx, sy,                                 /* Coin sup gauche du cadre */
     ex, ey;                                  /* Coin inf droit du cadre */

/*-- Calcule les coordonnées des coins du cadre ----------------------*/

 ex = ( sx = STARTX + x * (LARGEUR + DISTANCE) - 1 ) + LARGEUR + 1;
 ey = ( sy = STARTY + y * (HAUTEUR + DISTANCE) - 1 ) + HAUTEUR + 1;

 Line( sx, sy, ex, sy, Couleur );                      /* Trace le cadre */
 Line( ex, sy, ex, ey, Couleur );
 Line( ex, ey, sx, ey, Couleur );
 Line( sx, ey, sx, sy, Couleur );
}

/*****************************************************************
*  ChangeDacReg: Modifie le contenu d'un registre DAC en mémoire *
*                et dans la table DAC de la carte vidéo, puis    *
*                l'affiche sur l'écran                           *
*----------------------------------------------------------------*
*  Entrées : DREGP = Pointeur sur le registre DAC à modifier     *
*            NUM   = Numéro du registre DAC                      *
*            COMP  = Numéro de la composante à modifier (0-2)    *
*                    0 = Rouge, 1 = Vert, 2 = Bleu               *
*            INCR  = Incrément pour cette composante             *
*****************************************************************/

void ChangeDacReg( DACREG *DRegP, BYTE Num, BYTE Comp, BYTE Incr )
{
 if(( DRegP->RGB[ Comp ] += Incr) > 63 )  /* Incrémente la composante */
   DRegP->RGB[ Comp ] = 0;                          /* Sup à 53: ramène à 0 */
 SetDac( Num, 1, DRegP );                        /* Charge le registre DAC */
 PrintDac( *DRegP, Num, 15 );                 /* Affiche le nouveau contenu */
}

/*****************************************************************
*  Demo: Présente la programmation des registres DAC et le système *
*        de couleur de la carte VGA                              *
*----------------------------------------------------------------*
*  Entrées : néant                                               *
*****************************************************************/
```

```
/
void Demo( void )
{
 int      x, y,
          ix, jx,
          iy, jy,
          k, f;                          /* Compteurs d'itérations */
 char     ch;                                          /* Touche*/
 DACARRAY dacbuf;                        /* Table DAC complète */
 DACREG   DReg;                          /* Registre DAC courant */

 /*-- Dessine l'écran ----------------------------------------------*/

 setpage( 0 );                                  /* Traite la page 0 */
 showpage( 0 );                                 /* Affiche la page 0 */
 GetDac( 0, 256, dacbuf );            /* Charge la table DAC complète */

 GrafPrintf( 10, 0, 255, 0, "VDACC  -  (c) 1991 by MICHAEL TISCHER" );

 /*-- Construit le bloc de 16*16 couleurs --------------------------*/

 iy = STARTY;                           /* Point de départ sur l'écran */
 jy = STARTY + HAUTEUR - 1;
 f  = 0;
 for( y = 0; y < 16; ++y )          /* Parcourt les 16 lignes de blocs */
 {
   ix = STARTX;
   jx = STARTX + LARGEUR - 1;
   for( x = 0; x < 16; ++x )     /* Parcourt les 16 colonnes de blocs */
   {
     for ( k = iy; k <= jy; ++k )      /* Dessine blocs avec segments */
       Line( ix, k, jx, k, (BYTE) f );
     ix += LARGEUR + DISTANCE;                /* prochain bloc à droite */
     jx += LARGEUR + DISTANCE;
     ++f;                                        /* Couleur suivante */
   }
   iy += HAUTEUR + DISTANCE;                          /* Position suivante */
   jy += HAUTEUR + DISTANCE;
 }

/*-- Lit les entrées de l'utilisateur et réagit en conséquence -------*/

 ix = 0;                /* Commence en haut à gauche avec la couleur 0 */
 iy = 0;
 jx = 0;
 jy = 0;
 k  = 0;
 GetDac( 0, 1, &DReg );                                 /* Lit la couleur 0 */
 Encadre( 0, 0, 15 );                        /* Encadre le champ de couleur */
 PrintDac( DReg, 0, 15 );                          /* et affiche le contenu */
 do
 {
   ch = (char) getch();                                 /* Lit la touche frappée */
   if( ch )                                               /* touche étendue ? */
     switch( ch )                                         /* Non, on exploite */
     {
       case 'r' :
         ChangeDacReg( &DReg, (BYTE) k, 0, +1 );                 /* r=Rouge+ */
         break;
       case 'v' :
         ChangeDacReg( &DReg, (BYTE) k, 1, +1 );                 /* g=Vert+ */
         break;
       case 'b' :
         ChangeDacReg( &DReg, (BYTE) k, 2, +1 );                 /* b=Bleu+ */
         break;
       case 'R' :
         ChangeDacReg( &DReg, (BYTE) k, 0, -1 );                 /* R=Rouge- */
         break;
       case 'V' :
         ChangeDacReg( &DReg, (BYTE) k, 1, -1 );                 /* G=vert- */
         break;
       case 'B' :
         ChangeDacReg( &DReg, (BYTE) k, 2, -1 );                 /* B=Bleu- */
         break;
       case ' ' :
         {                                  /* Space = rétablit la valeur d'origine */
           DReg = dacbuf[ k ];
           ChangeDacReg( &DReg, (BYTE) k, 1, 0 );
           break;
         }
     }
   else                                                 /* Code de touche étendu */
     switch( getch() )
     {
       case 72 : if( iy == 0 )                          /* Curseur vers le haut */
                   jy = 15;
                 else
                   jy = iy - 1;
                 break;

       case 80 : if ( iy == 15 )                        /* Curseur vers le bas */
                   jy = 0;
                 else
                   jy = iy + 1;
                 break;

       case 75 : if( ix == 0  )                         /* Curseur à gauche */
                   jx = 15;
                 else
                   jx = ix - 1;
                 break;

       case 77 : if( ix == 15 )                         /* Curseur à droite */
                   jx = 0;
                 else
                   jx = ix + 1;
     }

   if( ix != jx  ||  iy != jy )          /* Nouvelle position du curseur ?*/
   {                                                              /* Oui */
     Encadre( ix, iy, 0 );                          /* Efface l'ancien cadre */
     Encadre( jx, jy, 15 );                         /* Trace le nouveau cadre */
     ix = jx;                      /* Mémorise le nouveau champ de couleur */
     iy = jy;
     k  = iy*16+ix;                /* Calcule le numéro du nouveau champ */
     GetDac( k, 1, &DReg );                         /* Charge le registre DAC */
     PrintDac( DReg, (BYTE) k, 15 );                          /* et l'affiche */
   }
 }
 while( ch != 13 );                  /* répète jusqu'à frappe de <Entrée> */

 SetDac( 0, 256, dacbuf );                          /* restaure la table DAC */
}

/*-------------------------------------------------------------------*/
/*--                        PROGRAMME PRINCIPAL                    --*/
/*-------------------------------------------------------------------*/

void main( void )
{
 union REGS  regs;

 if( IsVga() )                                    /* A-t-on une carte VGA ? */
 {                                                          /* Oui, c'est parti */
   init320400();                              /* Initialise le mode graphique */
   Demo();
   regs.x.ax = 0x0003;                              /* rétablit le mode texte */
   int86( 0x10, &regs, &regs );
 }
 else
   printf( "VDACC  -  (c) 1991, 92 by MICHAEL TISCHER\n\nATTENTION"\
           "Ce programme exige une carte VGA.\n\n" );
}
/
```

Listing : V3240CA.ASM (déjà présenté en 4.8.7)

Listing : VDACP.PAS

```
{*********************************************************************
*                          V D A C P . P A S                         *
**-------------------------------------------------------------------**
* Fonction        : Montre comment programmer les registres DAC      *
*                   dans les 256 couleurs du mode graphique de la    *
*                   carte VGA. Le programme utilise les routines     *
*                   en assembleur du module V3240PA.ASM              *
**-------------------------------------------------------------------**
* Auteur          : MICHAEL TISCHER                                  *
* Développé le    : 02.01.1991                                       *
* Dernière MAJ    : 14.01.1991                                       *
*********************************************************************}

program VDACP;

uses dos, crt;

{-- Déclarations de types ------------------------------------------}

type DACREG = record                              { Décrit un registre DAC }
                case integer of
                  0 : ( Rouge, Vert, Bleu : BYTE );    { Composants RGB }
                  1 : ( RGB : array[ 1..3] of BYTE );
              end;
     DACARRAY = array [0..255] of DACREG;          {Table DAC complète }

{-- Références externes aux routines en assembleur ----------------}

{$L v3240pa}        { Intègre le module en assembleur }

procedure init320400; external;
procedure setpix( x, y : integer; couleur : byte ); external;
function  getpix( x, y: integer ) : byte ; external;
procedure setpage( page : byte ); external;
procedure showpage( page : byte ); external;

{-- Constantes ----------------------------------------------------}

const MAXX = 319;                                 { Coordonnées maximales }
      MAXY = 399;

      LARGEUR  = 10;          { Largeur d'un bloc de couleur en pixels }
      HAUTEUR = 20;           { Hauteur d'un bloc de couleur en pixels }
      DISTANCE = 2;                          { Distance entre les blocs }
      LARGEURT = 16 * LARGEUR + ( 15 * DISTANCE );     { Largeur totale}
      HAUTEURT  = 16 * HAUTEUR + ( 15 * DISTANCE );  { Hauteur totale }
      STARTX   = ( MAXX - LARGEURT ) div 2;      { Coin bloc sup gauche }
      STARTY   = ( MAXY - HAUTEURT ) div 2;

{*********************************************************************
* IsVga   : Teste la présence d'une carte VGA.                       *
**-------------------------------------------------------------------**
* Entrée  : néant                                                    *
* Sortie  : TRUE ou FALSE                                            *
*********************************************************************}

function IsVga : boolean;

var Regs : Registers;                    { Registres pour l'interruption}

begin
  Regs.AX := $1a00;                { La fonction 1Ah n'existe qu'en VGA }
  Intr( $10, Regs );
  IsVga := ( Regs.AL = $1a );
end;

{*********************************************************************
* PrintChar : Affiche un caractère en mode graphique                 *
**-------------------------------------------------------------------**
* Entrée : caractere = le caractère à afficher                       *
*          x, y      = Coordonnées du coin sup gauche                *
*          cc        = Couleur du caractère                          *
*          cf        = Couleur du fond                               *
* Info   : Le caractère est dessiné dans une matrice de 8*8 pixels   *
*          sur la base du jeu de caractères 8*8 en ROM               *
*********************************************************************}

procedure PrintChar( caractere : char; x, y : integer; cc, cf : byte );

type CARADEF = array[0..255,0..7] of byte; {Structure jeu de caractères}
     CARAPTR = ^CARADEF;              { Pointe sur le jeu de caractères }

var  Regs  : Registers;       { Registres pour gérer les interruptions }
     ch    : char;                                  { Pixel du caractère }
     i, k,                                      { Compteur d'itérations }
     Masque : byte;          { Masque binaire pour dessiner le caractère}

const fptr : CARAPTR = NIL;   { Pointe sur le jeu de caractères en ROM }

begin
  if fptr = NIL then             { A-t-on déjà déterminé ce pointeur ? }
    begin                                                      { Non}
      Regs.AH := $11;                    { Appelle l'option $1130 de la }
      Regs.AL := $30;                          { fonction vidéo du BIOS }
      Regs.BH := 3;             { pour obtenir un pointeur sur le jeu 8*8 }
      Intr( $10, Regs );
      fptr := ptr( Regs.ES, Regs.BP );              { Compose le pointeur}
    end;

  if ( cf = 255 ) then                          { Caractère transparent ? }
    for i := 0 to 7 do  { Oui ne dessine que les pixels du premier plan}
      begin
        Masque := fptr^[ord(caractere),i];{Motif binaire pour une ligne}
        for k := 0 to 7 do
          begin
            if ( Masque and 128 <> 0 ) then        { Pixel à dessiner? }
              setpix( x+k, y+i, cc );                          { Oui }
            Masque := Masque shl 1;
          end;
      end
  else                                        { Non, tient compte du fond }
    for i := 0 to 7 do                          { Parcourt les lignes }
      begin
        Masque := fptr^[ord(caractere),i];{Motif binaire pour une ligne}
        for k := 0 to 7 do
          begin
            if ( Masque and 128 <> 0 ) then          { Premier plan ? }
              setpix( x+k, y+i, cc )                           { Oui }
            else
              setpix( x+k, y+i, cf );                      { Non, fond}
            Masque := Masque shl 1;
          end;
      end;
end;

{*********************************************************************
* Line : Trace un segment dans la fenêtre graphique en appliquant    *
*        l'algorithme de Bresenham                                   *
**-------------------------------------------------------------------**
*  Entrée : X1, Y1  = Coordonnées de l'origine (0 - ...)             *
*           X2, Y2  = Coordonnées de l'extrémité terminale           *
*           COULEUR = couleur du segment                             *
*********************************************************************}

procedure Line( x1, y1, x2, y2 : integer; couleur : byte );

var d, dx, dy,
    aincr, bincr,
    xincr, yincr,
    x, y                : integer;

{-- Procédure accessoire pour échanger deux variables entières --------}

procedure SwapInt( var i1, i2: integer );

var dummy : integer;

begin
  dummy := i2;
  i2    := i1;
  i1    := dummy;
end;
```

```
{
{-- Procédure principale ------------------------------------------}

begin
  if ( abs(x2-x1) < abs(y2-y1) ) then     (Parcours : par axe X ou Y ? )
    begin                                            ( par l'axe des Y )
      if ( y1 > y2 ) then                          ( y1 supérieur à y2 ? )
        begin
      SwapInt( x1, x2 ):                        ( Oui, échange X1 et X2, )
      SwapInt( y1, y2 ):                                ( Y1 et Y2 )
        end;

      if ( x2 > x1 ) then xincr := 1          ( Fixe le pas horizontal )
                     else xincr := -1;

      dy := y2 - y1;
      dx := abs( x2-x1 );
      d  := 2 * dx - dy;
      aincr := 2 * (dx - dy);
      bincr := 2 * dx;
      x := x1;
      y := y1;

      Setpix( x, y, couleur );                ( Dessine le premier point )
      for y:=y1+1 to y2 do                          ( Parcourt l'axe des Y )
        begin
          if ( d >= 0 ) then
            begin
              inc( x, xincr );
              inc( d, aincr );
            end
          else
            inc( d, bincr );
          Setpix( x, y, couleur );
        end;
    end
  else                                                    ( par l'axe des X )
    begin
      if ( x1 > x2 ) then                          ( x1 plus grand que x2 ? )
        begin
      SwapInt( x1, x2 );                              ( Oui, échange X1 et X2 )
      SwapInt( y1, y2 );                                    ( Y1 et Y2 )
        end;

      if ( y2 > y1 ) then yincr := 1          ( Fixe le pas vertical )
                     else yincr := -1;

      dx := x2 - x1;
      dy := abs( y2-y1 );
      d  := 2 * dy - dx;
      aincr := 2 * (dy - dx);
      bincr := 2 * dy;
      x := x1;
      y := y1;

      Setpix( x, y, couleur );                ( Dessine le premier point )
      for x:=x1+1 to x2 do                          ( Parcourt l'axe des X )
        begin
          if ( d >= 0 ) then
            begin
              inc( y, yincr );
              inc( d, aincr );
            end
          else
            inc( d, bincr );
          Setpix( x, y, couleur );
        end;
    end;
end;

{*****************************************************************
*  GrafPrint : Affiche une chaîne formatée sur l'écran graphique  *
**----------------------------------------------------------------**
*  Entrées: X, Y   = Coordonnées de départ (0-...)                *
*           CC     = Couleur des caractères                       *
*           CF     = Couleur du fond (255 = transparent)          *
*           STRING = Chaîne avec indications de formatage         *
*****************************************************************}

procedure GrafPrint( x, y : integer; cc, cf : byte; strt : string );

var i : integer;                                  ( Compteur d'itérations )

begin
  for i:=1 to length( strt ) do
    begin
      printchar( strt[i], x, y, cc, cf );       ( Affiche par printchar )
      inc( x, 8 );                    ( x à la position du caractère suivant )
    end;
end;


{*****************************************************************
*  GetDac: Détermine les contenus d'un certain nombre de registres DAC *
**----------------------------------------------------------------**
*  Entrées : FIRST = Numéro du premier registre (0-255)          *
*            NBR   = Nombre de registres DAC                     *
*            BUF   = buffer qui doit recevoir les contenus des   *
*                    registres DAC. Ce doit être une variable du type *
*                    DACREG ou un tableau de variables de ce type *
*  Info    : Le buffer transmis doit comporter trois octets par  *
*            registre DAC à lire (pour la composante rouge, verte et *
*            bleue de chaque couleur)                            *
*****************************************************************}

procedure GetDac( First, Nbr : integer; var Buf );

var Regs : Registers;                  ( Registres pour gérer l'interruption)

begin
  Regs.AX := $1017;                    ( Numéro de la fonction et de l'option )
  Regs.BX := First;                          ( Numéro du premier registre DAC)
  Regs.CX := Nbr;                          ( Nombre des registres à charger )
  Regs.ES := seg( Buf );                ( Charge un pointeur sur le buffer )
  Regs.DX := ofs( Buf );
  intr( $10, Regs );                        ( Déclenche l'interruption vidéo )
end;

{*****************************************************************
*  SetDac  : Charge un certain nombre de registres DAC           *
**----------------------------------------------------------------**
*  Entrées : FIRST = Numéro du premier registre DAC  (0-255)     *
*            NBR   = Nombre de registres                         *
*            BUF   = Buffer, d'où sont tirées les valeurs à transférer *
*                    dans les registres DAC. Il doit s'agir d'une *
*                    variable de type DACREG ou d'un tableau de  *
*                    variables de ce type                        *
*  Info    : cf GetDac                                           *
*****************************************************************}

procedure SetDac( First, Nbr : integer; var Buf );

var Regs : Registers;                  ( Registres pour gérer l'interruption )

begin
  Regs.AX := $1012;                    ( Numéro de la fonction et de l'option )
  Regs.BX := First;                          ( Numéro du premier registre DAC )
  Regs.CX := Nbr;                          ( Nombre de registres à charger )
  Regs.ES := seg( Buf );                ( Charge un pointeur sur le buffer )
  Regs.DX := ofs( Buf );
  intr( $10, Regs );                 ( Déclenche l'interruption vidéo du BIOS )
end;

{*****************************************************************
*  PrintDac : Affiche le contenu d'un registre DAC et règle la couleur *
*             dans le registre DAC 255                           *
**----------------------------------------------------------------**
*  Entrées : DREG    = Registre DAC                              *
*            NUM     = Numéro de ce registre                     *
*            COULEUR = Couleur d'affichage                       *
*****************************************************************}

procedure PrintDac( DReg : DACREG; Num, Couleur : BYTE );

var numstr,                                  ( Chaîne pour numéro de registre )
    rstr,                                    ( Chaîne pour composante rouge )
    vstr,                                    ( Chaîne pour composante verte )
    bstr : string[3];                        ( Chaîne pour composante bleue )

begin
  SetDac( 255, 1, DReg );                ( Couleur dans le registre DAC 255 )
  str( Num : 3, numstr );     ( Convertit en chaînes couleurs et numéro)
  str( DReg.Rouge : 2, rstr );
  str( DReg.Vert : 2, vstr );
  str( DReg.Bleu : 2, bstr );
  GrafPrint( 60, MAXY-10, Couleur, 0, 'DAC:' + numstr + '  R:' + rstr +
                                      '  V:' + vstr + '  B:' + bstr);
end;

{*****************************************************************
*  Encadre : Trace un cadre autour d'un champ de couleur         *
**----------------------------------------------------------------**
*  Entrées : X       = Abscisse X du champ de couleur (0-15)     *
*            Y       = Ordonnée Y du champ de couleur (0-15)     *
*            COULEUR = Couleur du cadre                          *
*  Info    : L'épaisseur du cadre est de 1 pixel indépendamment  *
*            de la distance qui sépare les champs.               *
*****************************************************************}

procedure Encadre( X, Y, Couleur : BYTE );

var sx, sy,                                  ( Coin sup gauche du cadre)
    ex, ey : integer;                        ( Coin inf droit du cadre )

begin
  sx := STARTX + X * (LARGEUR + DISTANCE) - 1;{calcule les coordonnées )
```

```
  ex := sx + LARGEUR + 1;                                  { du cadre }
  sy := STARTY + Y * (HAUTEUR + DISTANCE) - 1;
  ey := sy + HAUTEUR + 1;
  Line( sx, sy, ex, sy, Couleur );                  { Dessine le cadre }
  Line( ex, sy, ex, ey, Couleur );
  Line( ex, ey, sx, ey, Couleur );
  Line( sx, ey, sx, sy, Couleur );
end;

{*****************************************************************
*  ChangeDacReg : Modifie le contenu d'un registre DAC en mémoire et *
*                 dans la table DAC de la carte vidéo, puis l'affiche *
*                 sur l'écran                                       *
*-----------------------------------------------------------------*
*  Entrées : DREG  = Registre DAC à modifier                        *
*            NUM   = Numéro du registre DAC                         *
*            COMP  = Numéro de la composante à modifier    (1-3)    *
*                    1 = Rouge, 2 = Vert, 3 = Bleu                  *
*            INCR  = Incrément pour cette composante                *
*****************************************************************}

procedure ChangeDacReg( var DReg : DACREG; Num, Comp : BYTE;
                            Incr : integer );

begin
  Inc( DReg.RGB[ Comp ], Incr );              { Incrémente la composante}
  If DReg.RGB[ Comp ] > 63 then                          { Sup à 63 ? }
    DReg.RGB[ Comp ] := 0;                                { Oui, remet à 0 }
  SetDac( Num, 1, DReg );                          { Charge le registre DAC }
  PrintDac( DReg, Num, 15 );                    { Affiche le nouveau contenu}
end;

{*****************************************************************
*  Demo : Présente la programmation des registres DAC et le système de *
*         couleur de la carte VGA                                 *
*-----------------------------------------------------------------*
*  Entrée : néant                                                 *
*****************************************************************}

procedure Demo;

var x,  y,
    ix, jx,
    iy, jy,
    k,  f  : integer;                          { Compteur d'itérations }
    ch     : char;                                        { Touche }
    dacbuf : DACARRAY;                          { Table DAC complète }
    DReg   : DACREG;                            { Registre DAC courant }

begin
{-- Dessine l'écran ------------------------------------------------}

  SetPage( 0 );                                    { Traite la page 0 }
  ShowPage( 0 );                                   { Affiche la page 0 }
  GetDac( 0, 256, dacbuf );                 { Charge la table DAC complète }

  GrafPrint( 10, 0, 255, 0,
             'VDACP  -  (c) 1991 by MICHAEL TISCHER' );

{-- Construit le bloc de 16*16 champs de couleur -------------------}

  iy := STARTY;                              { Point de départ sur l'écran }
  jy := STARTY + HAUTEUR - 1;
  f  := 0;
  for y := 0 to 15 do                    { Parcourt les 16 lignes de blocs }
    begin
      ix := STARTX;
      jx := STARTX + LARGEUR - 1;
      for x := 0 to 15 do             { Parcourt les 16 colonnes de blocs }
        begin
          for k := iy to jy do   { Dessine les blocs avec des segments }
            Line( ix, k, jx, k, f );
          Inc( ix, LARGEUR + DISTANCE );        { Prochain bloc à droite}
          Inc( jx, LARGEUR + DISTANCE );
          Inc( f );                                  { Couleur suivante }
        end;
      Inc( iy, HAUTEUR + DISTANCE );                 { Position suivante }
      Inc( jy, HAUTEUR + DISTANCE );
    end;

{-- Lit les entrées de l'utilisateur et réagit en conséquence ------}

  ix := 0;                  { Commence en haut à gauche avec la couleur 0 }
  iy := 0;
  jx := 0;
  jy := 0;
  k := 0;
  GetDac( 0, 1, DReg );                                  { Lit la couleur 0 }
  Encadre( 0, 0, 15 );                                   { Encadre le champ }
  PrintDac( DReg, 0, 15 );                          { et affiche le contenu }
  repeat
    ch := ReadKey;                                     { Attend une frappe }
    If ( ch <> #0 ) then                                { Code étendu ? ? }
      case ch of                                        { Non, on exploite }
        'r' : ChangeDacReg( DReg, k, 1, +1 );                { r = Rouge + }
        'v' : ChangeDacReg( DReg, k, 2, +1 );                { v = Vert + }
        'b' : ChangeDacReg( DReg, k, 3, +1 );                { b = Bleu + }
        'R' : ChangeDacReg( DReg, k, 1, -1 );                { R = Rouge - }
        'V' : ChangeDacReg( DReg, k, 2, -1 );                { V = Vert - }
        'B' : ChangeDacReg( DReg, k, 3, -1 );                { B = Bleu - }
        ' ' : begin                  { Space = rétablit la valeur d'origine }
                DReg := dacbuf[ k ];
                ChangeDacReg( DReg, k, 1, 0 );
              end;
      end
    else                                                  { Code étendu }
      case ReadKey of
        #72 : If ( iy = 0 ) then                    { Curseur vers le haut }
                jy := 15
              else
                jy := iy - 1;

        #80 : If ( iy = 15 ) then                   { Curseur vers le bas }
                jy := 0
              else
                jy := iy + 1;

        #75 : If ( ix = 0  ) then                       { Curseur à gauche }
                jx := 15
              else
                jx := ix - 1;

        #77 : If ( ix = 15 ) then                       { Curseur à droite }
                jx := 0
              else
                jx := ix + 1;
      end;

    If ( ix <> jx ) or ( iy <> jy ) then          { Nouvelle position ? }
      begin                                                       { Oui }
        Encadre( ix, iy, 0 );                       { Enlève l'ancien cadre }
        Encadre( jx, jy, 15 );                     { Trace un nouveau cadre }
        ix := jx;                  { Mémorise le nouveau champ de couleur }
        iy := jy;
        k := iy*16+ix;                              { Calcule son numéro }
        GetDac( k, 1, DReg );                      { Charge le registre DAC}
        PrintDac( DReg, k, 15 );                              { et l'affiche }
      end;
  until ch = #13;              { On recommence jusqu'à frappe de <Entrée>}

  SetDac( 0, 256, dacbuf );                        { restaure la table DAC}
end;

{-----------------------------------------------------------------}
{--                        Programme principal                  ----}
{-----------------------------------------------------------------}

begin
  If IsVga then                                  { A-t-on une carte VGA ? }
    begin                                                  { Oui, c'est parti }
      Init320400;                              { Initialise le mode graphique }
      Demo;
      Textmode( C080 );                                { Rétablit le mode texte }
    end
  else
    writeln( 'VDACP  -  (c) 1991 by MICHAEL TISCHER'#13#10#10 +
             'Attention ce programme exige une carte VGA ' +
             '.'#13#10 );
end.
```

Listing : V3240PA.ASM (déjà présenté en 4.8.7)

4.8.9. Les Sprites

L'utilisateur d'un ordinateur crée quelque chose de plus fascinant que de simples graphiques en mouvement. Quand un Pac-Man bondit sur l'écran, un vaisseau spatial pivote autour de son axe personnel ou un dinausore surgit subitement des ténèbres imaginaires d'une jungle digitalisée, voilà de quoi émerveiller la majorité des spectateurs.

A vrai dire, ce qui se cache réellement derrière une telle animation est pratiquement toujours un énorme travail de programmation à calculer en l'espace d'une douzaine d'heures.

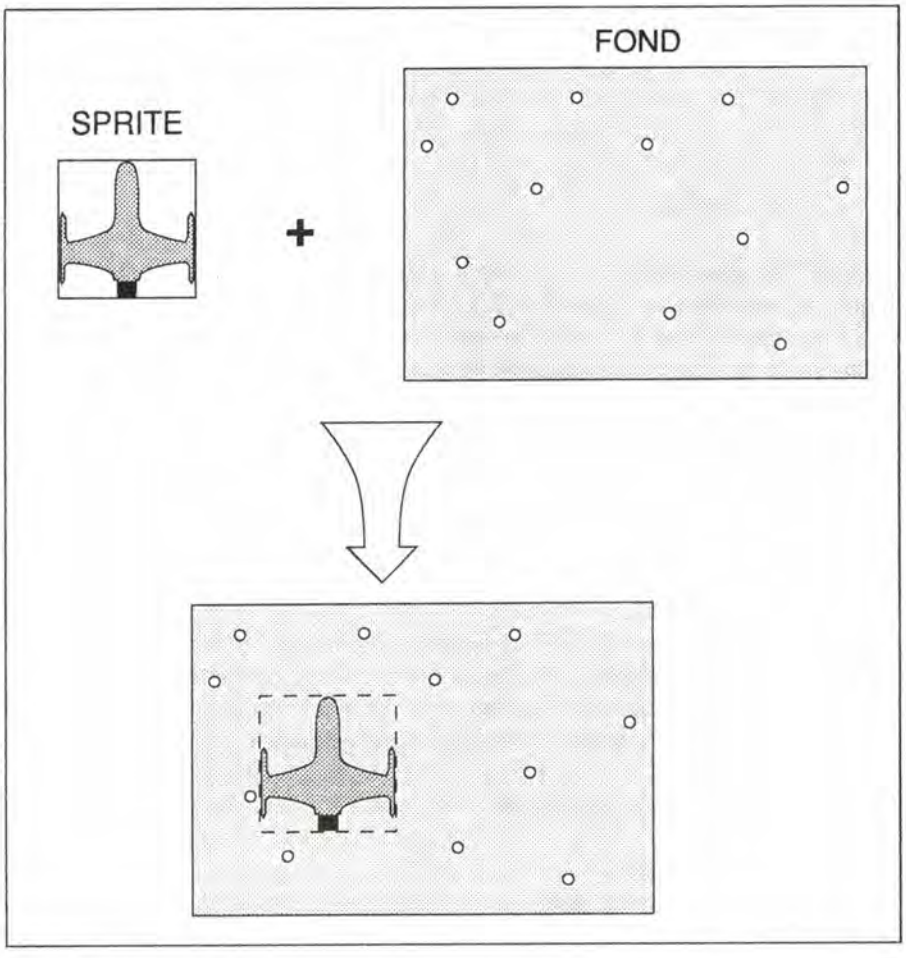
L'électronique vidéo du PC n'aide pas l'utilisateur de manière suffisante dans la réalisation de ces animations. Voici la mise en oeuvre de ce travail.

La question principale posée dans ce chapitre est de savoir comment réaliser une animation sur le PC malgré les limitations imposées. Concrètement, il s'agit de programmer ce qu'on appelle des Sprites, ou mieux des objets graphiques, utilisables dans la majorité des jeux. Voici les différents thèmes étudiés :

- ✓ Que sont que les Sprites et quel rôle jouent-ils dans la programmation des animations ?
- ✓ Le principe des 2 pages
- ✓ Programmation de Sprites dans les divers modes graphiques.

Que sont les Sprites ?

Un Sprite est un bloc de points rectangulaire affecté à une image déterminée telle un Pac-Man. Un tel objet peut être déplacé sur l'écran à l'aide de routines appropriées. A cet effet, il est utile d'assurer une transparence avec le fond pour que le bloc rectangulaire, à savoir le Sprite, ne recouvre pas le fond. Sinon, l'observateur ne peut pas distinguer à tout moment la forme rectangulaire du Sprite. Pour éviter ce risque, on peut colorier entièrement le fond avec une seule couleur et lui affecter la même couleur que celle utilisée pour tous les points du rectangle Sprite ne faisant pas partie de l'objet à afficher. Mais cela réduit considérablement la qualité de l'animation.



Affichage Sprite transparent

Il faut pouvoir en outre affecter un Sprite à une autre image pendant l'exécution du programme. Le but est de créer des changements de situation pour mettre par exemple en évidence l'alimentation d'un vaisseau spatial en énergie ou la perte d'un avion. Et là encore c'est une tâche qui fait défaut dans le cadre de la programmation système.

La pure programmation consiste en revanche à spécifier les coordonnées du déplacement des Sprites pendant l'exécution du programme et déterminer les collisions. Cette section ne vous décharge pas de cette tâche mais fournit toutes les routines Sprite nécessaires pour la bonne mise en oeuvre des Sprites et pour la programmation système. A côté des routines de définition de l'image d'un Sprite, vous trouverez des routines de déplacement sur l'écran pour faire disparaître les Sprites et pour changer subitement l'image affichée.

La programmation de ces Sprites est démontrée à l'aide des cartes EGA et VGA. L'exemple porte d'abord sur les deux modes 256 couleurs de la carte VGA (décrits en 4.8.7) puis sur le mode graphique 16 couleurs des cartes EGA et VGA avec une résolution de 640*350 points parce que mode est plus difficile à programmer que les modes 256 couleurs.

Dans les programmes d'exemple présentés en une version Pascal et C, l'écran est tout d'abord rempli avec les caractères du jeu PC représentant en quelque sorte le fond de l'affichage Sprite. Sur la base de ce fond apparaissent six Sprites en forme de petits vaisseaux se déplaçant verticalement à travers l'écran. Ils se heurtent tôt ou tard contre le bord supérieur ou inférieur de l'écran, ce qui modifie non seulement leur direction mais aussi leur aspect. Les modules en langage évolué sont soutenus également par des routines Assembleur appropriées allant à l'encontre de la haute vitesse d'exécution lors du déplacement des Sprites.

Le principe des 2 pages

Un principe fondamental lié au travail des animations concerne également les Sprites et il s'agit en l'occurrence d'utiliser simultanément deux pages écran. C'est la seule méthode permettant d'éviter le vacillement de l'écran qui risque d'apparaître incontestablement avec une seule page écran. Le responsable n'est autre que le mécanisme permettant d'afficher et déplacer les Sprites sur l'écran.

Cette procédure se réalise en plusieurs étapes au cours desquelles la région couverte par le Sprite doit être traitée plusieurs fois. Il faut d'abord sauvegarder le fond du Sprite pour qu'il puisse être restauré par la suite lorsqu'il faut le déplacer en un autre endroit de l'écran. Cette manipulation s'effectue évidemment à l'insu de l'observateur par la simple raison qu'une partie déterminée de la RAM vidéo est lue au départ.

A l'étape suivante, le Sprite est copié à l'endroit qui lui est réservé dans la RAM vidéo et rendu ainsi visible sur l'écran. Cela ne fait pas vaciller l'écran mais le problème surgit lorsqu'il faut déplacer le Sprite pour donner une impression de mouvement. Dans ce cas, il faut d'abord restaurer l'ancien fond en recopiant le modèle de points trouvé à cet endroit dans la RAM vidéo.

Ce processus est visible par l'utilisateur et provoque la disparition du Sprite de l'écran. Toujours est-il que pour occuper la nouvelle position, la totalité du cycle de sauvegarde, d'écriture, de restauration, etc. recommence depuis le début après la restauration de la portion de l'écran recouverte par le Sprite.

Cet effet étant impossible à réaliser avec une seule page écran, on utilise deux pages écran dont l'affichage s'interchange en permanence pour programmer les Sprites. La page écran traitée est justement celle qui n'est pas affichée pour que les modifications ne soient pas visibles par l'utilisateur jusqu'à ce que cette page écran soit éditée et que le traitement de la suivante ait commencé.

Le recours à deux pages écran explique généralement pourquoi vous recherchez en vain un module Sprite pour le mode graphique VGA à forte résolution avec ses 640*480 points. Ce mode n'offre pas deux pages écran parce que la première page nécessite déjà 150 Ko et la seconde n'en dispose que de 106 Ko. La création de Sprites dans les modes graphiques 16 couleurs des cartes EGA et VGA reste limitée au mode EGA 640*350 points. Disponible également sur la carte VGA, il est amplement suffisant pour la plupart des jeux qui représentent le débouché principal des Sprites.

Quel que soit le mode vidéo et la manière dont un Sprite est écrit dans la RAM vidéo et son fond sauvegardé, de nombreux problèmes surviennent lors de la mise en oeuvre des Sprites. Ces problèmes sont évoqués dans les programmes présentés dans cette section à travers des routines de même nom. Les structures de données portent aussi des noms identiques surtout celles qui sont utilisées ici même si les différentes caractéristiques de Pascal et C se reproduisent naturellement dans ce contexte.

En raison de ces similitudes, il convient de souligner d'abord les ressemblances entre les programmes et leur structure fondamentale avant d'évoquer les problèmes en détail et de ce fait les différences. Ils se distinguent surtout par la diversité du mode d'organisation de la RAM vidéo dans les modes graphiques. Ils constituent le point de départ pour vos programmes personnels si vous souhaitez adapter les modules présentés à une autre carte vidéo ou à l'un des modes graphiques étendus des cartes Super VGA.

Structure des programmes Sprite

Le point de départ des programmes Sprite est une routine appelée DEMO. Elle est appelée à partir du programme principal ou de la fonction MAIN() dès que la présence d'une carte EGA ou VGA a été confirmée. Le mode graphique nécessaire est d'abord mis en place pour que DEMO puisse fonctionner en mode graphique.

A l'intérieur de DEMO, les pages 0 et 1, utilisées ci-après, sont préalablement remplies par un modèle de caractères du jeu PC. Les routines telles que GRAFPRINTF (module C) ou PRINTCHAR (module Pascal) présentées en 4.8.6 et 4.8.7 entrent alors en action. Ces routines se basent sur les modules V3220CA.ASM, V3240PA.ASM décrites en 4.8.7 et contiennent des routines de base pour le travail en modes graphiques. Il s'agit de routines pour définir et consulter des points, pour afficher et commuter entre les pages graphiques et pour initialiser un mode graphique précis.

Après la sortie des différents caractères constituant le fond des Sprites à afficher, un message de Copyright apparaît au milieu de l'écran. Il reste ensuite à définir les Sprites à l'aide d'une routine portant le nom COMPILESPRITE dans tous les modules Sprite.

Sa tâche consiste à transformer une image représentée sous la forme d'un tableau chaîne en un format binaire utilisé ensuite pour afficher les Sprites sur l'écran. Cette routine ne permet pas pour l'instant de créer un Sprite puisque cette tâche est assurée par une autre routine appelée CREATESPRITE appelée à l'étape suivante.

L'appel de COMPILESPRITE se traduit exclusivement par la création d'un modèle de bits affectés par la suite à un nombre quelconque de Sprites en guise d'image affichée. Un nombre varié de paramètres est à transmettre à COMPILESPRITE en fonction du module Sprite concerné. Dans tous les modules, les deux premiers paramètres ont la même signification. Le premier représente le tableau chaîne stockant l'image du Sprite et le second reproduit le nombre de chaînes contenues dans ce tableau.

Ce paramètre définit simultanément la hauteur du Sprite variant entre une centaine de lignes de points et même davantage. Chaque ligne de points est représentée par une chaîne du tableau où la première chaîne correspond à la ligne de points supérieure du Sprite, la chaîne suivante à la seconde ligne de points, etc.

A l'intérieur des différentes chaînes, chaque caractère est réservé à un point graphique de la ligne de points concernée. Un Sprite étant toujours représenté dans un cadre rectangulaire, toutes les chaînes du tableau doivent avoir une largeur identique. Mais comme la hauteur du Sprite est à transmettre comme un paramètre à COMPILESPRITE, il n'est pas indispensable de la spécifier car elle peut être obtenue à partir de la largeur de la première chaîne dans le tableau.

```
static char *SPRITE_HAUT [20] =
              { "                AA              ",
                "               AAAA             ",
                "               AAAA             ",
                "                AA              ",
                "              GGBBGG            ",
                "             GBBCCBBG           ",
                "            GBBBCCBBBG          ",
                "           GBBBBBBBBBBG         ",
                "           GBBBBBBBBBBG         ",
                " G       GBBBBBBBBBBBBG       G ",
                "GCG     GGDBBBBBBBBBBDGG     GCG",
                "GCG   GGBBBDBBB  BBBDBBBGG   GCG",
                "GCBGGGBBBBBDBB     BBDBBBBBGGGBCG",
                "GCBBBBBBBBBBDB     BDBBBBBBBBBBCG",
                "BBBBBBBBBBBBDB BB BDBBBBBBBBBBBB",
                "GGCBBBBBBBDBBBBBBBBBDBBBBBBBCG ",
                "  GGCCBBBDDDDDDDDDDDDDBBBCCG    ",
                "    GGBBDDDDDGGGGGDDDDDDBBG     ",
                "      GDDDDGGG    GGGDDDDG       ",
                "       DDDD         DDDD        " };
```

Codage d'une image d'un Sprite dans un tableau C (voir ligne de listing en haut)

Les modules Sprite n'étant pas conçus pour être utilisés en modes graphiques monochromes, il ne suffit pas de faire la différence entre l'état "Point oui" ou "Point non" à l'intérieur des lignes de points. Dans la chaîne, il faut en outre coder une couleur à affecter au point graphique concerné du Sprite. Ainsi, le crochet correspond au noir

(code couleur 0), le A majuscule au bleu (code couleur 1), le B au vert (code couleur 2) et toutes les lettres suivantes aux autres couleurs de l'échelle des couleurs.

Les points n'appartenant pas au Sprite, c'est-à-dire ceux qui doivent être transparents pour rendre le fond visible, sont indiqués par un espace.

Quel que soit le mode utilisé par COMPILESPRITE pour convertir le modèle Sprite en un affichage binaire dans les divers Sprites, un pointeur est retourné obligatoirement sur la structure de type SPLOOK définie dans COMPILESPRITE à travers la Heap. Elle contient toutes les informations importantes concernant l'image du Sprite et doit servir pour créer des Sprites. En fait, complètement à l'inverse de la définition initiale du Sprite d'après le tableau chaîne qui n'a plus aucun intérêt après l'appel de COMPILESPRITE.

Dans tous les programmes Sprite, deux Sprites sont définis à l'aide de COMPILESPRITE. Ils sont représentés par deux tableaux chaîne SPRITEHAUT et SPRITEBAS et affichent simultanément deux navettes spatiales de même nature se déplaçant verticalement dans le sens opposé. Le pointeur retourné est utilisé pour créer les Sprites à afficher ensuite sur l'écran dans une boucle.

La constante SPRANZ définit le nombre de boucles à exécuter. Dans tous les Sprites, elle est réglée sur la valeur six. Elle spécifie en fait le nombre de Sprites créés car l'appel de CREATESPRITE provoque la création d'un Sprite chaque fois que la boucle est traitée. D'après les expériences que vous avez pu acquérir avec les Sprites, vous savez que cette constante peut aisément être augmentée afin d'afficher plus de Sprites sur l'écran. Les Sprites tendent alors à se rapprocher davantage jusqu'à se superposer. Cela finit par produire des effets disgrâcieux sur l'écran parce que les diverses routines Sprite ne sont pas en mesure de manipuler la collision entre les Sprites comme nous l'avons déjà signalé plus haut.

Mais avant d'en arriver là, un nombre Sprite précis vous indique que les Sprites se mettent soudain à se déplacer à soubresaut sur l'écran alors que le mouvement redevient fluide dès qu'on supprime un Sprite. La valeur entraînant une telle réaction dépend considérablement de la vitesse de l'ordinateur et sa carte graphique. La cause de ce phénomène est expliquée plus loin lors de la description du déplacement des Sprites.

La fonction CREATESPRITE permet de créer les Sprites à l'intérieur de la boucle citée plus haut. En guise de premier paramètre, elle attend un pointeur sur l'image d'un Sprite comme celui retourné par COMPILESPRITE. La taille et l'aspect du Sprite à créer sont également à définir mais vous pouvez les modifier ultérieurement. Les autres paramètres à transmettre à CREATESPRITE dépendent du programme Sprite concerné et ne sont pas décrits dans le cadre de ces programmes.

Quelle que soit son utilité dans les modules Sprite, CREATESPRITE retourne toujours un pointeur sur une structure de type SPID contenant toutes les données relatives au Sprite créé. Outre le pointeur sur l'image, la position actuelle du Sprite dans les deux pages écran et d'autres informations liées à la version sont fournies. Comme pour COMPILESPRITE, cette structure est définie à travers la Heap.

Des Sprites sont créés à l'intérieur de la boucle pour la création des Sprites, mais leur état initial sur l'écran ainsi que leur vitesse de déplacement y sont également définis. Dans les deux cas, ces informations sont sélectionnées à travers une fonction aléatoire où la vitesse de déplacement en direction X représentée par la variable locale DX est réglée sur 0. Le but est d'obliger les Sprites à se déplacer uniquement sur le plan vertical pour qu'ils ne viennent pas s'entrechoquer. Essayez tout de même de sélectionner une valeur différente de 0 pour tester la réaction des Sprites lors de vos expériences.

Le pointeur est fixé sur le Sprite concerné (ou mieux sur sa structure SPID) et représente la vitesse de déplacement dans une variable locale nommée SPRITES, un tableau composé d'une structure simple.

Les Sprites ainsi créés sont affichés sur l'écran à l'aide de la procédure SETSPRITE. Comme premier paramètre, elle attend un pointeur sur le descripteur Sprite retourné par CREATESPRITE. Les autres paramètres demandés sont les coordonnées X et Y de la position Sprite dans la première page écran (page 0). Les mêmes informations sont exigées pour la seconde page écran (page 1).

Les deux coordonnées sont traitées séparément parce qu'elles ne doivent absolument pas être identiques à cause du mécanisme de l'affichage sur deux pages. Sinon, le Sprite risque de réapparaître à la même position après la commutation de la première page vers la seconde, ce qui ne permet plus de simuler un mouvement fluide. La vitesse de ce mouvement commande d'ailleurs le déplacement de la position Sprite dans la seconde page par rapport à la première.

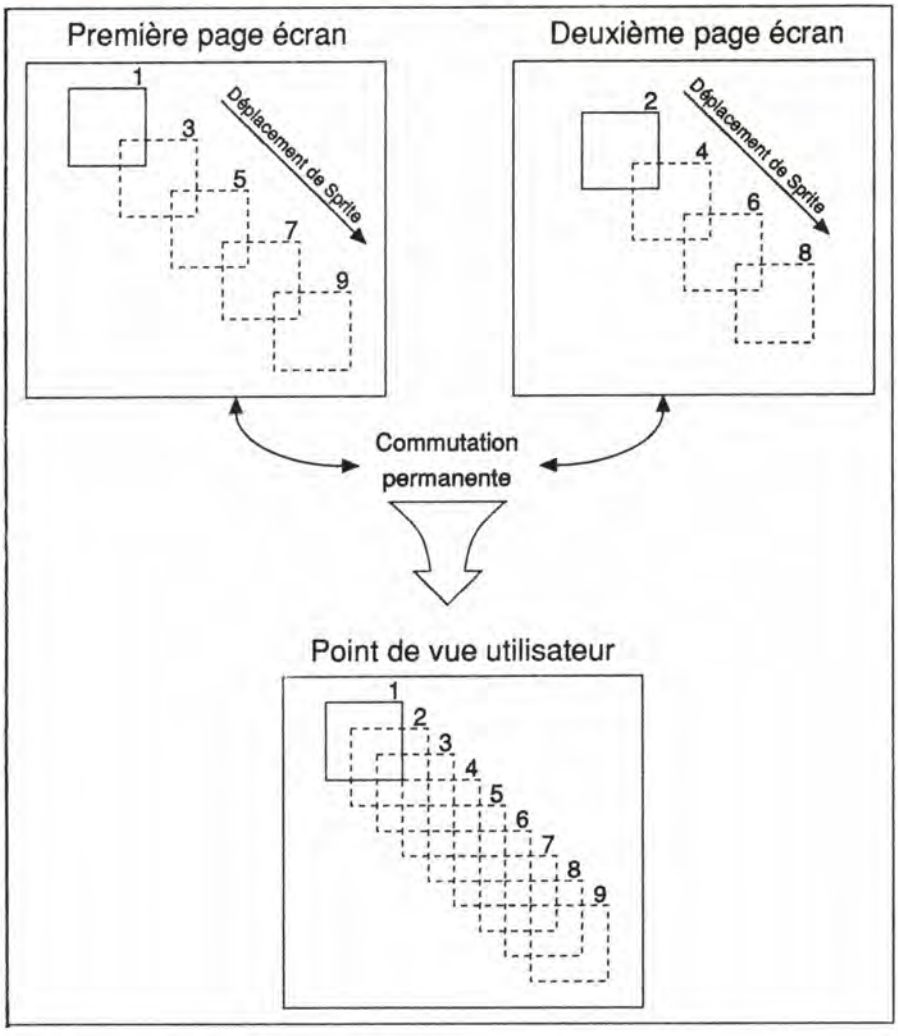
Mouvement séparé

Imaginez qu'un Sprite doit se déplacer verticalement sur l'écran et notamment depuis le bord supérieur avec la coordonnée Y 0 vers le bord inférieur. A chaque nouvelle configuration de l'écran, invisible par l'utilisateur, le Sprite doit se décaler d'un point par rapport à son origine.

Au début du mouvement, il doit être construit dans la première page écran à partir de la coordonnée Y 0. Dans la seconde, il doit apparaître à la coordonnée Y 1 pour que l'utilisateur ait l'impression que le Sprite s'est décalé d'un point lors du passage de la première page écran vers la seconde.

Lors de l'affichage de la seconde page écran, le Sprite doit être décalé de deux points dans la première page et non d'un point. Avec un décalage d'un point, le Sprite risque de se trouver à la même position qu'il occupait dans la seconde page lors de la commutation. Avec un décalage de deux points, il se place un point en arrière par rapport à la position de la seconde page et lors de la commutation vers la première page, l'utilisateur remarque un déplacement. La vitesse de déplacement effective doit être doublée eu égard au déplacement à l'intérieur des différentes pages écran.

Tout ce qui concerne ici l'axe Y vaut naturellement pour l'axe X. Le déplacement de la position de départ représentée dans cette figure et le doublement de la vitesse de déplacement doivent être respectés dans ce contexte.



Affichage et déplacement des Sprites dans diverses les pages écran

Une fois les Sprites créés et affichés, ils sont ensuite déplacés dans une boucle dont l'exécution ne s'arrête que lorsque l'utilisateur appuie sur une touche. Une commutation entre la première page et la seconde s'effectue dans cette boucle et les Sprites sont dirigés vers la page qui n'est justement pas affichée.

Pour chaque Sprite, une fonction MOVESPRITE est appelée à cet effet dans une boucle. Elle représente une des rares routines Sprite qui conviennent telles quelles dans tous les

programmes Sprite parce qu'elles se basent exclusivement sur des routines Sprite d'ordre inférieur où se manifestent les différences entre les Sprites. MOVESPRITE ne se laisse pas influencer par les diverses utilisations de ces routines.

En guise d'arguments, cette routine reçoit un pointeur sur le descripteur Sprite, le numéro de la page où le déplacement doit avoir lieu ainsi que le nombre de points dans les sens X et Y à prendre en compte pour déplacer le Sprite. La position Sprite en cours et la vitesse de déplacement en points permettent de calculer la nouvelle position Sprite en faisant attention au heurt avec le bord de l'écran. Le déplacement du Sprite est en fonction de la nouvelle position Sprite et non l'ancienne.

On le rend d'abord invisible à sa position en cours en restaurant son arrière-plan initial à l'aide de la routine RESTORESPRITEBG. L'arrière-plan est ensuite configuré pour la future position Sprite à l'aide de GETSPRITEBG et stocké dans un buffer interne. En dernière phase, le Sprite est placé à sa nouvelle position à l'aide d'une nouvelle routine nommée PRINTSPRITE.

Comme résultat, MOVESPRITE retourne un octet dont les bits décrivent les chocs subis avec les bords de l'écran lors du déplacement. Ces collisions peuvent être consultées au moyen des constantes OUT_LEFT, OUT_TOP, OUT_RIGHT, OUT_BOTTOM et OUT_NO.

Elles vont servir également dans DEMO car à la suite d'une collision avec le bord supérieur ou inférieur de l'écran, le sens de déplacement du Sprite se modifie ainsi que son image. Vous apercevez alors le nez de la navette en position de vol, ce qui est d'ailleurs naturel pour une navette spatiale convenable.

La vitesse d'exécution de la boucle de déplacement à l'intérieur de DEMO et par conséquent la vitesse de déplacement du Sprite dépend essentiellement de la vitesse du processeur, de la carte vidéo et du système bus qui réunit les deux. La fréquence d'image joue un rôle non moins important car la permutation de la page écran affichée à l'intérieur de la boucle de déplacement au moyen de SHOWPAGE tend à synchroniser cette boucle avec la fréquence d'image.

La responsabilité de cette action incombe à une boucle de traitement dans la routine Assembleur SHOWPAGE. Elle veille à ce que cette routine ne soit retournée que lorsqu'une nouvelle configuration de l'écran a commencé. La définition d'une nouvelle page écran ne devient effective qu'à cette condition et rendue visible à l'utilisateur. Cette méthode empêche que l'utilisateur de SHOWPAGE n'accède à une page écran qui est encore momentanément visible parce que la configuration suivante n'a pas encore commencé.

Dans certains cas, cela fait perdre un cinquantième de secondes selon la fréquence d'image parce que l'appel de SHOWPAGE s'effectue une fois la nouvelle configuration commencée. La conséquence de ce ralentissement est que le temps nécessaire pour commuter entre deux pages augmente considérablement à partir d'un certain nombre de Sprites si bien que l'observateur n'a plus une impression de continuité. Nous

rencontrons à nouveau le mouvement saccadé des Sprites dont nous avons déjà parlé auparavant.

D'une part, la vitesse de déplacement des Sprites est déterminée par la vitesse d'exécution de la boucle et par conséquent par le nombre de traitements de la boucle par seconde. D'autre part, on peut agir sur la vitesse des Sprites et aussi sur le nombre de points à tenir en compte pour replacer le Sprite par rapport à son ancienne position. Là aussi, il existe une limite à partir de laquelle le déplacement du Sprite ne s'effectue plus en souplesse.

Pour que votre programme n'ait pas à faire face à des problèmes de vitesse d'exécution dès l'affichage de quelques Sprites, nous avons écrit en Assembleur les principales routines assurant le lien entre le programme et la carte vidéo. Ces routines Assembleur sont appelées indirectement par des routines en langage évolué telles que PRINTSPRITE, GETSPRITEBG et RESTORESPRITEBG. Elles se servent à cet effet des routines GETVIDEO et PUTVIDEO jouant le rôle d'interface directe pour les diverses routines Assembleur.

La tâche de PUTVIDEO et GETVIDEO consiste à lire une portion de l'écran dans un buffer ou remplir une portion de l'écran avec des points dont le modèle est transmis dans un buffer. Les routines PRINTSPRITE, GETSPRITEBG et RESTORESPRITEBG peuvent ainsi afficher un Sprite sur l'écran, sauver son arrière-plan et le restaurer par la suite.

Les routines Assembleur appelées par GETVIDEO et PUTVIDEO comportent une différence portant sur les modes graphiques et programmes Sprite même si cela a des implications sur des routines telles que COMPILESPRITE ou GETVIDEO et PUTVIDEO. Ces routines constituent le point de mire des paragraphes suivants consacrés aux Sprites et leurs modes graphiques.

Les modes 256 couleurs de la carte VGA occupent la place principale. Bien qu'ils ne soient pas accessibles à un utilisateur de carte EGA, ils facilitent néanmoins bien des opérations par rapport aux modes 16 couleurs. Vous comprenez donc pourquoi la description des routines de programmation des Sprites en mode 640*350 points des cartes EGA et VGA se trouve à la fin de cette section.

Sprites en 320*200 points et 256 couleurs

Malgré la faible résolution fournie par les 320*200 points du mode graphique 256 couleurs, l'utilisation des Sprites dans un tel contexte offre un avantage incontestable en mettant à votre disposition deux pages écran et non une. Il va sans dire que deux pages écran sont nécessaires pour déplacer et afficher les Sprites, mais il ne faut pas oublier que les 128 Ko de RAM restants peuvent être utilisés pour sauver les modèles des Sprites ainsi que leurs arrière-plans.

Il ne s'agit nullement d'économiser de la place dans la mémoire principale. Au contraire, l'intérêt est d'obtenir une nette augmentation de la vitesse d'exécution. Pour décrire ou sauvegarder une portion de l'écran, vous vous souvenez sans doute qu'il faut charger un octet à partir de l'un des quatre plans de bits ou inscrire dans l'un des quatre plans de bits sans oublier que le bus système doit apporter sa contribution. Notez que ces limitations n'ont aucune valeur pour la recopie des zones de mémoire à l'intérieur de la RAM vidéo.

Dans ce cas, aucune communication entre l'unité centrale et la carte vidéo à travers le bus système ne s'avère indispensable. Ici, les quatre registres Latch de la carte VGA suffisent pour copier simultanément quatre octets au moyen d'une seule instruction MOVSB. L'instruction Assembleur REP MOVSB permet de surcroît de copier en une seule passe la totalité d'une ligne graphique. Tout cela suffit pour mettre à profit la partie inutilisée de la RAM vidéo pour ce type de tâches même si cela comporte un hic comme nous allons le voir immédiatement.

Mais revenons tout d'abord aux programmes. Dans la version C, les modules S3220C.C, S3220CA.ASM et V3220CA.ASM ont la charge de créer des Sprites en mode graphique 320*200 points avec 256 couleurs. Le module V3220CA.ASM a déjà fait ses preuves dans ce domaine selon la description fournie en 4.8.7. En revanche, les modules S3220C.C et S3220CA.ASM sont nouveaux et où le module C contient les routines en langage évolué depuis COMPILESPRITE jusqu'à GETVIDEO.

Le module Assembleur S3220CA.ASM contient uniquement la routine BLOCKMOVE servant à copier de part et d'autre une zone rectangulaire de points à l'intérieur de diverses parties de la RAM vidéo. Les versions Pascal de ces programmes se comportent de manière similaire et portent les noms S3220P.PAS, S3220PA.ASM et V3220PA.ASM. Encore une fois, V3220PA.ASM a déjà été décrit en 4.8.7.

En raison du souhait exprimé pour placer directement les modèles Sprite dans la RAM vidéo et pouvoir les recopier à l'endroit voulu aussi vite que possible, la structure de la RAM vidéo en mode 320*200 points a entraîné des répercussions mode de fonctionnement du programme. Ainsi, la largeur de chaque Sprite doit être arrondie à un multiple de quatre parce que qu'il faut toujours copier simultanément quatre points et donc quatre octets à travers les quatre registres Latch de la carte vidéo.

Si la valeur n'est pas arrondie, il convient obligatoirement de vérifier que seuls les octets nécessaires ont été effectivement copiés lors de la recopie du dernier bloc de quatre d'une ligne. Cela exclut en fait certains registres Latch du processus de recopie. Une telle performance ne peut être réalisée qu'avec une programmation fort poussée qui n'a aucun rapport avec le simple fait d'arrondir la largeur à un multiple de quatre.

Ce n'est pas le seul obstacle à franchir. Pour peu que l'on commence à concevoir le Sprite modèle à une coordonnée X divisible par quatre dans la RAM vidéo, on ne peut la recopier par la suite qu'à travers une coordonnée X divisible par quatre également. Un Sprite dont la construction commence à la coordonnée X 0 peut être copié à

n'importe quel moment aux coordonnées X 4, 8, 96 ou 224 et non aux coordonnées X telles que 1, 2, 5, 13 ou 182.

Cela s'explique tout simplement par le fait qu'un accès en lecture à la RAM vidéo en mode graphique 320*200 points ne peut commencer qu'à un point dont la coordonnée X est un multiple de quatre à condition que les quatre registres Latch soient chargés simultanément. La même règle s'applique lorsqu'il s'agit de reporter ensuite le contenu de ces registres dans la RAM vidéo.

Pour décaler les quatre points des quatre registres Latch d'un point vers la droite, ce qui revient à décaler par exemple le Sprite vers la coordonnée X 1, 5 ou 25, on devrait :

- ✓ décaler chaque fois le contenu des registres Latch d'un registre après le chargement, c'est-à-dire du registre Latch 0 vers le registre Latch 1, du 1 vers le 2, du 2 vers le 3, etc. mais cela n'est pas du tout possible ;

- ✓ marquer le contenu du quatrième registre Latch pour l'opération d'écriture suivante en copiant auparavant dans le registre Latch 0, ce qui est également une action impossible ;

- ✓ masquer le registre Latch 0 avant l'écriture pour que son contenu ne soit pas écrasé dans la RAM vidéo à cause de la coordonnée X divisible par quatre. Cela est par contre admis.

Sachant que deux de ces trois opérations sont impossibles à réaliser, cela suppose naturellement une programmation plus importante. Mais on se contente tout simplement de placer quatre copies d'un Sprite dans la RAM vidéo en commençant la structure du premier à partir d'une coordonnée X divisible par quatre. On installe le second immédiatement à droite ou plutôt un point à droite et non à une coordonnée X divisible par quatre. Ainsi, une colonne de points reste disponible dans la marge gauche du Sprite. Celle-ci est copiée ultérieurement mais elle est considérée comme transparente pour mettre en évidence l'arrière-plan tout en restant invisible.

Le même principe est appliqué aux troisième et quatrième Sprites. On leur affecte deux ou trois colonnes de points et non une seule à être considérées comme transparentes.

Pour que le Sprite apparaisse par la suite sur l'écran, il faut d'abord recalculer la coordonnée X sur une coordonnée X divisible par quatre. La différence entre la coordonnée X spécifiée et le résultat du calcul indique le nombre de colonnes vides au début du Sprite et reproduit ainsi le numéro du Sprite qui sert pour la recopie. Voici la formule à utiliser :

```
CibleX = int(X/4) ou tout simplement X and not (4)
Sprite = X - int(X/4) * 4 ou tout simplement X and 3
```

Il reste un dernier problème à évoquer celui du fond transparent. Tant qu'on se contente de charger les quatre registres Latch et écrire leur contenu à la position cible, on copie

en fait la totalité du rectangle Sprite sans toucher l'arrière-plan. Avant l'écriture des quatre registres Latch, il faut masquer les registres Latch à travers le registre Map Mask du contrôleur Sequencer qui risquent sinon d'écraser un point d'arrière-plan.

Vous n'êtes pas obligé de masquer toujours les mêmes registres Latch parce que les mêmes points n'interviennent pas à chaque fois dans les différents groupes de quatre des points Sprite. C'est pourquoi, il convient de programmer individuellement le registre Map Mask avant d'écrire dans les registres Latch. Il faut choisir une valeur calculée lors de la compilation de l'image Sprite dans COMPILESPRITE et transmise comme la partie d'un tableau à la routine BLOCKMOVE.

Tout cela complique quelque peu la tâche mais il faut obligatoirement passer par là si l'on souhaite copier directement les Sprites dans la RAM vidéo. Encore heureux que cette action n'est indispensable que lors de l'affichage des Sprites. Il faut savoir que le contenu des quatre registres Latch peut être écrasé définitivement de la RAM vidéo en cas de sauvegarde du fond du Sprite ou de sa restauration. Ici, il n'est donc pas indispensable de programmer le registre Map Mask.

Cette méthode qui consiste à conserver les modèles Sprite directement dans la RAM vidéo et les copier avec des valeurs pour le registre Map Mask grâce à un tableau se rencontre naturellement dans les diverses routines des modules S3220C.C et S3220P.PAS ainsi que dans la procédure BLOCKMOVE des modules Assembleur S3220CA.ASM et S3220PA.ASM.

Le processus commence avec la routine COMPILESPRITE. Outre le tableau chaîne avec l'image du Sprite, elle attend la hauteur du Sprite et quatre paramètres supplémentaires. Il s'agit tout d'abord de la page écran dans laquelle les modèles Sprites sont à concevoir. On peut citer ici la page 2 ou 3 vu que les pages écran 0 et 1 sont à utiliser par la suite pour l'affichage du Sprite.

L'argument suivant est la ligne de points à partir de laquelle les modèles sont à concevoir dans la page spécifiée. Ici, vous pouvez choisir une valeur quelconque comprise entre 0 et 200 (hauteur de Sprite). Mais lorsque vous utilisez des Sprites de toutes sortes, veillez à ce que les modèles Sprite ne se superposent pas dans la RAM vidéo.

Dans la ligne spécifiée, les quatre copies du Sprite se placent obligatoirement les unes à côté des autres. Vous pouvez d'ailleurs le constater sur l'écran en affichant la page concernée à l'intérieur de la routine DEMO après l'appel de COMPILESPRITE par simple appel de SHOWPAGE.

Après la ligne de points, COMPILESPRITE attend un caractère représentant le caractère doté du plus petit code couleur dans le tableau chaîne. Il sert à définir l'image du Sprite. Normalement, vous serez tenté d'indiquer ici "A". Mais dans le tableau chaîne, vous pouvez utiliser également des lettres minuscules ou des nombres pour coder la couleur qui vous intéresse. Pour ce paramètre, indiquez tout simplement un A minuscule ou le chiffre 0.

Le dernier paramètre intervient également dans le cadre de la couleur des divers points du Sprite. Il indique le numéro de la couleur à affecter aux points Sprites munis du plus petit code (soit par exemple "A", "a" ou "0). Il représente également la base pour les autres codes couleur. Ainsi, "B", "b" ou "1" correspondent par exemple au code couleur suivant.

Cette méthode permet de lire les couleurs au-delà de 128 sans être obligé de passer par les caractères grecs, les trémas ou les caractères de cadre dans le tableau Sprite. Hormis les deux caractères cités en dernier, les espaces correspondent grosso modo aux points transparents qui laissent apparaître le fond du Sprite.

COMPILESPRITE se sert des informations reçues pour construire d'abord quatre fois le Sprite dans la page citée comme nous l'avons déjà décrit plus haut. Dans ce cas, les points de fond reçoivent le code 255 pour qu'ils puissent se distinguer des points non-transparents.

Les quatre Sprites sont ensuite exécutés pour compléter le tableau. Ce dernier est utilisé ultérieurement pour programmer le registre Map Mask lors de l'appel de BLOCKMOVE. Pour les besoins de ce tableau, la mémoire est d'abord allouée à travers la Heap. Puis pour chaque point des quatre Sprites ainsi créés, un quartet doté de la valeur adéquate y est stocké pour le registre Map Mask.

Tout comme pour les autres informations, le pointeur est également fixé sur ce tableau dans le descripteur de l'image Sprite (SPLOOK) transmis comme d'habitude au moyen d'un pointeur.

De plus amples informations doivent être transmises non seulement à COMPILESPRITE mais aussi à CREATESPRITE contrairement aux autres programmes Sprite. En dehors de l'indispensable pointeur sur le descripteur de l'image Sprite, cette routine attend concrètement l'état des deux zones de la RAM vidéo devant servir à recevoir le fond du Sprite des pages écran 0 et 1. A cet effet, il faut indiquer les coordonnées X et Y à partir desquelles le fond est à définir ainsi que la page écran souhaitée (2 ou 3). Notez qu'il faut une zone dont la largeur représente le double du Sprite concerné et une hauteur simple parce que les deux buffers des pages 0 et 1 vont être placés côte à côte.

Une fois le Sprite créé selon cette méthode, il peut être affiché sur l'écran à l'aide de la routine SETSPRITE et mis en mouvement avec MOVESPRITE. Vous devez appeler comme à l'accoutumée les routines PRINTSPRITE, GETSPRITEBG et RESTORESPRITEBG servant ici à la routine Assembleur BLOCKMOVE. Une routine PUTVIDEO et GETVIDEO fait défaut dans l'implémantation des modules Sprite pour le mode 320*200 points.

BLOCKMOVE attend de nombreux paramètres parmi lesquels les zones cible et source. Ces deux zones sont représentées par le numéro de leur page écran ainsi que les coordonnées X et Y. Cette routine attend en outre la largeur de la zone rectangulaire en points et sa hauteur en lignes de points. Enfin, BLOCKMOVE retourne un pointeur sur le tableau qui contient les valeurs pour la programmation du registre Map Mask.

Si tous les points du rectangle spécifié doivent être copiés en dépit des points du fond, il n'est pas évidemment pas utile d'indiquer un tel tableau. Dans ce cas, la routine attend tout simplement un pointeur nul représenté par la constante prédéfinie NIL dans la version Pascal et par NOBITMASK dans la version C.

Si BLOCKMOVE rencontre un tel pointeur nul lors de son exécution, elle se branche sur une routine de recopie plus rapide que la routine ordinaire qui programme le registre Map Mask avant chaque four byte transfer. Dans cette routine, une ligne complète de points issue de la zone écran spécifiée est copiée simultanément sans tenir compte des points transparents.

Dans tous les cas, le mode Write 1 est défini avant de pénétrer dans la boucle de recopie. Ce mode autorise seulement le transfert simultané du contenu des quatre registre Latch dans les quatre plans de bits. Le mode Write initial est remis en place avant la fin de la routine.

En voilà assez concernant les programmes Sprite en mode graphique 320*200 points avec 256 couleurs. Ils fonctionnent plus vite que les autres programmes Sprite parce la RAM vidéo reste à leur disposition en tant que mémoire intermédiaire pour les modèles et le fond des Sprites. Cet avantage est néanmoins réduit à cause de la faible résolution qui ne correspond même pas au standard auquel l'utilisateur est habitué. Malgré la vitesse quelque peu réduite, nous pensons que le mode graphique 320*200 points prévaut par rapport au mode 320*400 dans la mesure où il faut 256 couleurs et qu'il est impossible de travailler en mode 640*350 points.

Les sections suivantes vous montrent comment exploiter les Sprites dans ces deux modes. Mais voici les listings des modules pour la programmation des Sprites en mode graphique 320*200 points.

Listing : S3220C.C

```
/*****************************************************************
*                          S 3 2 2 0 C . C                       *
******----------------------------------------------------------**
*Fonction          : montre comment travailler avec des sprites dans *
*                    le mode graphique VGA 320*200 en 256 couleurs avec*
*                    quatre pages d'écran                        *
*                                                                *
* Le programme utilise les routines en assembleur des modules    *
*            S3220CA.ASM et  V3220CA.ASM .                       *
******----------------------------------------------------------**
*  Auteur        : MICHAEL TISCHER                               *
*  Développé le :  9.09.1990                                     *
*  Dernière MAJ : 14.02.1992                                     *
******----------------------------------------------------------**
*  Modèle mémoire    : SMALL                                     *
******----------------------------------------------------------**
*  (MICROSOFT C)                                                 *
*  Compilation       : CL /AS s3220c.c v3220ca s3220ca           *
******----------------------------------------------------------**
*  (BORLAND TURBO C)                                             *
*  Compilation       : Utiliser un fichier de projet avec le contenu *
*                      suivant                                   *
*                       s3220c.c                                 *
*                       v3220ca.obj                              *
*                       s3220ca.obj                              *
******----------------------------------------------------------**
*  Appel             : s3220c                                    *
*****************************************************************/

#include <dos.h>
#include <stdarg.h>
#include <stdio.h>
#include <string.h>
#include <stdlib.h>
#include <conio.h>

/*-- Déclarations dépendantes du compilateur--------------------*/

#ifdef __TURBOC__
  #include <alloc.h>
#else
  #include <malloc.h>
  #define random(x) ( rand() % (x+1) )          /* Fonction aléatoire */
#endif

/*-- Déclarations de types -------------------------------------*/

typedef unsigned char BYTE;

typedef struct {                                  /* Image d'un sprite */
                BYTE largeur,                      /* Largeur totale */
                     hauteur,                /* Hauteur en lignes de pixels */
                     page,                        /* Page de mémorisation */
                     *bmskp;               /* Pointeur sur le masque binaire */
```

```
                mskLen;                 /* Longueur d'une entrée */
            int  ligne;             /* Ligne de pixels où commence */
            ) SPLOOK;                   /* le sprite dans sa page */

typedef struct {                        /* Descripteur de sprite (ID) */
            BYTE   fondpage;                        /* Page de fond */
            int    x[2], y[2],      /* Coordonnées en pages 0 et 1 */
                   fondx, fondy;            /* Buffer pour le fond */
            SPLOOK *splookp;            /* Pointeur sur l'image */
            } SPID;

/*-- Références externes aux routines en assembleur ----------------*/

extern void Init320200( void );
extern void setpix( int x, int y, unsigned char couleur);
extern BYTE getpix( int x, int y );
extern void setpage( BYTE page );
extern void showpage( BYTE page );
extern void far * getfontptr( void );
extern void waitvsync( void );
extern void blockmove( BYTE depage, int dex, int dey,
                       BYTE apage, int ax, int ay,
                       BYTE largeur, BYTE hauteur, BYTE *tmskp );

/*-- Constantes ----------------------------------------------------*/

#define NOBITMASK (BYTE *) 0

#define MAXX 319                        /* Coordonnées maximales */
#define MAXY 199

#define OUT_LEFT   1    /* Marquage des collisions dans SpriteMove() */
#define OUT_TOP    2
#define OUT_RIGHT  4
#define OUT_BOTTOM 8
#define OUT_NO     0


/*****************************************************************
*  IsVga: Teste la présence d'une carte VGA.                     *
*-----------------------------------------------------------------*
*  Entrée : néant                                                *
*  Sortie  : 0 si pas de carte VGA, sinon # 0                    *
*****************************************************************/

BYTE IsVga( void )
{
 union REGS Regs;               /* Registres pour gérer l'interruption */

 Regs.x.ax = 0x1a00;             /* La fonction 1AH n'existe qu'en VGA */
 int86( 0x10, &Regs, &Regs );
 return ( Regs.h.al == 0x1a );              /* Est-elle disponible ? */
}
/*****************************************************************
*  PrintChar : Ecrit un caractère en dehors de la zone visible   *
*              de la mémoire d'écran                             *
*-----------------------------------------------------------------*
*  Entrée  :  caractere = caractère à afficher                   *
*             x, y    = Coordonnées du coin supérieur gauche     *
*             cc      = Couleur du caractère                     *
*             cf      = Couleur du fond                          *
*  Info    : Le caractère est dessiné dans une matrice de 8*8 pixels *
*            sur la base du jeu de caractères 8*8 en ROM         *
*****************************************************************/

void PrintChar( char caractere, int x, int y, BYTE cc, BYTE cf )
{
 typedef BYTE   CARDEF[256][8];     /* Structure du jeu de caractères */
 typedef CARDEF far *CARPTR;        /* Pointe sur un jeu de caractères */

 BYTE     i, k,                         /* Compteur d'itérations */
          masque;       /* Masque binaire pour dessiner le caractère */

 static   CARPTR fptr = (CARPTR) 0;      /* Jeu de caractères en ROM */

 if( fptr == (CARPTR) 0 )      /* A-t-on déjà déterminé ce pointeur ? */
   fptr = getfontptr();  /*Non, détermine avec la fonction assembleur */

  /*- Dessine le caractère pixel par pixel ---------------------------*/

 if( cf == 255 )                        /* Caractère transparent ? */
   for ( i = 0; i < 8; ++i ) /* Ne dessine que les pixels du 1er plan */
   {
     masque = (*fptr)[caractere][i];  /* Motif binaire pour une ligne */
     for( k = 0; k < 8; ++k, masque <<= 1 )  /* Parcourt les colonnes */
       if( masque & 128 )                    /* Pixel à dessiner ? */
         setpix( x+k, y+i, cc );                        /* Oui */
   }
 else                                   /* Non dessine chaque pixel */
   for ( i = 0; i < 8; ++i )              /* Parcourt les lignes */
   {
     masque = (*fptr)[caractere][i];  /* Motif binaire pour une ligne */
     for( k = 0; k < 8; ++k, masque <<= 1 )  /* Parcourt les colonnes */
       setpix( x+k, y+i, (BYTE) (( masque & 128 ) ? cc : cf) );
   }
}

/*****************************************************************
*  Line: Trace un segment dans la fenêtre graphique en appliquant *
*        l'algorithme de Bresenham                               *
*-----------------------------------------------------------------*
*  Entrées : X1, Y1 = Coordonnées de l'origine                   *
*            X2, Y2 = Coordonnées de l'extrémité terminale       *
*            COULEUR = couleur dusegment                         *
*****************************************************************/

/*-- Fonction accessoire pour échanger deux variables entières -------*/

void SwapInt( int *i1, int *i2 )
{
 int dummy;

 dummy = *i2;  *i2  = *i1;  *i1  = dummy;
}

/*-- Procédure principale --------------------------------------------*/

void Line( int x1, int y1, int x2, int y2, BYTE couleur )
{
 int d, dx, dy,
     aincr, bincr,
     xincr, yincr,
     x, y;

 if( abs(x2-x1) < abs(y2-y1) )     /* Sens du parcours : axe X ou Y ? */
 {                                                        /* Par Y */
   if( y1 > y2 )                              /* y1 plus grand que y2? */
   {
     SwapInt( &x1, &x2 );                     /* Oui échange X1 et X2, */
     SwapInt( &y1, &y2 );                                /* Y1 et Y2 */
   }

   xincr = ( x2 > x1 ) ? 1 : -1;            /* Fixe le pas horizontal */

   dy    = y2 - y1;
   dx    = abs( x2-x1 );
   d     = 2 * dx - dy;
   aincr = 2 * (dx - dy);
   bincr = 2 * dx;
   x     = x1;
   y     = y1;

   setpix( x, y, couleur );                /* dessine le premier pixel */
   for( y=y1+1; y<= y2; ++y )                 /* Parcourt l'axe des Y */
   {
     if ( d >= 0 )
     {
       x += xincr;
       d += aincr;
     }
     else
       d += bincr;
     setpix( x, y, couleur );
   }
 }
 else                                                     /* par X */
 {
   if( x1 > x2 )                              /* x1 plus grand que x2? */
   {
     SwapInt( &x1, &x2 );                     /* Oui, échange X1 et X2 */
     SwapInt( &y1, &y2 );                                /* Y1 et Y2 */
   }

   yincr = ( y2 > y1 ) ? 1 : -1;             /* Fixe le pas vertical */

   dx    = x2 - x1;
   dy    = abs( y2-y1 );
   d     = 2 * dy - dx;
   aincr = 2 * (dy - dx);
   bincr = 2 * dy;
   x     = x1;
   y     = y1;

   setpix( x, y, couleur );                /* Dessine le premier pixel */
   for(x=x1+1; x<=x2; ++x )                   /* Parcourt l'axe des X */
   {
     if( d >= 0 )
     {
       y += yincr;
       d += aincr;
     }
     else
       d += bincr;
     setpix( x, y, couleur );
```

```
    }
}
}

/*****************************************************************
*  GrafPrintf: Affiche une chaîne formatée sur l'écran graphique *
******************************************************************
*  Entrées : X, Y   = Coordonnées de départ (0 - ...)            *
*            CC     = Couleur des caractères                     *
*            CF     = Couleur du fond  (255 = transparent)       *
*            STRING = Chaîne avec indications de formatage       *
*            ...    = Expressions comme pour printf              *
*****************************************************************/

void GrafPrintf( int x, int y, BYTE cc, BYTE cf, char * string, ... )
{
 va_list  parameter;     /* Liste de paramètres pour les macros VA_...*/
 char     affichage[255],          /* Buffer pour la chaîne formatée */
          *cp;

 va_start( parameter, string );          /* Convertit les paramètres */
 vsprintf( affichage, string, parameter );              /* Formate */
 for( cp = affichage; *cp; ++cp, x += 8 )          /* Affiche la chaîne */
   PrintChar( *cp, x, y, cc, cf );         /* formatée par PrintChar */
}

/*****************************************************************
*  CreateSprite: Crée un sprite à l'aide d'un motif de pixels    *
*                préalablement compilé                           *
******************************************************************
*  Entrée : SPLOOKP = Pointeur sur la structure de données produite*
*                     par CompileSprite                          *
*           FONDPAGE = Page d'écran qui doit mémoriser le fond    *
*                      du sprite                                 *
*           FONDX,   = Coordonnées dans la page de fond où doit être *
*           FONDY      être mémorisé le fond du sprite            *
*  Sortie  : Pointeur sur la structure du sprite créée           *
*  Info    : La mémorisation du fond du sprite nécessite deux zones *
*            contigues de la taille du sprite                    *
*****************************************************************/

SPID *CreateSprite(SPLOOK *splookp, BYTE fondpage, int fondx, int fondy)
{
 SPID *spidp;                     /* Pointe sur la structure du sprite */

 spidp = (SPID *) malloc( sizeof(SPID) );   /* Mémoire pour structure */
 spidp->splookp  = splookp;                 /* Y reporte les données */
 spidp->fondpage = fondpage;
 spidp->fondx    = fondx;
 spidp->fondy    = fondy;

 return spidp;                 /* Retourne un pointeur sur la structure */
}

/*****************************************************************
*  CompileSprite: Crée le motif binaire d'un sprite              *
*                 à l'aide d'une définition connue au moment de  *
*                 l'exécution                                    *
******************************************************************
*  Entrées : BUFP = Pointeur sur un tableau de pointeurs référençant *
*            des chaînes de caractères qui représentent le motif du *
*            sprite                                              *
*            HAUTEUR = Hauteur du sprite et nombre de chaînes de *
*                      caractères                                *
*            PAGE    = Page graphique pour dessiner le sprite    *
*            Y       = Ligne de pixels où commence le dessin     *
*            CLR     = Caractère ASCII associé à la plus petite couleur*
*            COULEURPP = Premier code de couleur pour CLR        *
*  Info    : Les sprites sont dessinés à partir du bord gauche de la *
*            ligne indiquée                                      *
*****************************************************************/

SPLOOK *CompileSprite( char **bufp, BYTE shauteur, BYTE gpage, BYTE y,
                       char clr, BYTE couleurpp )
{
 BYTE   largeur,          /* Longueur des chaînes = largeur des motifs */
        c,                                  /* Mémorise un caractère */
        i, k, l,                           /* Compteurs d'itérations */
        pixc,     /* Compteurs pixels pour compilation masque binaire */
        pixm,
        *lspb;           /* Pointeur courant dans le buffer du sprite */
 int    distance,                    /* Distance entre deux sprites */
        lx, ly;                           /* Coordonnées courantes */
 SPLOOK *splookp;        /* Pointeur sur la structure du sprite créée */

        /*-- Crée une structure d'image de sprite et la remplit -----*/

 splookp          = (SPLOOK *) malloc( sizeof(SPLOOK) );
 largeur          = strlen( *bufp );
 distance         = splookp->largeur = ( ( largeur + 3 + 3 ) / 4 ) * 4;
 splookp->bmskp   = (BYTE *) malloc( (distance*shauteur+7)/8*4 );
 splookp->hauteur = shauteur;
 splookp->ligne   = y;
 splookp->page    = gpage;
 splookp->msklen  = ( distance * shauteur + 7 ) / 8;

 /*-- Remplit le fond du sprite dans sa page d'origine
      avec les codes pour le fond de caractères transparent------*/

 setpage( gpage );                          /* Fixe la page de dessin */
 for( ly = y + shauteur-1, lx = 4 * distance - 1; ly >= (int) y; --ly)
   Line( 0, ly, lx, ly, 255 );

       /*-- Dessine quatre fois le sprite dans sa page d'origine -----*/

 for( l = 0, lx = 0; l < 4; ++l, lx+=distance+1 )
   for( i = 0; i < shauteur; ++i )
     for( k = 0; k < largeur; ++k )
       setpix( lx+k, y+i, ( c = *(*(bufp+i)+k) ) == ' '
                   ? 255
                   : couleurpp+(c-clr));

 /*-- Parcourt les quatre sprites créés et génère des masques binaires */
       /*-- pour copier les sprites dans les plans de bits          -*/

 for( l = pixm = pixc = 0, lx = 0; l < 4; ++l, lx+=distance )
 {
   lspb = splookp->bmskp + splookp->msklen * l;
   for( i = 0; i < shauteur; ++i )
     for( k = 0; k < distance; ++k )
     {
       pixm >>= 1;              /* Décale masque pixels d'1 bit a droite */
       if( getpix( lx+k, y+i ) != 255 )          /* Pixel de fond ? */
         pixm |= 128;                        /* Non fixe un bit du masque */
       if( ++pixc == 8 )           /* A-t-on déjà traité huit pixels? */
       { /* Oui mémorise le masque binaire dans le buffer des sprites */
         *lspb++ = pixm;
         pixc = pixm = 0;       /* compteur de pixels et le masque à 0 */
       }
     }
   if( pixc )                        /* Dernier quartet dans le buffer ? */
   {                                                          /* Non */
     *lspb = pixm >> 4;  /* quartet haut dans quartet bas et mémorise */
     pixc = pixm = 0;           /* compteur de pixels et le masque à 0 */
   }
 }
 return splookp;     /* Retourne un pointeur sur le buffer des sprites */
}

/*****************************************************************
*  PrintSprite : Affiche un sprite dans une page donnée          *
******************************************************************
*  Entrées : SPIDP = Pointeur sur la structure du sprite         *
*            PAGE = Page concernée                               *
*                   (0 ou 1)                                     *
*****************************************************************/

void PrintSprite( register SPID *spidp, BYTE page )
{
 BYTE   largeur;                          /* Largeur totale du sprite */
 int    x;                       /* Abscisse X du sprite dans sa page */
 SPLOOK *splookp;                   /* Pointeur sur l'image du sprite */

 largeur = (splookp = spidp->splookp)->largeur;
 x       = spidp->x[page];
 blockmove(
           splookp->page,
           largeur * (x % 4),
           splookp->ligne,
           page,
           x & 3,
           spidp->y[page],
           largeur,
           splookp->hauteur,
           splookp->bmskp + (x % 4) * splookp->msklen
          );
}

/*****************************************************************
*  GetSpriteBg: Lit le fond du sprite et le mémorise à l'emplacement *
*               prévu                                            *
******************************************************************
*  Entrée  : SPIDP = Pointe sur la structure du sprite           *
*            PAGE = Page d'où est tiré le fond                   *
*                   (0 ou 1)                                     *
*****************************************************************/

void GetSpriteBg( register SPID *spidp, BYTE page )
{
 SPLOOK *splookp;                   /* Pointeur sur l'image du sprite */

 splookp = spidp->splookp;
 blockmove(
           page,
           spidp->x[page],
```

```
        spidp->y[page],
        spidp->fondpage,
        spidp->fondx + ( splookp->largeur * page ),
        spidp->fondy,
        splookp->largeur,
        splookp->hauteur,
        NOBITMASK
       );
}

/*********************************************************************
*  RestoreSpriteBg: Rétablit dans la page d'origine le fond d'un sprite*
*                   sauvegardé au préalable                           *
**-----------------------------------------------------------------**
*  Entrée  : SPIDP = Pointeur sur la structure du sprite             *
*            PAGE = Page où doit être recopié le fond                *
*                   (0 ou 1)                                         *
*********************************************************************/

void RestoreSpriteBg( register SPID *spidp, BYTE page )
{
 SPLOOK *splookp;                    /* Pointeur sur l'image du sprite */

 splookp = spidp->splookp;
 blockmove(
        spidp->fondpage,
        spidp->fondx + ( splookp->largeur * page ),
        spidp->fondy,
        page,
        spidp->x[page], spidp->y[page],
        splookp->largeur,
        splookp->hauteur,
        NOBITMASK
       );
}

/*********************************************************************
*  MoveSprite: Déplace un sprite                                     *
**-----------------------------------------------------------------**
*  Entrée  : SPIDP  = Pointeur sur la structure du sprite            *
*            PAGE  = Page où doit être rétabli le fond (0 ou 1)      *
*            DELTAX = Pas de déplacement dans les directions X et Y  *
*            DELTAY                                                  *
*  Sortie  : Marquage de collision, cf   constantes OUT_...          *
*********************************************************************/

BYTE MoveSprite( SPID *spidp, BYTE page, int deltax, int deltay )
{
 int  nouvx, nouvy;                  /* Nouvelles coordonnées du sprite */
 BYTE out;                          /* Indique une collision avec le bord */

/*-- Décale l'abscisse X et teste s'il y a collision avec le bord-----*/

 if( ( nouvx = spidp->x[page] + deltax ) < 0 )
 {
   nouvx = 0 - deltax - spidp->x[page];
   out = OUT_LEFT;
 }
 else
   if( nouvx > 319 - spidp->splookp->largeur )
   {
     nouvx = 640-nouvx-2*(spidp->splookp->largeur);
     out = OUT_RIGHT;
   }
   else
     out = OUT_NO;

/*-- Décale l'ordonnée Y et teste s'il y a collision avec le bord----*/

 if( ( nouvy = spidp->y[page] + deltay ) < 0 )    /* bord supérieur ? */
 {                                          /* Oui, doit être négatif */
   nouvy = 0 - deltay - spidp->y[page];
   out |= OUT_TOP;
 }
 else
   if( nouvy + spidp->splookp->hauteur > 199+1 )       /* Bord inf ? */
   {                                  /* Oui, deltay doit être positif */
     nouvy = 400-nouvy-2*(spidp->splookp->hauteur);
     out |= OUT_BOTTOM;
   }

/*-- Fixe une nouvelle pos que si différente de l'ancienne ---*/

 if( nouvx != spidp->x[page]  ||  nouvy != spidp->y[page] )
 {                                              /* Nouvelle position */
   RestoreSpriteBg( spidp, page );               /* Rétablit le fond */
   spidp->x[page] = nouvx;          /* Prend les nouvelles coordonnées */
   spidp->y[page] = nouvy;
   GetSpriteBg( spidp, page );                 /* Lit le nouveau fond */
   PrintSprite( spidp, page );   /* Dessine sprite dans page indiquée */
 }
 return out;
}

/*********************************************************************
*  SetSprite: Place le sprite à une position donnée                  *
**-----------------------------------------------------------------**
*  Entrée  : SPIDP = Pointeur sur la structure du sprite             *
*            x0, y0 = Coordonnées du sprite en page 0                *
*            x1, y1 = Coordonnées du sprite en page 1                *
*  Info    : Cette fonction doit être déclenchée avant le premier    *
*            appel à MoveSprite                                      *
*********************************************************************/

void SetSprite( SPID *spidp, int x0, int y0, int x1, int y1 )
{
 spidp->x[0] = x0;      /* Mémorise les coordonnées dans la structure */
 spidp->x[1] = x1;
 spidp->y[0] = y0;
 spidp->y[1] = y1;

 GetSpriteBg( spidp, 0 );             /* Lit le fond du sprite en page */
 GetSpriteBg( spidp, 1 );                                    /* 0 et 1 */
 PrintSprite( spidp, 0 );              /* Dessine le sprite en page 0 */
 PrintSprite( spidp, 1 );                                    /* et 1 */
}

/*********************************************************************
*  RemoveSprite: Retire un sprite de l'emplacement qu'il occupe      *
*                et le rend invisible                                *
**-----------------------------------------------------------------**
*  Entrée  : SPIDP = Pointeur sur la structure du sprite             *
*  Info    : A l'issue de cette fonction il faut appeler la fonction *
*            SetSprite() avant de déplacer le sprite                 *
*            par MoveSprite()                                        *
*********************************************************************/

void RemoveSprite( SPID *spidp )
{
 RestoreSpriteBg( spidp, 0 );         /* Rétablit le fond du sprite en */
 RestoreSpriteBg( spidp, 1 );                              /* pages 0 et 1 */
}

/*********************************************************************
*  Demo: Démontre l'usage des différentes fonctions de ce module     *
*********************************************************************/

void Demo( void )
{
 static char *VaisseauMontant[20] =
                { "                AA              ",
                  "               AAAA             ",
                  "               AAAA             ",
                  "                AA              ",
                  "               GGBGG            ",
                  "              GBBCCBBG          ",
                  "             GBBBCCBBBG         ",
                  "            GBBBBBBBBBG         ",
                  "            GBBBBBBBBBG         ",
                  "  G       GBBBBBBBBBBBBG       G ",
                  " GCG     GBDBBBBBBBBBBDGG     GCG ",
                  " GCG    GGBBBDBBB   BBBDBBBGG   GCG ",
                  " GCBGGGBBBBBDBB     BBDBBBBBGGGBCG ",
                  " GCBBBBBBBBBBDB     BDBBBBBBBBBBCG ",
                  " BBBBBBBBBBBDB BB BDBBBBBBBBBBBB ",
                  " GGCBBBBBBDBBBBBBBBBBDBBBBBBCG ",
                  "  GGCCBBBDDDDDDDDDDDDDBBBCCG   ",
                  "    GGBBDDDDGGGGGDDDDDDBG      ",
                  "      GDDDDGGG    GGGDDDDG      ",
                  "       DDDD        DDDD       " };

 static char *VaisseauDescendant[20] =
                {
                  "       DDDD        DDDD       ",
                  "      GDDDDGGG    GGGDDDDG      ",
                  "    GGBBDDDDGGGGGDDDDDDBBG      ",
                  "  GGCCBBBDDDDDDDDDDDDDBBBCCG    ",
                  " GGCBBBBBBDBBBBBBBBBBDBBBBBBCG ",
                  " BBBBBBBBBBBDB BB BDBBBBBBBBBBB ",
                  " GCBBBBBBBBBDB     BDBBBBBBBBCG ",
                  " GCBGGGBBBBBDBB    BBDBBBBBGGGBCG ",
                  " GCG    GGBBBDBBB  BBBDBBBGG   GCG ",
                  " GCG      GDDBBBBBBBBBBDGG     GCG ",
                  "  G       GBBBBBBBBBBBBG       G ",
                  "            GBBBBBBBBBG         ",
                  "            GBBBBBBBBBG         ",
                  "             GBBBCCBBG          ",
                  "              GGBGG             ",
                  "               AA               ",
                  "               AAAA             ",
                  "               AAAA             ",
                  "                AA              " };

 #define NBSPR   6                          /* Nombre de sprites */
```

```
#define LARGEUR  38     /* Largeur notice de Copyright en caractères */
#define HAUTEUR  6                              /* Hauteur en lignes */
#define SX       (MAXX-(LARGEUR*8)) / 2              /* Coordonnées */
#define SY       (MAXY-(HAUTEUR*8)) / 2               /* de départ */

struct {                                  /* Décrit les sprites gérés */
        SPID *spidp;                  /* Pointeur sur l'identificateur */
        int  deltax[2],       /* Déplacement horizontal pages 0 et 1 */
             deltay[2];   /* Déplacement vertical pour pages 0 et 1 */
       } sprites[ NBSPR ];
BYTE   page,                              /* Page présentement traitée */
       out;                        /* Mémorise l'indicateur de collision */
int    x, y, i,                           /* Compteurs d'itérations */
       dx, dy;                            /* valeurs de déplacement */
char   lc;
SPLOOK *Vaisseauupp, *Vaisseaudnp;        /* Pointe sur les sprites */

srand( *(long far *) 0x0040006c ); /* Initialise gén nbrs aléatoires */

/*-- Remplit les deux premières pages graphiques avec des caractères -*/

for( page = 0; page < 2; ++ page )
{
  setpage( page );
  for( lc = 0, y = 0; y < 200-8; y += 12 )
    for( x = 0; x < 320-8; x += 8 )
      GrafPrintf( x, y, lc % 255, 255, "%c", lc++ & 127 );

        /*-- Affiche le copyright----------------------------------*/

  Line( SX-1, SY-1, SX+LARGEUR*8, SY-1, 15 );
  Line( SX+LARGEUR*8, SY-1, SX+LARGEUR*8, SY+HAUTEUR*8,15 );
  Line( SX+LARGEUR*8, SY+HAUTEUR*8, SX-1, SY+HAUTEUR*8, 15 );
  Line( SX-1, SY+HAUTEUR*8, SX-1, SY-1, 15 );
  GrafPrintf( SX, SY,    15, 4,
              "                                          " );
  GrafPrintf( SX, SY+8,  15, 4,
              " S3220C (c) 1990, 92  Michael Tischer " );
  GrafPrintf( SX, SY+16, 15, 4,
              "                                          " );
  GrafPrintf( SX, SY+24, 15, 4,
              "   Démonstration de sprites en mode   " );
  GrafPrintf( SX, SY+32, 15, 4,
              "         VGA 320x200 256 couleurs     " );
  GrafPrintf( SX, SY+40, 15, 4,
              "                                          " );
}

      /*-- Construit les motifs binaires des sprites ----------------*/

Vaisseauupp = CompileSprite( VaisseauMontant,   20, 2,  0, 'A', 1 );
Vaisseaudnp = CompileSprite( VaisseauDescendant, 20, 2, 40, 'A', 1 );

/*-- Fabrique les différents sprites  -----------------------------*/

for( i = 0; i < NBSPR ; ++ i)
{
  sprites[ i ].spidp = CreateSprite( Vaisseauupp, 3, ( i % 3 ) * 100,
                                     (i / 3) * 30 );
  do                                /* sélectionne les déplacements */
  {
    dx = 0;
    dy = random(8) - 4;
  }
  while ( dx==0  &&  dy==0 );

  sprites[ i ].deltax[0] = sprites[ i ].deltax[1] = dx * 2;
  sprites[ i ].deltay[0] = sprites[ i ].deltay[1] = dy * 2;

  x = ( 320 / NBSPR * i ) + (320 / NBSPR - 40) / 2 ;
  y = random( 200 - 40 );
  SetSprite( sprites[ i ].spidp, x, y, x - dx, y - dy );
}

/*-- Déplace sprites et fait rebondir aux extrémités de l'écran ---*/

page = 1;                                 /* Commence en page 1 */
while( !kbhit() )       /* Une frappe de touche interrompt la boucle */
{
  showpage( 1 - page );                   /* Affiche l'autre page */

  for( i = 0; i < NBSPR; ++ i)            /* Parcourt les sprites */
  {                          /* déplace les sprites et teste les collisions */
    out = MoveSprite( sprites[i].spidp, page, sprites[i].deltax[page],
                      sprites[i].deltay[page] );
    if( out & OUT_TOP  ||  out & OUT_BOTTOM )          /* Contact ? */
    {           /* Oui change la direction du déplacement et l'image */
      sprites[i].deltay[page] = 0 - sprites[i].deltay[page];
      sprites[i].spidp->splookp = ( out & OUT_TOP ) ? Vaisseaudnp
                                                    : Vaisseauupp;
    }
    if( out & OUT_LEFT  ||  out & OUT_RIGHT )
      sprites[i].deltax[page] = 0 - sprites[i].deltax[page];
  }
  page = (page+1) & 1;              /* Passe de 1 à 0 et vice-versa */
}

/*-------------------------------------------------------------------*/
/*-- Programme principal                                           --*/
/*-------------------------------------------------------------------*/

void main( void )
{
  union REGS  regs;

  if( IsVga() )                           /* A-t-on une carte VGA ? */
  {                                             /* Oui, c'est parti */
    init320200();                    /* Initialise le mode graphique */
    Demo();
    getch();                         /* Attend une frappe de touche */
    regs.x.ax = 0x0003;                    /* Revient au mode texte */
    int86( 0x10, &regs, &regs );
  }
  else
    printf( "S3220C.C - (c) 1990,92 by MICHAEL TISCHER\n\nATTENTION "\
            "Ce programme exige une carte VGA\n\n" );
}
```

Listing : S3220CA.ASM

```
;*************************************************************************
;*                          S 3 2 2 0 C A . A S M                        *
;*-----------------------------------------------------------------------*
;*   Fonction      : contient les routines pour travailler avec les      *
;*                   sprites dans le mode 320*200-256 de la carte VGA    *
;*-----------------------------------------------------------------------*
;*   Auteur        : MICHAEL TISCHER                                     *
;*   Développé le  : 8.09.1990                                           *
;*   Dernière MAJ  : 14.02.1992                                          *
;*-----------------------------------------------------------------------*
;*   Modèle mémoire : SMALL                                              *
;*-----------------------------------------------------------------------*
;*   Assemblage    : MASM /mx S3220CA;     ou     TASM -mx S3220CA       *
;*                   ... puis lier à S3220C.C                            *
;*************************************************************************

IGROUP group _text                  ;Regroupe les segments de programme
DGROUP group _bss,  _data           ;Regroupe les segments de données
       assume CS:IGROUP, DS:DGROUP, ES:DGROUP, SS:DGROUP

_BSS   segment word public 'BSS'        ;Segment des variables statiques
_BSS   ends                                          ;non initialisées

_DATA  segment word public 'DATA'       ;Segment des variables globales
                                        ;et statiques non initialisées
_DATA  ends

;-- Constantes ---------------------------------------------------------

SC_INDEX       = 3c4h      ;Registre d'index du contrôleur du séquenceur
SC_MAP_MASK    = 2                    ;Numéro du registre Map Mask
SC_MEM_MODE    = 4               ;Numéro du registre de mode mémoire
GC_INDEX       = 3ceh          ;Registre d'index du contrôleur graphique
GC_GRAPH_MODE  = 5               ;Numéro du registre de mode graphique

VERT_RETRACE   = 3DAh                  ;Registre d'état d'entrée #1
PIXX           = 320                     ;Résolution horizontale

;-- Programme ----------------------------------------------------------

_TEXT  segment byte public 'CODE'                ;Segment de programme

;-- Déclarations publiques ---------------------------------------------
```

```
;
public   _blockmove
;
;:-- BLOCKMOVE: Déplace un groupe de pixels dans la mémoire d'écran-----
;:-- Déclaration : void blockmove( unsigned char depage, int dex,
;:--                               int dey, unsigned char verspage,
;:--                               int versx, int versy, BYTE largeur,
;:--                               byte hauteur, byte *bmskp );
;
_blockmove proc near
;
sframe4  struc                              ;Structure d'accès à la pile
 bp4     dw ?                                       ;mémorise BP
 additif dw ?                                ;variable locale
 restz   dw ?
 movec   dw ?
 dataseg dw ?
 ret_adr4 dw ?                         ;Adresse de retour à l'appelant
 depage  dw ?                                ;Page d'origine
 dex     dw ?                               ;Abscisse d'origine
 dey     dw ?                              ;Ordonnées d'origine
 verspage dw ?                              ;Page de destination
 versx   dw ?                          ;Abscisse X de destination
 versy   dw ?                          ;Ordonnée Y de destination
 largeur dw ?                                    ;Largeur
 hauteur dw ?                                    ;Hauteur
 bmskp   dw ?              ;Pointe sur le buffer avec le masque binaire
sframe4  ends                                ;Fin de la structure
;
frame  equ [ bp - bp4 ]              ;adresse les éléments de la structure
;
       sub   sp,8                     ;8 octets pour les variables locales
;
       push  bp                        ;Prépare l'adressage des paramètres
       mov   bp,sp                                ; par le registre BP
;
       push  ds
       push  si
       push  di
;
       mov   frame.dataseg,ds
;
       mov   dx,GC_INDEX                    ;Lit le mode Write courant et
       mov   al,GC_GRAPH_MODE                     ; fixe le mode Write 1
       out   dx,al
       inc   dx
       in    al,dx
       push  ax                                  ;Empile le mode courant
       and   al,not 3
       or    al,1
       out   dx,al
;
       mov   al,4       ;DS va pointer sur le début de la page d'origine
       mov   cl,byte ptr frame.depage
       mul   cl
       or    al,0A0h
       xchg  ah,al
       mov   ds,ax
;
       mov   al,4 ;ES va pointeur sur le début de la page de destination
       mov   cl,byte ptr frame.verspage
       mul   cl
       or    al,0A0h
       xchg  ah,al
       mov   es,ax
;
       mov   ax,PIXX / 4       ;SI va pointer sur la position d'origine
       mul   frame.dey
       mov   si,frame.dex
       shr   si,1
       shr   si,1
       add   si,ax
;
       mov   ax,PIXX / 4  ;DI va pointer sur la position de destination
       mul   frame.versy
       mov   di,frame.versx
       shr   di,1
       shr   di,1
       add   di,ax
;
       mov   dh,byte ptr frame.hauteur                    ;DH = Lignes
       mov   dl,byte ptr frame.largeur                    ;DL = octets
       shr   dl,1
       shr   dl,1
;
       mov   bx,PIXX / 4                        ;BX = offset ligne suivante
       sub   bl,dl
       xor   ch,ch                    ;Octet haut du compteur toujours nul
       cmp   frame.bmskp,0                    ;Pas de fond à respecter ?
       jne   mt2                             ;Si, utilise une autre routine
;
       push  dx                                      ;Sauvegarde DX sur la pile
       mov   dx,SC_INDEX          ;Assure l'accès à tous les plans de bits
       mov   ah,0Fh
       mov   al,SC_MAP_MASK
       out   dx,ax
       pop   dx                                             ;Reprend DX
;
       ;-- Routine de copie pour tous les quatre plans de bits. --------
       ;-- ne respecte pas le fond ---------------------------------------
;
mt1:   mov   cl,dl                                  ;Nombre d'octets en CL
;
       rep   movsb                                        ;Copie une ligne
       add   di,bx                                 ;DI sur ligne suivante
       add   si,bx                                          ;ainsi que SI
       dec   dh                                   ;Reste-t-il une ligne ?
       jne   mt1                                     ;Oui---> on continue
       jmp   short mtend                          ;Non, prépare la sortie
;
       ;-- Routine de copie pour plans de bits individuels avec --------
       ;-- exploitation du tableau de masques binaires transmis --------
;
mt2:   mov   byte ptr frame.restz,dh     ;Mémorise d'abord les variables
       mov   byte ptr frame.movec,dl      ;qui sont stockées sur la pile
       mov   frame.additif,bx                        ;sous forme locale
;
       mov   al,SC_MAP_MASK      ;Adresse le registre MAP MASK permanent
       mov   dx,SC_INDEX
       out   dx,al
       inc   dx                          ;DX pointe sur le registre de donnéesx
;
       mov   bx,frame.bmskp              ;BX = tableau de masques binaires
       push  ds
       mov   ds,frame.dataseg
       mov   al,[bx]                                ;Charge le premier octet
       xor   ah,ah                          ;Commence avec un octet pair
       pop   ds
;
mt3:   mov   cl,byte ptr frame.movec                ;Nombre d'octets en CL
;
mt4:   out   dx,al                                  ;Fixe le masque binaire
       movsb                                              ;Copie 4 octets
;
       inc   ah                            ;Incrémente le compteur pair/impair
       test  ah,1                                        ;Valeur impaire ?
       jne   mt5                                    ;Oui, déplace le quartet
;
       ;-- Octet pair, passe à l'octet suivant du buffer ---------------
;
       inc   bx                           ;BX sur le prochain octet de masque
       push  ds
       mov   ds,frame.dataseg
       mov   al,[bx]                                ;Charge l'octet suivant
       pop   ds
       loop  mt4                          ;Passe aux quatre latches suivants
       jmp   short mt6
;
mt5:   shr   al,1                           ;Transfère dans le quartet faible
       shr   al,1                         ;le masque binaire de l'octet impair
       shr   al,1
       shr   al,1
       loop  mt4                            ;Pass aux quatre latches suivants
;
mt6:   add   di,frame.additif                       ;DI sur ligne suivante
       add   si,frame.additif                             ; ainsi que SI
       dec   byte ptr frame.restz                     ;Reste une ligne ?
       jne   mt3                                     ;Oui---> on continue
;
mtend: mov   dx,GC_INDEX                   ;Rétablit l'ancien mode Write
       pop   ax
       mov   ah,al
       mov   al,GC_GRAPH_MODE
       out   dx,ax
;
       pop   di
       pop   si
       pop   ds
       pop   bp
;
       add   sp,8                              ;Supprime les variables locales
       ret
;
_blockmove endp
;
;:-- Fin ---------------------------------------------------------------
;
_text    ends                                    ;Fin du segment de programme
         end                                     ;Fin du source en assembleur
;
```

Listing : V3220CA.ASM (déjà mentionné en 4.8.7)

Listing : S3220P.PAS

```
{*********************************************************************
*                          S3220P.PAS                                *
**--------------------------------------------------------------------**
*  Fonction : Montre comment travailler avec des sprites dans le mode *
*             graphique VGA 320*200 en 256 couleurs avec quatre pages *
*             d'écran. Le programme utilise les routines en assembleur *
*             des modules V3220PA.ASM et S3220PA.ASM                  *
**--------------------------------------------------------------------**
*  Auteur       : MICHAEL TISCHER                                     *
*  Développé le : 12.09.1990                                          *
*  Dernière MAJ : 14.01.1991                                          *
*********************************************************************}

program S3220P;

uses dos, crt;

{-- Références externes aux routines en assembleur ------------------}

{$L v3220pa}  { Intègre un module en assembleur }

procedure init320200; external;
procedure setpix( x, y : integer; couleur : byte ); external;
function  getpix( x, y: integer ) : byte : external;
procedure setpage( page : byte ); external;
procedure showpage( page : byte ); external;

{$L s3220pa}    { Intègre un module en assembleur }

procedure blockmove( depage : byte; dex, dey : integer;
                     apage : byte; ax, ay : integer;
                     largeur, hauteur: byte; bmskp : pointer ); external;

{-- Constantes ------------------------------------------------------}

const MAXX = 319;                                  { Coordonnées maximales }
      MAXY = 199;

      OUT_LEFT   = 1;     { Indicateurs de collision pour SpriteMove() }
      OUT_TOP    = 2;
      OUT_RIGHT  = 4;
      OUT_BOTTOM = 8;
      OUT_NO     = 0;                                          { no problem }

{-- Déclarations de types -------------------------------------------}

type SPLOOK = record                                  { Image d'un sprite }
                largeur,                                 { Largeur totale }
                hauteur,                        { Hauteur en lignes de pixels }
                page,                             { Page de mémorisation ... }
                msklen : byte;                       { Longueur d'une entrée }
                bmskp  : pointer;      { Pointeur sur le masque binaire }
                ligne  : integer;     { Ligne de mémorisation du sprite }
              end;                                              { dans sa page }
     SPLP = ^SPLOOK;                                { Pointeur sur l'image }

     SPID = record                              { Descripteur de sprite (ID) }
              fondpage :byte;                             { Page de fond }
              x, y : array [0..1] of integer;     { Coord. pages 0 et 1 }
              fondx, fondy : integer;             { Buffer pour le fond }
              splookp : SPLP;                    { Pointeur sur l'image }
            end;
     SPIP = ^SPID;                                 { Pointeur sur descripteur }

     BYTEAR = array[0..10000] of byte;  { Pour adresser les différents }
     BARPTR = ^BYTEAR;                                          { buffers }

     PTRREC = record       { Pour décomposer un pointeur ou un LONGINT }
                ofs,
                seg : word;
              end;

{*********************************************************************
*  IsVga : Teste la présence d'une carte VGA.                        *
**--------------------------------------------------------------------**
*  Entrée  : néant                                                   *
*  Sortie  : TRUE ou FALSE                                           *
*********************************************************************}

function IsVga : boolean;

var Regs : Registers;                    { Registres pour l'interruption }

begin
  Regs.AX := $1a00;                  { La fonction 1Ah n'existe qu'en VGA }
  Intr( $10, Regs );
  IsVga := ( Regs.AL = $1a );
end;

{*********************************************************************
*  PrintChar : Affiche un caractère en mode graphique                *
**--------------------------------------------------------------------**
*  Entrée : caractere = le caractère à afficher                      *
*           x, y      = Coordonnées du coin sup gauche               *
*           cc        = Couleur du caractère                         *
*           cf        = Couleur du fond                              *
*  Info   : Le caractère est dessiné dans une matrice de 8*8 pixels  *
*           sur la base du jeu de caractères 8*8 en ROM               *
*********************************************************************}

procedure PrintChar( caractere : char; x, y : integer; cc, cf : byte );

type CARADEF = array[0..255,0..7] of byte; {Structure jeu de caractères}
     CARAPTR = ^CARADEF;                  { Pointe sur le jeu de caractères }

var  Regs  : Registers;          { Registres pour gérer les interruptions }
     ch    : char;                                    { Pixel du caractère }
     i, k,                                        { Compteur d'itérations }
     Masque : byte;           { Masque binaire pour dessiner le caractère}

const fptr : CARAPTR = NIL;   { Pointe sur le jeu de caractères en ROM }

begin
  if fptr = NIL then              { A-t-on déjà déterminé ce pointeur ? }
    begin                                                          { Non}
      Regs.AH := $11;                        { Appelle l'option $1130 de la }
      Regs.AL := $30;                                { fonction vidéo du BIOS }
      Regs.BH := 3;                { pour obtenir un pointeur sur le jeu 8*8 }
      Intr( $10, Regs );
      fptr := ptr( Regs.ES, Regs.BP );                { Compose le pointeur }
    end;

  if ( cf = 255 ) then                          { Caractère transparent ? }
    for i := 0 to 7 do  { Oui ne dessine que les pixels du premier plan}
      begin
        Masque := fptr^[ord(caractere),i];{Motif binaire pour une ligne}
        for k := 0 to 7 do
          begin
            if ( Masque and 128 <> 0 ) then          { Pixel à dessiner? }
              setpix( x+k, y+i, cc );                                { Oui }
            Masque := Masque shl 1;
          end;
      end
  else                                        { Non, tient compte du fond }
    for i := 0 to 7 do                          { Parcourt les lignes }
      begin
        Masque := fptr^[ord(caractere),i];{Motif binaire pour une ligne}
        for k := 0 to 7 do
          begin
            if ( Masque and 128 <> 0 ) then          { Premier plan ? }
              setpix( x+k, y+i, cc )                                 { Oui }
            else
              setpix( x+k, y+i, cf );                          { Non, fond}
            Masque := Masque shl 1;
          end;
      end;
end;

{*********************************************************************
*  Line : Trace un segment dans la fenêtre graphique en appliquant   *
*         l'algorithme de Bresenham                                  *
**--------------------------------------------------------------------**
```

```
{*  Entrée : X1, Y1 = Coordonnées de l'origine (0 - ...)          *
{*           X2, Y2 = Coordonnées de l'extrémité terminale        *
{*           COULEUR = couleur du segment                         *
{*****************************************************************}

procedure Line( x1, y1, x2, y2 : integer; couleur : byte );

var d, dx, dy,
    aincr, bincr,
    xincr, yincr,
    x, y                : integer;

{-- Procédure accessoire pour échanger deux variables entières --------}

procedure SwapInt( var i1, i2: integer );

var dummy : integer;

begin
  dummy := i2;
  i2    := i1;
  i1    := dummy;
end;

{-- Procédure principale ----------------------------------------------}

begin
  if ( abs(x2-x1) < abs(y2-y1) ) then    { Parcours : par axe X ou Y ? }
    begin                                          { par l'axe des Y }
      if ( y1 > y2 ) then                        { y1 supérieur à y2 ? }
        begin
      SwapInt( x1, x2 );                      { Oui, échange X1 et X2, }
      SwapInt( y1, y2 );                                  { Y1 et Y2 }
        end;

      if ( x2 > x1 ) then xincr := 1          { Fixe le pas horizontal }
                     else xincr := -1;

      dy := y2 - y1;
      dx := abs( x2-x1 );
      d  := 2 * dx - dy;
      aincr := 2 * (dx - dy);
      bincr := 2 * dx;
      x := x1;
      y := y1;

      Setpix( x, y, couleur );                { Dessine le premier point }
      for y:=y1+1 to y2 do                      { Parcourt l'axe des Y }
        begin
          if ( d >= 0 ) then
            begin
              inc( x, xincr );
              inc( d, aincr );
            end
          else
            inc( d, bincr );
          Setpix( x, y, couleur );
        end;
    end
  else                                                  { par l'axe des X }
    begin
      if ( x1 > x2 ) then                     { x1 plus grand que x2 ? }
        begin
      SwapInt( x1, x2 );                       { Oui, échange X1 et X2 }
      SwapInt( y1, y2 );                                  { Y1 et Y2 }
        end;

      if ( y2 > y1 ) then yincr := 1          { Fixe le pas vertical }
                     else yincr := -1;

      dx := x2 - x1;
      dy := abs( y2-y1 );
      d  := 2 * dy - dx;
      aincr := 2 * (dy - dx);
      bincr := 2 * dy;
      x := x1;
      y := y1;

      Setpix( x, y, couleur );                { Dessine le premier point }
      for x:=x1+1 to x2 do                      { Parcourt l'axe des X }
        begin
          if ( d >= 0 ) then
            begin
              inc( y, yincr );
              inc( d, aincr );
            end
          else
            inc( d, bincr );
          Setpix( x, y, couleur );
        end;
    end;
end;
```

```
{*****************************************************************}
{*  GrafPrint : Affiche une chaîne formatée sur l'écran graphique *
{**-------------------------------------------------------------**}
{*  Entrées : X, Y  = Coordonnées de départ (0-...)              *
{*            CC    = Couleur des caractères                     *
{*            CF    = Couleur du fond (256 = transparent)        *
{*            STRING = Chaîne avec indications de formatage      *
{*****************************************************************}

procedure GrafPrint( x, y : integer; cc, cf : byte; strt : string );

var i : integer;                                { Compteur d'itérations }

begin
  for i:=1 to length( strt ) do
    begin
      printchar( strt[i], x, y, cc, cf );       { Affiche par printchar }
      inc( x, 8 );                  { x à la position du caractère suivant }
    end;
end;

{*****************************************************************}
{*  CreateSprite : Crée un sprite à l'aide d'un motif de pixels  *
{*                 préalablement compilé                         *
{**-------------------------------------------------------------**}
{*  Entrées : SPLOOKP  = Pointeur sur la structure de données produite *
{*                       par CompileSprite                       *
{*            FONDPAGE = Page d'écran qui doit mémoriser le fond du *
{*                       sprite                                  *
{*            FONDX,   = Coordonnées dans la page de fond où doit être *
{*            FONDY      mémorisé le fond du sprite             *
{*  Sortie  : Pointeur sur la structure du sprite créée          *
{*  Info    : La mémorisation du fond du sprite nécessite deux zones *
{*            contiguës de la taille du sprite.                  *
{*****************************************************************}

function CreateSprite( splookp : SPLP; fondpage : byte;
                       fondx, fondy : integer              ) : SPIP;

var spidp : SPIP;                          { Pointeur sur la structure créée}

begin
  new( spidp );                  { Alloue de la mémoire pour la structure }
  spidp^.splookp := splookp;                      { Y reporte les données }
  spidp^.fondpage := fondpage;
  spidp^.fondx := fondx;
  spidp^.fondy := fondy;

  CreateSprite := spidp;         { Retourne un pointeur sur la structure }
end;

{*****************************************************************}
{*  CompileSprite : Crée le motif binaire d'un sprite à l'aide d'une *
{*                  définition connue au moment de l'exécution    *
{**-------------------------------------------------------------**}
{*  Entrée : BUFP      = Pointeur sur un tableau de pointeurs    *
{*                       référençant des chaînes de caractères qui *
{*                       représentent le motif du sprite         *
{*           HAUTEUR   = Hauteur sprite et nb de chaînes de caractères *
{*           PAGE      = Page graphique pour dessiner le sprite  *
{*           Y         = Ligne où commence le dessin             *
{*           CLR       = Caractère ASCII associé à plus petite couleur *
{*           COULEURPP = Premier code de couleur pour CLR        *
{*  Info   : Les sprites sont dessinés en partant du bord gauche de la *
{*           ligne indiquée                                      *
{*****************************************************************}

function CompileSprite( var buf; hauteur, page : byte;
                        y : integer; clr : char; couleurpp : byte ) : SPLP;

type BYPTR  = ^byte;                            { Pointeur sur un octet }

var  largeur,                  { Longueur des chaînes = largeur du motif }
     c,                                    { mémorise un caractère }
     couleur,                                { Couleur d'un pixel }
     i, k, l,                              { Compteurs d'itérations }
     pixc,       { Compteurs de pixels pour compiler le masque binaire }
     pixm     : byte;                              { Masque de pixels }
     distance,                          { Distance entre les sprites }
     lx, ly   : integer;                  { Coordonnées courantes }
     splookp  : SPLP;                  { Pointeur sur la structure créée }
     lspb     : BYPTR;   { Pointeur courant dans le buffer des sprites }
     bptr     : barptr;          {Pour adresser le buffer avec l'image }

begin
        {-- Crée un structure Sprite Look et la remplit -------------}

  new( splookp );
  bptr := @buf;                          { Pointeur sur le buffer du logo }
  largeur := bptr^[0];       { Longueur des chaînes ou largeur du logo }
  distance := ( ( largeur + 3 + 3 ) div 4 ) * 4;
```

```
  splookp^.largeur := distance;
  splookp^.msklen := (distance*hauteur+7) div 8;
  getmem( splookp^.bmskp, splookp^.msklen * 4 );
  splookp^.hauteur  := hauteur;
  splookp^.ligne  := y;
  splookp^.page  := page;

{-- Replit le fond du sprite dans sa page d'origine avec les codes --}
                        {-- pour le fond de caractères transparent --}

  setpage( page );                                  { Fixe la page de dessin }
  lx := 4 * distance - 1;
  for ly:=y+hauteur-1 downto y do
    Line( 0, ly, lx, ly, 255 );

          {-- Dessine quatre fois le sprite dans sa page d'origine --}

  ix := 0;                                          { Commence au bord gauche }
  for l := 1 to 4 do                                  { Dessine quatre fois }
    begin
      for j := 0 to hauteur-1 do                      { Parcourt les lignes }
        for k := 0 to largeur-1 do                    { Parcourt les colonnes }
          begin
            c := bptr^[j*(largeur+1)+k+1];                 { Lit la couleur }
            if ( c = 32 ) then                          { Pixel de fond ? }
              setpix( lx+k, y+j, 255 ){ Oui met le code de couleur 255 }
            else                    { Non met le code de couleur indiqué }
              setpix( lx+k, y+j, couleurpp+(c-ord(clr)) );
          end;
      inc( lx, distance+1 );                          { Colonne suivante }
    end;

{ Parcourt les quatre sprites dessinés et génère les masques binaires }
{-- pour copier les sprites dans les plans de bits --}

  pixm := 0;
  pixc := 0;
  lx := 0;
  for l := 0 to 3 do
    begin
      lspb := splookp^.bmskp;
      inc( PTRREC( lspb ).ofs, splookp^.msklen * l );

      for j := 0 to hauteur-1 do
        for k := 0 to distance-1 do
          begin
            pixm := pixm shr 1;{ Décale masque d'un bit vers la droite }
            if ( getpix( lx+k, y+j ) <> 255 ) then    {Pixel de fond ? }
              pixm := pixm or 128;          { Non, fixe un bit du masque }
            inc( pixc, 1 );
            if ( pixc = 8 ) then        { A-t-on déjà traité huit pixels }
              begin{ Oui mémorise le masque dans le buffer des sprites }
                lspb^ := pixm;
                inc( PTRREC( lspb ).ofs, 1 );
                pixc := 0; { Remet à 0 le compteur de pixels et masque }
                pixm := 0;
              end;
          end;

      if ( pixc <> 0 ) then          { Dernier quartet dans le buffer ? }
        begin                                                   { Non}
          lspb^ := pixm shr 4;       { Quartet haut dans le quartet bas }
          pixc := 0; { Réinitialise le compteur de pixels et le masque }
          pixm := 0;
        end;

      inc( lx, distance ); { LX pointe sur le début du sprite suivant }
    end;

  CompileSprite := splookp; { Renvoie un ptr sur le buffer des sprites }
end;

{*********************************************************************
*  PrintSprite : Affiche un sprite dans une page donnée              *
**--------------------------------------------------------------------**
*  Entrées : SPIDP = Pointeur sur la structure du sprite             *
*            PAGE  = Page concernée (0 ou 1)                         *
*********************************************************************}

procedure PrintSprite( spidp : SPIP; page : byte );

var largeur  : byte;                          { Largeur totale du sprite }
    x        : integer;                 { Abscisse X du sprite dans sa page }
    splookp : SPLP;                     { Pointeur sur l'image du sprite }

begin
  splookp := spidp^.splookp;
  largeur  := splookp^.largeur;
  x        := spidp^.x[page];
  blockmove( splookp^.page, largeur * (x mod 4), splookp^.ligne, page,
             x and not(3), spidp^.y[page], largeur, splookp^.hauteur,
             @BARPTR(splookp^.bmskp)^[(x mod 4) * splookp^.msklen] );
end;

{*********************************************************************
*  GetSpriteBg : Lit le fond du sprite et le mémorise à l'emplacement *
*                prévu                                               *
**--------------------------------------------------------------------**
*  Entrée : SPIDP = Pointeur sur la structure du sprite              *
*           PAGE  = Page d'où est tiré le fond (0 ou 1)               *
*********************************************************************}

procedure GetSpriteBg( spidp : SPIP; page : BYTE );

var splookp : SPLP;                     { Pointeur sur l'image du sprite }

begin
  splookp := spidp^.splookp;
  blockmove( page, spidp^.x[page], spidp^.y[page],
             spidp^.fondpage, spidp^.fondx + ( splookp^.largeur * page ),
             spidp^.fondy, splookp^.largeur, splookp^.hauteur, NIL );
end;

{*********************************************************************
*  RestoreSpriteBg : Rétablit dans la page d'origine le fond d'un     *
*                    sprite sauvegardé au préalable                  *
**--------------------------------------------------------------------**
*  Entrées : SPIDP = Pointeur sur la structure du sprite             *
*            PAGE  = Page où doit être recopié le fond (0 ou 1)      *
*********************************************************************}

procedure RestoreSpriteBg( spidp : SPIP; page : BYTE );

var splookp : SPLP;                     { Pointeur sur l'image du sprite }

begin
  splookp := spidp^.splookp;
  blockmove( spidp^.fondpage, spidp^.fondx + ( splookp^.largeur * page ),
             spidp^.fondy, page, spidp^.x[page], spidp^.y[page],
             splookp^.largeur, splookp^.hauteur, NIL );
end;

{*********************************************************************
*  MoveSprite : Déplace un sprite                                    *
**--------------------------------------------------------------------**
*  Entrée : SPIDP  = Pointeur sur la structure du sprite             *
*           PAGE   = Page où doit être rétabli le fond (0 ou 1)      *
*           DELTAX = Déplacement dans les directions X et Y          *
*           DELTAY                                                   *
*  Sortie : Indicateurs de collision cf constantes OUT_...          *
*********************************************************************}

function MoveSprite( spidp : SPIP; page : byte;
                     deltax, deltay : integer    ) : byte;

var nouvx, nouvy : integer;           { nouvelles coordonnées du sprite }
    out          : byte;            { Indique les collisions avec le bord }

begin
    {-- Décale l'abscisse X et teste s'il y a collision avec le bord--}

  nouvx := spidp^.x[page] + deltax;
  if ( nouvx  < 0 ) then
    begin
      nouvx := 0 - deltax - spidp^.x[page];
      out := OUT_LEFT;
    end
  else
    if ( nouvx > MAXX - spidp^.splookp^.largeur ) then
      begin
        nouvx := (2*(MAXX+1))-nouvx-2*(spidp^.splookp^.largeur);
        out := OUT_RIGHT;
      end
    else
      out := OUT_NO;

    {-- Décale l'ordonnée Y et teste s'il y a collision avec le bord --}

  nouvy := spidp^.y[page] + deltay;                                 { }
  if ( nouvy < 0 ) then                                  { Bord sup ? }
    begin                                  { Oui deltay doit être négatif }
      nouvy := 0 - deltay - spidp^.y[page];
      out := out or OUT_TOP;
    end
  else
    if ( nouvy + spidp^.splookp^.hauteur > MAXY+1  ) then  { bord inf ? }
      begin                                  { Oui deltay doit être positif }
        nouvy := (2*(MAXY+1))-nouvy-2*(spidp^.splookp^.hauteur);
        out := out or OUT_BOTTOM;
      end;

{Ne fixe une nouvelle position que si elle est différente de l'ancienne}

  if ( nouvx <> spidp^.x[page] ) or  ( nouvy <> spidp^.y[page] ) then
    begin                                          { Nouvelle position }
      RestoreSpriteBg( spidp, page );                { Rétablit le fond }
```

```
    spidp^.x[page] := nouvx;          { Prend les nouvelles coordonnées }
    spidp^.y[page] := nouvy;
    GetSpriteBg( spidp, page );                { Lit le nouveau fond }
    PrintSprite( spidp, page );       { Dessine le sprite dans la page }
  end;

 MoveSprite := out;
end;

{**************************************************************************
 * SetSprite : Place le sprite à une position donnée                      *
 *------------------------------------------------------------------------*
 * Entrée : SPIDP  = Pointeur sur la structure du sprite                  *
 *          x0, y0 = Coordonnées du sprite en page 0                      *
 *          x1, y1 = Coordonnées du sprite en page 1                      *
 * Info   : Cette fonction doit être déclenchée avant le premier          *
 *          appel à MoveSprite()                                          *
 **************************************************************************}

procedure SetSprite( spidp : SPIP; x0, y0, x1, y1 : integer );

begin
 spidp^.x[0] := x0;       { Mémorise les coordonnées dans la structure }
 spidp^.x[1] := x1;
 spidp^.y[0] := y0;
 spidp^.y[1] := y1;

 GetSpriteBg( spidp, 0 );                { Lit le fond du sprite en page }
 GetSpriteBg( spidp, 1 );                                      { 0 et 1 }
 PrintSprite( spidp, 0 );                 { Dessine le sprite en page 0 }
 PrintSprite( spidp, 1 );                                      { et 1 }
end;

{**************************************************************************
 * RemoveSprite : Retire un sprite de l'emplacement qu'il occupe          *
 *                le rendant ainsi invisible                              *
 *------------------------------------------------------------------------*
 * Entrées : SPIDP = Pointeur sur la structure du sprite                  *
 * Info    : A l'issue de cette fonction il faut appeler la fonction      *
 *           SetSprite() avant de déplacer le sprite par MoveSprite()     *
 **************************************************************************}

procedure RemoveSprite( spidp : SPIP );

begin
 RestoreSpriteBg( spidp, 0 );              { Rétablit le fond du sprite }
 RestoreSpriteBg( spidp, 1 );                                  { en pages 0 et 1}
end;

{**************************************************************************
 * Demo : Démontre l'usage des différentes fonctions de ce module         *
 **************************************************************************}

procedure Demo;

const VaisseauMontant :array [1..20] of string[32] =
                ( '              AA              ',
                  '             AAAA             ',
                  '             AAAA             ',
                  '              AA              ',
                  '            GGBBGG            ',
                  '           GBBCCBBG           ',
                  '          GBBBCCBBBG          ',
                  '         GBGBBBBBBBBG         ',
                  '         GBBBBBBBBBG          ',
                  ' G     GBBBBBBBBBBBBG      G  ',
                  'GCG    GGDBBBBBBBBBDGG    GCG',
                  'GCG  GGBBBDBBB  BBBDBBBGG  GCG',
                  'GCBGGGBBBBBDBB    BBDBBBBBGGGBCG',
                  'GCBBBBBBBBBBDB    BDBBBBBBBBBBCG',
                  'BBBBBBBBBBBDB BB BDBBBBBBBBBBB',
                  'GGCBBBBBBBDBBBBBBBBBDBBBBBBCG ',
                  '  GGCCBBBDDDDDDDDDDDDDBBBCCG  ',
                  '    GGBBBDDDDGGGGGDDDDDBBG    ',
                  '      GDDDDGGG    GGGDDDDG     ',
                  '       DDDD          DDDD      ' );

const VaisseauDescendant :array [1..20] of string[32] =
                ( '       DDDD          DDDD      ',
                  '      GDDDDGGG    GGGDDDDG     ',
                  '    GGBBBDDDDGGGGGDDDDDBBBG    ',
                  '  GGCCBBBDDDDDDDDDDDDDBBBCCG   ',
                  'GGCBBBBBBBDBBBBBBBBBDBBBBBBCG ',
                  'BBBBBBBBBBBDB BB BDBBBBBBBBBBB',
                  'GCBBBBBBBBBDB     BDBBBBBBBBBCG',
                  'GCBGGGBBBBBDBB    BBDBBBBBGGGBCG',
                  'GCG  GGBBBDBBB  BBBDBBBGG  GCG',
                  'GCG    GGDBBBBBBBBBDGG    GCG',
                  ' G     GBBBBBBBBBBBBG      G  ',
                  '         GBBBBBBBBBG          ',
                  '         GBBBBBBBBBG          ',
                  '          GBBBCCBBBG          ',
                  '           GBBCCBBG           ',
                  '            GGBBGG            ',
                  '             AA               ',
                  '             AAAA             ',
                  '             AAAA             ',
                  '              AA              ' );

 NBSPR = 6;                                    { Nombre de sprites }
 LARGEUR = 37;     { Largeur de la notice de copyright en caractères }
 HAUTEUR = 6;                    { Hauteur en lignes de la notice }
 SX    = (MAXX-(LARGEUR*8)) div 2;          { Coordonnées de }
 SY    = (MAXY-(HAUTEUR*8)) div 2;          { départ }

type SPRITE = record                    { Pour gérer les sprites }
              spidp : SPIP;          { Pointeur sur identificateur }
              deltax,     { Déplacement horizontal pour pages 0 et 1 }
              deltay : array [0..1] of integer;{ Déplacement vertical}
            end;

var sprites    : array [1..NBSPR] of SPRITE;
    page,                                { Page présentement traitée }
    ic,                                  { Caractère courant }
    out        : byte;          { Mémorise les indicateurs de collision }
    x, y, i,                             { Compteurs d'itérations }
    dx, dy     : integer;                { Déplacements }
    Vaisseauupp,
    VaisseaudnpP : SPLP;                 { Pointeur sur les sprites }
    ch         : char;
begin

 Randomize;          { Initialise le générateur de nombres aléatoires }

                 {-- Remplit de caractères les deux premières pages --}

 for page := 0 to 1 do
   begin
     setpage( page );
     ic := 0;
     y := 0;
     while ( y < 200-8 ) do
       begin
         x := 0;
         while ( x < 320-8 ) do
           begin
             PrintChar( chr(ic and 127), x, y, ic mod 255, 0 );
             inc( ic );
             inc( x, 8 );
           end;
         inc( y, 12 );
       end;

     {-- Affiche l'avis de copyright ---------------------------------}

     Line( SX-1, SY-1, SX+LARGEUR*8, SY-1, 15 );
     Line( SX+LARGEUR*8, SY-1, SX+LARGEUR*8, SY+HAUTEUR*8,15 );
     Line( SX+LARGEUR*8, SY+HAUTEUR*8, SX-1, SY+HAUTEUR*8, 15 );
     Line( SX-1, SY+HAUTEUR*8, SX-1, SY-1, 15 );
     GrafPrint( SX, SY,    15, 4,
                                                        ' );
     GrafPrint( SX, SY+8,  15, 4,
                ' S3220P.PAS - (c) 1990 by M. TISCHER ' );
     GrafPrint( SX, SY+16, 15, 4,
                                                        ' );
     GrafPrint( SX, SY+24, 15, 4,
                ' Démonstration de sprite dans le     ' );
     GrafPrint( SX, SY+32, 15, 4,
                ' mode 320*200 de la carte VGA        ' );
     GrafPrint( SX, SY+40, 15, 4,
                                                        ' );
   end;

       {-- Construit les motifs binaires des sprites ----------------}

 Vaisseauupp := CompileSprite( VaisseauMontant,  20, 2, 0, 'A', 1 );
 Vaisseaudnp := CompileSprite( VaisseauDescendant, 20, 2, 40, 'A', 1 );

   {-- Fabrique les différents sprites ------------------------------}

 for i := 1 to NBSPR do
   begin
     sprites[ i ].spidp := CreateSprite( Vaisseauupp, 3, (i mod 3)*100,
                                         (i div 3) * 30 );
     repeat                                { Sélectionne les déplacements }
       dx := 0;
       dy := random(8) - 4;
     until ( dx <> 0 ) or ( dy <> 0 );

     sprites[ i ].deltax[0] := dx * 2;
     sprites[ i ].deltay[0] := dy * 2;
     sprites[ i ].deltax[1] := dx * 2;
     sprites[ i ].deltay[1] := dy * 2;

     x := ( (320 div NBSPR) * (i-1) ) + ((320 div NBSPR) - 40) div 2 ;
     y := random( 200 - 40 );
```

```
    SetSprite( sprites[ i ].spidp, x, y, x - dx, y - dy );
  end;

{ Déplace les sprites et les fait rebondir aux extrémités de l'écran }

  page := 1;                                        { Commence en page 1 }
  while ( not keypressed ) do{Une frappe de touche interrompt la boucle}
    begin
      showpage( 1 - page );                         { Affiche l'autre page }

      for i := 1 to NBSPR do                        { Parcourt les sprites }
        begin                { Déplace les sprites et teste les collisions }
          out := MoveSprite( sprites[i].spidp, page,
                             sprites[i].deltax[page],
                             sprites[i].deltay[page] );
          if ( ( out and OUT_TOP ) <> 0 ) or ( Contact en haut/en bas ?)
             ( ( out and OUT_BOTTOM ) <> 0 ) then
            begin
                    {-- Oui change la direction du déplacement et --}
                    {-- l'image                                   --}
              sprites[i].deltay[page] := -sprites[i].deltay[page];
              if ( ( out and OUT_TOP ) <> 0 ) then
                sprites[i].spidp^.splookp := Vaisseaudnp
              else
                sprites[i].spidp^.splookp := Vaisseauupp;
            end;
          if ( ( out and OUT_LEFT ) <> 0 ) or( Contact droite/gauche ? )
             ( ( out and OUT_RIGHT ) <> 0 ) then
            sprites[i].deltax[page] := -sprites[i].deltax[page];
        end;
      page := (page+1) and 1;           { Passe de 1 à 0 et vice-versa }
    end;
  ch := readkey;                  { Retire la touche du buffer du clavier }
end;

{------------------------------------------------------------------------}
{-- PROGRAMME PRINCIPAL                                                ----}
{------------------------------------------------------------------------}

begin
  if ( IsVga ) then                                  { A-t-on une carte VGA ? }
    begin                                                  { Oui, on y va }
      init320200;                                { Initialise le mode graphique }
      Demo;
      Textmode( COBO );                                { Revient au mode texte }
    end
  else
    writeln( 'S3220P.PAS  -  (c) 1990 by MICHAEL TISCHER'#13#10#10 +
             'Attention ! Ce programme exige une carte VGA ! ' +
             ''#13#10 );

end.
```

Listing : S3220PA.ASM

```
;*****************************************************************;
;*                      S 3 2 2 0 P A . A S M                    *;
;*---------------------------------------------------------------*;
;*   Fonction : contient les routines pour travailler avec les   *;
;*              sprites dans les mode 320*200-256 de la carte VGA*;
;*---------------------------------------------------------------*;
;*   Auteur          : MICHAEL TISCHER                           *;
;*   Développé le    : 08.09.1990                                *;
;*   Dernière MAJ    : 14.01.1991                                *;
;*---------------------------------------------------------------*;
;*   Assemblage      : MASM /mx S3220PA;     ou     TASM -mx S3220PA  *;
;*                     puis lier à   S3220P.PAS                  *;
;*****************************************************************;


;-- Constantes ---------------------------------------------------

SC_INDEX        = 3c4h   ;Registre d'index du contrpoleur du séquenceur .
SC_MAP_MASK     = 2                     ;Numéro du registre Map Mask
SC_MEM_MODE     = 4                  ;Numéro du registre de mode mémoire

GC_INDEX        = 3ceh        ;Registre d'index du contrôleur graphique .
GC_GRAPH_MODE   = 5               ;Numéro de registre de mode graphique

VERT_RETRACE    = 3DAh                      ;Registre d'état d'entrée #1
PIXX            = 320                        ;Résolution horizontale

;-- Segment de données -------------------------------------------

DATA    segment word public

DATA    ends

;-- Programme ----------------------------------------------------

CODE        segment byte public                     ;Segment de programme

        assume cs:code, ds:data

;-- Déclarations publiques ---------------------------------------

public      blockmove

;-----------------------------------------------------------------
;-- BLOCKMOVE: Déplace un groupe de pixels dans la mémoire d'écran
;-- Appel depuis TP: blockmove( depage : byte; dex, dey : integer;
;--                             verspage : byte; versx, versy : integer;
;--                             largeur, hauteur: byte; bmskp : pointer );

blockmove  proc near

sframe4    struc                            ;Structure d'accès à la pile
bp4        dw ?                                          ;Mémorise BP
additif    dw ?                                  ;variable locale
restz      dw ?
movec      dw ?
ret_adr4   dw ?                                ;Adresse de retour à l'appelant
bmskp      dd ?                  ;Pointe sur le buffer avec le masque binaire
hauteur    dw ?                                                ;Hauteur
largeur    dw ?                                                ;Largeur
versy      dw ?                                ;Ordonnée de destination
versx      dw ?                                ;Abscisse de destination
verspage   dw ?                                   ;Page de destination
dey        dw ?                                   ;Ordonnée d'origine
dex        dw ?                                   ;Abscisse d'origine
depage     dw ?                                      ;Page d'origine
sframe4    ends                                     ;Fin de la structure

frame   equ [ bp - bp4 ]              ;adresse les éléments de la structure

        sub   sp,6                     ;6 octets pour les variables locales

        push  bp                        ;Prépare l'adressage des paramètres
        mov   bp,sp                                 ; par le registre BP

        push  ds
        cld                                  ;Fixe le sens des traitements de chaines

        mov   dx,GC_INDEX                          ;Lit le mode Write courant et
        mov   al,GC_GRAPH_MODE                           ; fixe le mode Write 1
        out   dx,al
        inc   dx
        in    al,dx
        push  ax                                              ;Empile le mode courant
        and   al,not 3
        or    al,1
        out   dx,al

        mov   al,4        ;DS va pointer sur le début de la page d'origine
        mov   cl,byte ptr frame.depage
        mul   cl
        or    al,0A0h
        xchg  ah,al
        mov   ds,ax

        mov   al,4  ;ES va pointer sur le début de la page de destination
        mov   cl,byte ptr frame.verspage
        mul   cl
        or    al,0A0h
        xchg  ah,al
        mov   es,ax

        mov   ax,PIXX / 4         ;SI va pointer sur la position d'origine
        mul   frame.dey
        mov   si,frame.dex
        shr   si,1
        shr   si,1
        add   si,ax

        mov   ax,PIXX / 4   ;DI va pointer sur la position de destination
```

```
        mul   frame.versy
        mov   di,frame.versx
        shr   di,1
        shr   di,1
        add   di,ax

        mov   dh,byte ptr frame.hauteur                 ;DH = Lignes
        mov   dl,byte ptr frame.largeur                 ;DL = octets
        shr   dl,1
        shr   dl,1

        mov   bx,PIXX / 4                   ;BX = offset ligne suivante
        sub   bl,dl
        xor   ch,ch                 ;Octet haut du compteur toujours nul
        cmp   word ptr frame.bmskp+2,0        ;Pas de fond à respecter ?
        jne   mt2                           ;Si, utilise une autre routine

        push  dx                                    ;Sauve DX sur la pile
        mov   dx,SC_INDEX         ;Assure l'accès à tous les plans de bits
        mov   ah,0Fh
        mov   al,SC_MAP_MASK
        out   dx,ax
        pop   dx                                            ;Reprend DX

                ;-- Routine de copie pour tous les quatre plans de bits,
                                                ;--  ne respecte pas le fond

mt1:    mov   cl,dl                                 ;Nombre d'octets en CL

        rep movsb                                           ;Copie une ligne
        add   di,bx                                 ;DI sur ligne suivante
        add   si,bx                                         ; ainsi que SI
        dec   dh                                      ;Reste une ligne ?
        jne   mt1                                   ;Oui---> on continue
        jmp short mtend                             ;Non, prépare la sortie

                ;-- Routine de copie pour plans de bits individuels avec
                ;-- exploitation du tableau de masques binaires transmis

mt2:    mov   byte ptr frame.restz,dh ;Mémorise d'abord les variables qui
        mov   byte ptr frame.movec,dl           ;sont stockées dans la pile
        mov   frame.additif,bx              ;sous forme de variables locales

        mov   al,SC_MAP_MASK        ;Adresse le registre MAP MASK permanent
        mov   dx,SC_INDEX
        out   dx,al
        inc   dx                          ;DX pointe sur le registre de données

        push  ds
        lds   bx,frame.bmskp  ;BX pointe sur le tableau de masque binaire
        mov   al,[bx]                               ;Charge le premier octet
        xor   ah,ah                         ;Commence avec un octet pair

        pop   ds

mt3:    mov   cl,byte ptr frame.movec                ;Nombre d'octets en CL

mt4:    out   dx,al                                  ;Fixe le masque binaire
        movsb                                                ;Copie 4 octets

        inc   ah                        ;Incrémente le compteur pair/impair
        test  ah,1                                        ;Valeur impaire ?
        jne   mt5                                      ;Oui, déplace le quartet

              ;-- Octet pair, passe à l'octet suivant du buffer -------

        inc   bx                ;BX sur le prochain octet de masque binaire
        push  ds
        mov   ds,word ptr frame.bmskp+2
        mov   al,[bx]                                ;Charge l'octet suivant
        pop   ds
        loop  mt4                          ;Passe aux quatre latches suivants
        jmp short mt6

mt5:    shr   al,1                          ;Transfère dans le quartet faible
        shr   al,1                        ;le masque binaire de l'octet impair
        shr   al,1
        shr   al,1
        loop  mt4                          ;Passe aux quatres latches suivants

mt6:    add   di,frame.additif                          ;DI sur ligne suivante
        add   si,frame.additif                                  ;ainsi que SI
        dec   byte ptr frame.restz                    ;Reste une ligne ?
        jne   mt3                                     ;Oui ---> on continue

mtend:  mov   dx,GC_INDEX                    ;Rétablit l'ancien mode Write
        pop   ax
        mov   ah,al
        mov   al,GC_GRAPH_MODE
        out   dx,ax

        pop   ds
        pop   bp

        add   sp,6              ;SP passe par desus les variables locales
        ret   20              ;Au retour efface les paramètres sur la pile
                                                                        ;
blockmove  endp

;-- Fin ---------------------------------------------------------------

CODE    ends                                        ;Fin du segment de code
        end                                              ;Fin du programme
```

Listing : V3220PA.ASM (déjà mentionné en 4.8.7)

Sprites en mode graphique 320*400 points avec 256 couleurs

En mode graphique 320*400 points, la haute résolution s'offre aux deux pages écran utilisables pour recevoir les modèles et le fond des Sprites en mode 320*200 points. Ici, il n'existe aucune autre alternative que celle consistant à stocker les modèles et le fond des Sprites dans la mémoire RAM du PC et établir une routine de recopie entre la mémoire principale et la mémoire RAM de la carte vidéo. Du point de vue de la vitesse d'exécution, ces routines n'ont naturellement rien à envier à la copie directe à l'intérieur de la RAM vidéo. Mais comme les Sprites le montreront dans cette section, une telle insuffisance reste pratiquement indiscernable par l'utilisateur.

Par ailleurs, les routines de création et d'affichage des Sprites sont devenues beaucoup plus faciles. Dorénavant, il n'est plus possible de copier simultanément quatre octets entre la mémoire principale et la RAM vidéo comme c'est le cas au moyen des quatre

registres Latch dans la RAM vidéo. C'est aussi la raison pour laquelle il n'est pas indispensable d'étendre les Sprites sur une largeur divisible par quatre de même qu'il est inutile de les disposer par quatre. En bref, un Sprite de largeur quelconque peut être copié à une coordonnée X quelconque. Désormais, il n'est plus utile d'avoir recours à un tableau avec des valeurs pour le registre Map Mask. Toute médaille a tout de même deux côtés !

La modification de la sauvegarde et de la transmission des modèles Sprite dans la RAM vidéo se réflète naturellement dans les divers modules conçus pour la programmation des Sprites en mode graphique 320*400 points. Du côté C, il s'agit des modules S3240C.C, S3240CA.ASM et V3240CA.ASM. Les modules S3240P.PAS, S3240PA.ASM et V3240PA.ASM démontrent la programmation des Sprites dans ce mode vidéo au programmeur en Pascal.

Les routines PUTVIDEO et GETVIDEO représentent l'interface entre la mémoire principale et la RAM vidéo dans les modules cités plus haut. GETVIDEO permet de charger le contenu d'une zone rectangulaire de l'écran depuis la RAM vidéo vers la mémoire principale pour sauvegarder par exemple le fond d'un Sprite dans un buffer. A l'inverse, PUTVIDEO transfère le contenu d'une zone écran préalablement sauvée par GETVIDEO en un endroit quelconque de la RAM vidéo.

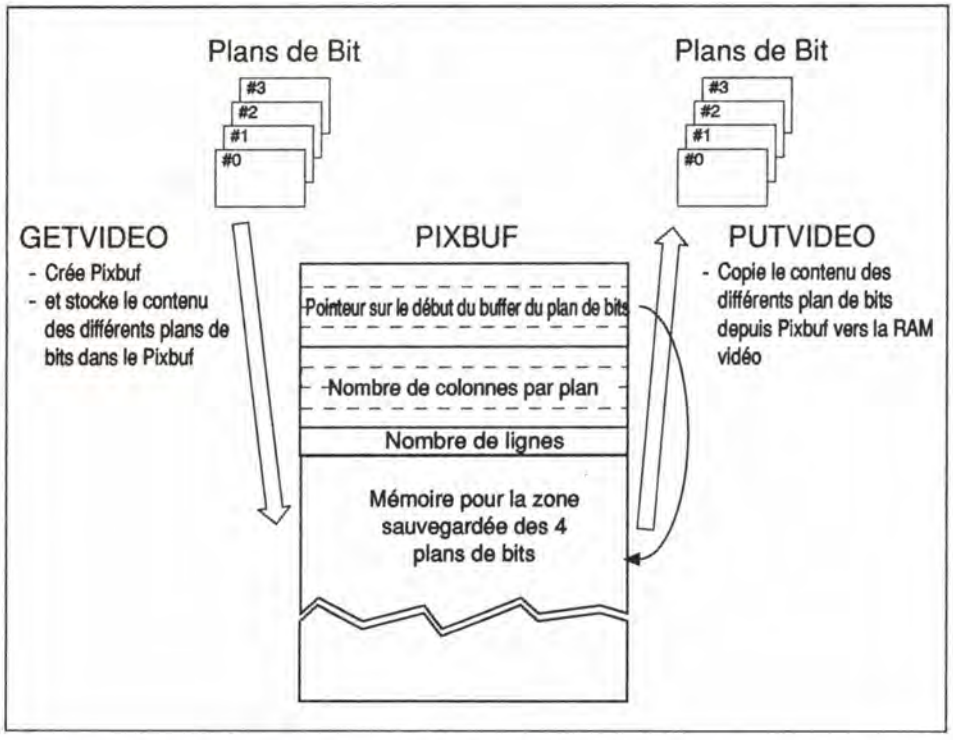
Les deux routines se servent de deux routines du module Assembleur S3240CA.ASM ou S3240PA.ASM portant les noms COPYBUF2PLANE et COPYPLANE2BUF. Leur tâche consiste à transférer une zone rectangulaire de points écran depuis un plan de bits vers la mémoire principale ou inversement.

Les quatre plans de bits dans lesquels sont répartis les points de tous les modes 256 couleurs sont traités séparément ici. Les deux routines sont par conséquent appelées quatre fois dans GETVIDEO et PUTVIDEO pour lire successivement les quatre plans de bits et y écrire. Cette opération est plus rapide que celle consistant à traiter en une seule passe quatre points des quatre plans de bits parce qu'il faut alors programmer le registre Read Map du contrôleur graphique ou le registre Map Mask du contrôleur Sequencer. La programmation de ces registres doit s'effectuer cependant une seule fois à chaque appel de COPYBUF2PLANE ou COPYPLANE2BUF parce qu'un seul plan de bits bien déterminé est adressé au cours de l'exécution de ces routines.

Vu que GETVIDEO et PUTVIDEO n'utilisent pas uniquement des zones écran rectangulaires dont la largeur est un multiple de quatre, la largeur des zones à traiter à l'intérieur des plans de bits n'est pas toujours identique. Imaginez par exemple un appel de GETVIDEO où il s'agit de charger une zone de la RAM vidéo s'étendant depuis la coordonnée X 0 jusqu'à la coordonnée X 6. Dans cette zone, deux points par ligne de points sont issus systématiquement du premier plan de bits (coordonnées X 0 et 4), également 2 du second plan de bits (coordonnées X 1 et 5) et encore 2 du troisième plan de bits (coordonnées X 2 et 6). Mais le quatrième plan de bits n'offre qu'un seul point (coordonnée X 3) convenant à cette zone.

Lors de son appel, GETVIDEO doit vérifier le nombre de points considérés dans chaque plan de bits. Elle doit conserver ces informations quelque part parce qu'elles peuvent servir pour un appel ultérieur de GETVIDEO. Ces informations - ainsi que d'autres - sont conservées dans un bloc de données. GETVIDEO le définit à travers la Heap lors de son appel et transmet à l'utilisateur au moyen d'un pointeur.

Ce bloc de données introduit par un tableau de quatre pointeurs porte la désignation PIXBUF. Ces derniers renvoient aux quatre buffers stockant le contenu de la zone écran extraite des quatre plans de bits. Vient ensuite un autre tableau à quatre entrées contenant le nombre de points stockés par plan de bits. Puis vient la hauteur de la zone c'est-à-dire le nombre de lignes de points.



*Sauvegarde de zones écran avec GETVIDEO et PUTVIDEO dans les programmes Sprite du mode graphique 320*400 points avec 256 couleurs*

La structure de données PIXBUF se termine avec la hauteur de la zone écran sauvegardée. Mais GETVIDEO associe directement le buffer à cette structure lequel reçoit les informations de points depuis les différents plans de bits. Cette action est possible parce qu'un buffer pixel de type PIXBUF est alloué à travers la Heap lors de l'appel de GETVIDEO. La taille du buffer nécessaire aux quatre plans de bits est ajoutée automatiquement à ce buffer. En guise de résultat, GETVIDEO reçoit une zone mémoire commençant par un PIXBUF et suivie des informations de points extraites des plans de bits au moyen de COPYPLANE2VIDEO.

GETVIDEO retourne un pointeur sur le buffer pixel ainsi créé. L'utilisateur peut utiliser ce pointeur aussi souvent que nécessaire lors d'un prochain appel de PUTVIDEO. La zone mémoire sauvegardée peut ainsi être copiée à tout moment en un endroit quelconque de la RAM vidéo.

GETVIDEO ne se contente pas de retourner un pointeur sur un buffer pixel. Elle peut également recevoir un pointeur sur un tel buffer en tant que dernier paramètre lors de son appel. Si un tel pointeur est précisé au lieu d'un NIL (en Pascal) ou de la constante ALLOKBUF (en C), les différentes informations sont reportées dans un buffer pixel déjà existant. Il s'agit d'un buffer défini par GETVIDEO lors d'un appel antérieur et référencé maintenant à l'aide du pointeur spécifié. L'avantage est qu'il n'est pas besoin d'allouer à chaque fois un nouveau buffer ce qui nécessite un temps assez important. Mais la condition préalable est de ne pas stocker une zone mémoire volumineuse dans le nouveau buffer pour que la place mémoire allouée ne soit pas insuffisante.

Dans les programmes Sprite concernant le mode 320*400 points, cette technique est par exemple utilisée pour créer l'arrière-plan des Sprites. Etant donné que la taille du Sprite est toujours la même, il en va naturellement de même pour l'arrière-plan. Il suffit donc d'allouer une seule fois un buffer approprié.

GETVIDEO et PUTVIDEO ne servent pas uniquement à sauvegarder l'arrière-plan d'un Sprite et le restaurer par la suite. Mais elles interviennent également dans la conversion de l'image d'un Sprite dans COMPILESPRITE. L'image du Sprite conservée dans un tableau chaîne est notamment utilisée pour construire un tel Sprite à partir des coordonnées (0/0) dans la page écran spécifiée par l'utilisateur comme un paramètre.

La zone recouverte par le Sprite est ensuite transférée dans un buffer pixel à travers un appel de GETVIDEO. Si le Sprite doit être affiché par la suite sur l'écran, il suffit de transmettre ce buffer pixel à PUTVIDEO sans oublier de mentionner la page écran concernée et les coordonnées. Cela explique également pourquoi le pointeur placé sur le buffer pixel est stocké dans la structure SPLOOK sous le nom PIXBP. Cette structure décrit l'image du Sprite.

Le principe suivi pour l'arrière-plan du Sprite est identique à celui utilisé pour le modèle du Sprite. Grâce à GETVIDEO, on définit tout simplement deux buffers pixel dont la taille correspond au Sprite concerné à l'intérieur de CREATESPRITE. Le premier reçoit l'arrière-plan de la première page écran, le second celui de la seconde page écran. Les pointeurs sur ces deux buffers pixels sont stockés dans un tableau appelé HGPTR au sein d'une structure de type SPID créée par CREATESPRITE pour identifier et utiliser un Sprite.

Les routines telles que PRINTSPRITE, GETSPRITEBG et RESTORESPRITEBG peuvent se servir du buffer pixel contenant le modèle du Sprite et des deux buffers d'arrière-plan pour afficher le Sprite sur l'écran, sauvegarder et restaurer son arrière-plan. Si vous comparez le code programme des diverses routines avec celui des modules Sprite du mode 320*200 points, vous constaterez que l'affaire s'est quelque peu compliquée.

La question de savoir comment traiter les points d'arrière-plan n'a pas été encore élucidée. A cet effet, PUTVIDEO attend en plus un paramètre booléen désigné par BG. Il est à transmettre à la routine Assembleur COPYBUF2PLANE et indique s'il faut tenir compte de l'arrière-plan lors de la description des plans.

Si vous précisez la valeur TRUE pour ce paramètre, COPYBUF2PLANE ne copie pas à l'aveuglette le contenu du buffer spécifié dans la RAM vidéo. Elle charge au contraire chaque octet individuellement et vérifie s'il représente un point d'arrière-plan. Les points représentés par un espace dans le tableau chaîne sont marqués par le code couleur 255 affecté à tous les points dans CREATESPRITE. Dès que COPYBUF2PLANE rencontre un point muni de ce code couleur, elle ignore tout simplement ce point, n'écrit pas cette couleur dans la RAM vidéo et poursuit avec le prochain point du buffer.

C'est ainsi qu'on peut résoudre facilement le problème toutes les fois que la recopie entre la mémoire principale et la RAM vidéo s'opère avec une certaine lenteur. Mais en pratique, c'est à peine si ce ralentissement est discerné.

Ainsi se termine la description des caractéristiques des Sprites en mode graphique 320*400 points. Si vous vous posez encore des questions à propos du fonctionnement des programmes, n'hésitez pas à consulter les listings richement documentés.

Listing : S3240C.C

```
/*****************************************************************
*                          S 3 2 4 0 C . C                       *
**---------------------------------------------------------------**
*  Fonction         : Montre comment travailler avec des sprites  *
*                     dans le mode graphique VGA 320*400 en 256   *
*                     couleurs avec deux pages d'écran            *
*                     Ce programme utilise les routines en assembleur *
*                     des modules S3240CA.ASM et V3240CA.ASM.     *
**---------------------------------------------------------------**
*  Auteur       : MICHAEL TISCHER                                 *
*  Développé le :  9.09.1990                                      *
*  Dernière MAJ : 14.02.1992                                      *
**---------------------------------------------------------------**
*  Modèle mémoire   : SMALL                                       *
**---------------------------------------------------------------**
*  (MICROSOFT C)                                                  *
*  Compilation      : CL /AS s3240c.c v3240ca s3240ca             *
**---------------------------------------------------------------**
*  (BORLAND TURBO C)                                              *
*  Compilation      : Utiliser un fichier de projet avec le contenu *
*                     suivant                                     *
*                     s3240c.c                                    *
*                     v3240ca.obj                                 *
*                     s3240ca.obj                                 *
**---------------------------------------------------------------**
*  Appel            : s3240c                                      *
*****************************************************************/

#include <dos.h>
#include <stdarg.h>
#include <stdio.h>
#include <string.h>
#include <stdlib.h>
#include <conio.h>

/*-- Déclarations dépendantes du compilateur ----------------------*/

#ifdef __TURBOC__
  #include <alloc.h>
#else
  #include <malloc.h>
  #define random(x) ( rand() % (x+1) )          /* Fonction aléatoire */
#endif

/*-- Déclarations de types ----------------------------------------*/

typedef unsigned char BYTE;
typedef BYTE BOOL;

typedef struct {    /* Buffer de pixels pour GetVideo() et PutVideo() */
        BYTE *bitptr[4],                /* Pointeur sur plans de bits */
             opar1[4],                  /* Nombre d'octets à copier */
             hauteur;                   /* Nombre de lignes */
                        /*-- Ici les octets des plans de bits --------*/
        } PIXBUF;
typedef PIXBUF *PIXPTR;           /* Pointeur sur un buffer de pixels */

typedef struct {                            /* Image d'un sprite */
        BYTE   largeur,                     /* Largeur totale */
               hauteur;                 /* Hauteur en lignes de pixels */
        PIXPTR pixbp;                   /* Pointeur sur bloc de pixels */
        } SPLOOK;

typedef struct {                        /* Descripteur de sprite (ID) */
        SPLOOK *splookp;                    /* Pointeur sur l'image */
        int    x[2], y[2];              /* Coordonnées en page 0 et 1 */
        PIXPTR fondptr[2];          /* Pointeur sur le buffer du fond */
        } SPID;

/*-- Références externes aux routines en assembleur ---------------*/

extern void init320400( void );
extern void setpix( int x, int y, unsigned char couleur);
extern BYTE getpix( int x, int y );
extern void setpage( BYTE page );
extern void showpage( BYTE page );
extern void far * getfontptr( void );

extern void copybuf2plane( BYTE *bufptr, BYTE page,
                           int ax, int ay, BYTE largeur,
                           BYTE hauteur, BOOL bg );
extern void copyplane2buf( BYTE *bufptr, BYTE page,
                           int dex, int dey, BYTE largeur,
                           BYTE hauteur );

/*-- Constantes ---------------------------------------------------*/
```

```
#define TRUE  ( 0 == 0 )
#define FALSE ( 0 == 1 )

#define MAXX 319                              /* Coordonnées maximales */
#define MAXY 399

#define OUT_LEFT   1 /* Indicateurs de collisions  pour SpriteMove() */
#define OUT_TOP    2
#define OUT_RIGHT  4
#define OUT_BOTTOM 8
#define OUT_NO     0                          /* Pas de collision */

#define ALLOCBUF ((PIXPTR) 0)     /* GetVideo(): allocation de buffer */

/*****************************************************************
*  IsVga: Teste la présence d'une carte VGA                      *
*----------------------------------------------------------------*
*  Entrée  : néant                                               *
*  Sortie  : 0, si aucune carte VGA n'est branchée, sinon # 0    *
*****************************************************************/

BYTE IsVga( void )
{
 union REGS Regs;                 /* Registres pour gérer l'interruption */

 Regs.x.ax = 0x1a00;              /* La fonction 1Ah n'existe qu'en VGA */
 int86( 0x10, &Regs, &Regs );
 return ( Regs.h.al == 0x1a );                /* Est-elle disponible ? */
}

/*****************************************************************
*  PrintChar : Ecrit un caractère en dehors de la zone visible   *
*              de la mémoire d'écran                             *
*----------------------------------------------------------------*
*  Entrée  :   caractere = caractère à afficher                  *
*              x, y      = Coordonnées du coin supérieur gauche  *
*              cc        = Couleur du caractère                  *
*              cf        = Couleur du fond                       *
*  Info    : Le caractère est dessiné dans une matrice de 8*8 pixels - *
*            sur la base du jeu de caractères 8*8 en ROM         *
*****************************************************************/

void PrintChar( char caractere, int x, int y, BYTE cc, BYTE cf )
{
 typedef BYTE CARDEF[256][8];       /* Structure du jeu de caractères */
 typedef CARDEF far *CARPTR;        /* Pointe sur un jeu de caractères */

 BYTE    i, k,                              /* Compteur d'itérations */
         masque;          /* Masque binaire pour dessiner le caractère */

 static CARPTR fptr = (CARPTR) 0;          /* Jeu de caractères en ROM */

 if( fptr == (CARPTR) 0 )       /* A-t-on déjà déterminé ce pointeur ? */
   fptr = getfontptr();     /* Non, détermine avec fonction assembleur */

  /*- Dessine le caractère pixel par pixel ---------------------------*/

 if( cf == 255 )                              /* Caractère transparent ? */
   for( i = 0; i < 8; ++i ) /* Oui, ne dessine que pixels du 1er plan */
   {
     masque = (*fptr)[caractere][i];  /* Lit motif bin pour une ligne */
     for ( k = 0; k < 8; ++k, masque <<= 1 ) /* Parcourt les colonnes */
      if ( masque & 128 )                    /* Pixel à dessiner ? */
       setpix( x+k, y+i, cc );                          /* Oui */
   }
 else                                    /* Non dessine chaque pixel */
   for ( i = 0; i < 8; ++i )                  /* Parcourt les lignes */
   {
     masque = (*fptr)[caractere][i];      /* Motif bin pour une ligne */
     for ( k = 0; k < 8; ++k, masque <<= 1 ) /* Parcourt les colonnes */
       setpix( x+k, y+i, (BYTE) (( masque & 128 ) ? cc : cf) );
   }
}

/*****************************************************************
*  Line: Trace un segment dans la fenêtre graphique en appliquant *
*        l'algorithme de Bresenham                               *
*----------------------------------------------------------------*
*  Entrées : X1, Y1 = Coordonnées de l'origine                   *
*            X2, Y2 = Coordonnées de l'extrémité terminale       *
*            COULEUR = couleur du segment                        *
*****************************************************************/

/*-- Fonction accessoire pour échanger deux variables entières -------*/

void SwapInt( int *i1, int *i2 )
{
 int   dummy;

 dummy = *i2; *i2  = *i1; *i1  = dummy;
}

/*-- Procédure principale ---------------------------------------------*/

void Line( int x1, int y1, int x2, int y2, BYTE couleur )
{
 int d, dx, dy,
     aincr, bincr,
     xincr, yincr,
     x, y;

 if( abs(x2-x1) < abs(y2-y1) )     /* Sens du parcours : axe X ou Y ? */
 {                                                          /* Par Y */
   if ( y1 > y2 )                        /* y1 plus grand que y2? */
   {
     SwapInt( &x1, &x2 );                /* Oui échange X1 et X2, */
     SwapInt( &y1, &y2 );                                /* Y1 et Y2 */
   }

   xincr = ( x2 > x1 ) ? 1 : -1;          /* Fixe le pas horizontal */

   dy    = y2 - y1;
   dx    = abs( x2-x1 );
   d     = 2 * dx - dy;
   aincr = 2 * (dx - dy);
   bincr = 2 * dx;
   x     = x1;
   y     = y1;

   setpix( x, y, couleur );              /* dessine le premier pixel */
   for(y=y1+1; y<= y2; ++y )               /* Parcourt l'axe des Y */
   {
     if ( d >= 0 )
     {
       x += xincr;
       d += aincr;
     }
     else
       d += bincr;
     setpix( x, y, couleur );
   }
 }
 else                                                       /* par X */
 {
   if( x1 > x2 )                         /* x1 plus grand que x2? */
   {
     SwapInt( &x1, &x2 );                /* Oui, échange X1 et X2 */
     SwapInt( &y1, &y2 );                                /* Y1 et Y2 */
   }

   yincr = ( y2 > y1 ) ? 1 : -1;          /* Fixe le pas vertical */

   dx    = x2 - x1;
   dy    = abs( y2-y1 );
   d     = 2 * dy - dx;
   aincr = 2 * (dy - dx);
   bincr = 2 * dy;
   x     = x1;
   y     = y1;

   setpix( x, y, couleur );              /* Dessine le premier pixel */
   for(x=x1+1; x<=x2; ++x )                /* Parcourt l'axe des X */
   {
     if( d >= 0 )
     {
       y += yincr;
       d += aincr;
     }
     else
       d += bincr;
     setpix( x, y, couleur );
   }
 }
}

/*****************************************************************
*  GrafPrintf: Affiche une chaîne formatée sur l'écran graphique *
*----------------------------------------------------------------*
*  Entrées : X, Y   = Coordonnées de départ (0 - ...)            *
*            CC     = Couleur des caractères                     *
*            CF     = Couleur du fond  (255 = transparent)       *
*            STRING = Chaîne avec indications de formatage       *
*              ...  = Expressions comme pour printf              *
*****************************************************************/

void GrafPrintf( int x, int y, BYTE cc, BYTE cf, char * string, ... )
{
 va_list parameter;     /* Liste de paramètres pour les macros VA_... */
 char    affichage[256],            /* Buffer pour la chaîne formatée */
         *cp;

 va_start( parameter, string );          /* Convertit les paramètres */
 vsprintf( affichage, string, parameter );              /* Formate */
 for( cp = affichage; *cp; ++cp, x+= 8 )        /* Affiche la chaîne */
   PrintChar( *cp, x, y, cc, cf );          /* formatée par PrintChar */
```

```
!}
!
!/*****************************************************************
!*  GetVideo: Charge le contenu d'une zone rectangulaire de la mémoire  *
!*            d'écran dans un buffer                                    *
!*----------------------------------------------------------------------*
!*  Entrée : PAGE   = Page d'écran (0 ou 1)                             *
!*           X1, Y1 = Coordonnées de départ                             *
!*           LARGEUR = Largeur de la zone recatngulaire en pixels       *
!*           HAUTEUR  = Hauteur de la zone rectangulaire en pixels      *
!*           BUFPTR = Pointeur sur un buffer de pixels qui va    -      *
!*                    mémoriser les informations                        *
!*  Sortie  : Pointeur sur le buffer créé qui contient la zone indiquée *
!*                                                                      *
!*  Info    : Si on donne au paramètre BUFPTR la valeur ALLOCBUF        *
!*            un nouveau buffer de pixels est alloué sur le tas et      *
!*            retourné. Ce buffer peut être transmis lors d'un nouvel   *
!*            appel si l'ancien contenu est effaçable et si la taille   *
!*            de la zone n'a pas changé                                 *
!*****************************************************************/
!
!PIXPTR GetVideo( BYTE page, int x1, int y1, BYTE largeur, BYTE hauteur,
!                 PIXPTR bufptr )
!{
! BYTE i,                                    /* Compteur d'itérations */
!      plancour,                             /* Plan de bits courant */
!      sb,                          /* Plan de bits des coordonnées de début */
!      eb,                           /* Plan de bits des coordonnées de fin */
!      b,                           /* Nombre d'octets dans un plan de bits */
!      am;     /* Nombre d'octets au milieu des deux groupes de quatre */
! BYTE *rptr; /* Pointeur sur la pos. d'un plan de bits dans le buffer */
!
! if( bufptr == ALLOCBUF )                  /* Pas de buffer transmis ? */
!   bufptr = malloc( sizeof( PIXBUF ) + largeur*hauteur );   /* alloue */
!
!  /*-- calcule le nombre d'octets par plan de bits ------------------*/
!
! am = (BYTE) (((x1+largeur-1) & 3 )     /* Nombre d'octets au milieu */
!              - ( (x1+4) & 3) ) >> 2;
! sb = (BYTE) (x1 % 4);                        /* Plan de bit de début */
! eb = (BYTE) ((x1+largeur-1) % 4);             /* Plan de bit de fin */
!
! rptr = (BYTE *) bufptr + sizeof( PIXBUF );
!
! /*-- Parcourt les quatre plans de bits ----------------------------*/
!
! for( i=0; i<4; ++i )
! {
!   plancour = (sb+i) % 4;
!   b = am;                      /* Nombre de base des octets à copier */
!   if( plancour >= sb )         /* dans le bloc des quatre du début ? */
!     ++b;                     /* Oui, ajoute un octet dans ce plan de bits */
!   if( plancour <= eb )         /* dans le bloc des quatre de la fin ? */
!     ++b;                   /* Oui ajoute un octet dans ce plan de bits */
!   bufptr->bitptr[i] = rptr;          /* Mémorise pointeur sur le début */
!   bufptr->oparl[i] = b;              /* Mémorise le nombre d'octets */
!   copyplane2buf( rptr, page, x1+i,                 /* Lit le contenu - */
!          y1, b, hauteur );                                 /* des plans */
!   rptr += (b * hauteur);                      /* Positionne le pointeur */
! );                                          /* sur le plan suivant dans */
!                                                          /* le buffer */
! bufptr->hauteur = hauteur;                    /* Mémorise la hauteur */
!
! return bufptr;     /* Renvoie à l'appelant le pointeur sur le buffer */
!}
!
!/*****************************************************************
!*  PutVideo: Réécrit dans la mémoire d'écran                           *
!*            le  contenu d'une zone d'écran rectangulaire préalablement*
!*            sauvegardée par GetVideo()                                *
!*----------------------------------------------------------------------*
!*  Entrée : BUFPTR  = Pointeur renvoyé par GetVideo et référençant     *
!*                     un buffer de pixels                              *
!*           PAGE    = Page d'écran   (0 ou 1)                          *
!*           X1, Y1 = Coordonnées de début                              *
!*           BG      = Indique si les pixels du fond (code couleur 255) *
!*                     doivent être écrits dans la mémoire d'écran      *
!*  Info    : Le buffer de pixels n'est pas effacé par cette procédure  *
!*            cette tâche étant remplie par FreePixBuf()                *
!*****************************************************************/
!
!void PutVideo( PIXPTR bufptr, BYTE page, int x1, int y1, BOOL bg )
!{
! BYTE  plancour,                            /* Plan de bits courant */
!       hauteur;
!
! hauteur = bufptr->hauteur;                    /* Hauteur de la zone */
! for( plancour=0; plancour<4; ++plancour )    /* Parcourt 4 bitplans */
!   copybuf2plane( bufptr->bitptr[plancour], page, x1+plancour,
!                  y1, bufptr->oparl[plancour], hauteur, bg );
!}
!
!/*****************************************************************
!*  FreePixBuf: Efface un buffer de pixels alloué sur le tas par        *
!*              GetVideo()                                              *
!*----------------------------------------------------------------------*
!*  Entrée : BUFPTR = Pointeur renvoyé par Getvideo et référençant      *
!*                    un buffer de pixels                               *
!*           LARGEUR = Largeur de la zone rectangulaire en pixels       *
!*           HAUTEUR = Hauteur de la zone rectangulaire en pixels       *
!*****************************************************************/
!
!void FreePixBuf( PIXPTR bufptr )
!{
! free( bufptr );
!}
!
!/*****************************************************************
!*  CreateSprite: Crée un sprite à l'aide d'un motif de pixels          *
!*                préalablement compilé                                 *
!*----------------------------------------------------------------------*
!*  Entrée : SPLOOKP = pointeur sur la structure de données produite    *
!*                     par CompileSpriteI()                             *
!*  Sortie : Pointeur sur la structure du sprite créée                  *
!*  Info   : La mémorisation du fond du sprite nécessite deux zones     *
!*           contiguës de la taille du sprite                           *
!*****************************************************************/
!
!SPID *CreateSprite( SPLOOK *splookp )
!{
! SPID   *spidp;                      /* Pointe sur la structure du sprite */
!
! spidp = (SPID *) malloc( sizeof(SPID) );   /* mémoire pour structure */
! spidp->splookp = splookp;                  /* Y reporte les données */
!
!/*- Crée deux buffers de fond dans lesquels GetVideo va sauvegarder -*/
!                                  /*- des zones de la mémoire d'écran -*/
!
! spidp->fondptr[0] = GetVideo( 0, 0, 0, splookp->largeur,
!                               splookp->hauteur, ALLOCBUF );
! spidp->fondptr[1] = GetVideo( 0, 0, 0, splookp->largeur,
!                               splookp->hauteur, ALLOCBUF );
! return spidp;      /* Renvoie un pointeur sur la structure du sprite */
!}
!
!/*****************************************************************
!*  CompileSprite: Crée le motif binaire d'un sprite à l'aide d'une     *
!*                 définition connue au moment de l'exécution           *
!*----------------------------------------------------------------------*
!*  Entrée : BUFP     = Pointeur sur un tableau de pointeurs référençant*
!*                      des chaînes de caractères qui représentent     *
!*                      le motif du sprite                             *
!*           HAUTEUR = Hauteur du sprite et nombre de chaînes de       *
!*                      caractères                                     *
!*           PAGE    = Page graphique pour construire le sprite        *
!*           CLR     = Caractère ASCII associé à la plus               *
!*                      petite couleur                                 *
!*           COULEURPP= Premier code de couleur pour CLR               *
!*  Info    : Les sprites sont dessinés à partir du bord gauche        *
!*            de la ligne indiquée                                     *
!*****************************************************************/
!
!SPLOOK *CompileSprite( char **bufp, BYTE hauteur, BYTE page,
!                       char fb, BYTE couleurpp )
!{
! BYTE   largeur,             /* Longueur des chaînes = largeur du motif */
!        c,                                  /* Mémorise un caractère */
!        i, k;                               /* Compteurs d'itérations */
! SPLOOK *splookp;   /* Ptr sur la structure de l'image du sprite créée*/
! PIXPTR pbptr;           /* Mémorise temporairement le fond du sprite */
!
! /*-- Crée la structure de l'image du sprite et la remplit ----------*/
!
! splookp          = (SPLOOK *) malloc( sizeof(SPLOOK) );
! largeur          = (BYTE) strlen( *bufp );
! splookp->largeur = largeur;
! splookp->hauteur = hauteur;
!
!   /*-- Construit le sprite dans la page indiquée à partir de (0,0)--*/
!
! setpage( page );                            /* Fixe la page de dessin */
! showpage( page );
! pbptr = GetVideo( page, 0, 0, largeur, hauteur, ALLOCBUF );  /* fond */
!
! for( i = 0; i < hauteur; ++i )                /* Parcourt les lignes */
!   for( k = 0; k < largeur; ++k )            /* Parcourt les colonnes */
!   {
!     c = *(*(bufp+i)+k);
!     setpix( k, i, (BYTE) (c == ' ' ? 255 : couleurpp+(c-fb)));
!   }
!
! /*-- Lit le sprite dans le buffer et restaure le fond    ------------*/
!
! splookp->pixbp = GetVideo( page, 0, 0, largeur, hauteur, ALLOCBUF );
! PutVideo( pbptr, page, 0, 0, FALSE );
! FreePixBuf( pbptr );                          /* Libère le buffer */
```

```
}

return splookp;        /* Renvoie un pointeur sur le buffer de l'image */
}

/*****************************************************************
*  PrintSprite : Affiche un sprite dans une page donnée          *
*****************************************************************
*  Entrée : SPIDP = Pointeur sur la structure du sprite          *
*           PAGE = Page concernée                                *
*                  (0 ou 1)                                      *
*****************************************************************/

void PrintSprite( register SPID *spidp, BYTE page )
{
 PutVideo( spidp->splookp->pixbp,
           page, spidp->x[page], spidp->y[page], TRUE );
}

/*****************************************************************
*  GetSpriteBg: Lit le fond du sprite et le mémorise à l'emplacement *
*               prévu                                            *
*****************************************************************
*  Entrée : SPIDP = Pointeur sur la structure du sprite          *
*           PAGE  = Page d'où est tiré le fond                   *
*                   (0 ou 1)                                     *
*****************************************************************/

void GetSpriteBg( register SPID *spidp, BYTE page )
{
 GetVideo(
           page, spidp->x[page],
           spidp->y[page],
           spidp->splookp->largeur,
           spidp->splookp->hauteur,
           spidp->fondptr[page]
         );
}

/*****************************************************************
*  RestoreSpriteBg: Rétablit dans la page d'origine le fond d'un sprite *
*                   sauvegardé au préalable                      *
*****************************************************************
*  Entrée : SPIDP = Pointeur sur la structure du sprite          *
*           PAGE  = Page où doit être recopié le fond (0 ou 1)   *
*****************************************************************/

void RestoreSpriteBg( register SPID *spidp, BYTE page )
{
 PutVideo(
           spidp->fondptr[page],
           page,
           spidp->x[page],
           spidp->y[page],
           FALSE
         );
}

/*****************************************************************
*  MoveSprite: Déplace un sprite                                 *
*****************************************************************
*  Entrée : SPIDP  = Pointeur sur la structure du sprite         *
*           PAGE   = Page où doit être recopié le fond (0 ou 1)  *
*           DELTAX = Déplacement dans les directions X et Y      *
*           DELTAY                                               *
*  Sortie  : Indicateur de collision, cf constantes OUT_...      *
*****************************************************************/

BYTE MoveSprite( SPID *spidp, BYTE page, int deltax, int deltay )
{
 int    nouvx, nouvy;              /* Nouvelles coordonnées du sprite */
 BYTE   out;          /* Indique une collision avec le bord de l'écran */

    /*-- Décale l'abscisse X et teste s'il y a collision avec le bord */
 if( ( nouvx = spidp->x[page] + deltax ) < 0 )
 {
   nouvx = 0 - deltax - spidp->x[page];
   out = OUT_LEFT;
 }
 else
   if( nouvx > MAXX - spidp->splookp->largeur )
   {
     nouvx = (2 * ( MAXX + 1 ) ) - nouvx - 2 * spidp->splookp->largeur;
     out = OUT_RIGHT;
   }
   else
     out = OUT_NO;

/*-- Décale l'ordonnée Y et teste s'il y a collision avec le bord ----*/

 if( ( nouvy = spidp->y[page] + deltay ) < 0 )          /* Bord sup ?*/
 {                                          /* Oui deltay doit être négatif */
   nouvy = 0 - deltay - spidp->y[page];
   out |= OUT_TOP;
 }
 else
   if( nouvy + spidp->splookp->hauteur > MAXY+1 )       /* Bord inf ?*/
   {                                        /* Oui, deltay doit être positif */
     nouvy = (2 * (MAXY+1) ) - nouvy - 2 * spidp->splookp->hauteur;
     out |= OUT_BOTTOM;
   }

/*-- Fixe nouvelle pos que si elle est différente de l'ancienne */

 if ( nouvx != spidp->x[page]  ||  nouvy != spidp->y[page] )
 {                                          /* Nouvelle position */
   RestoreSpriteBg( spidp, page );          /* Rétablit le fond */
   spidp->x[page] = nouvx;                  /* Mémorise les nouvelles */
   spidp->y[page] = nouvy;                  /*   coordonnées */
   GetSpriteBg( spidp, page );              /* Lit le nouveau fond */
   PrintSprite( spidp, page );   /* Dessine sprite dans page indiquée */
 }
 return out;
}

/*****************************************************************
*  SetSprite: Place le sprite à une position donnée              *
*****************************************************************
*  Entrée : SPIDP  = Pointeur sur la structure du sprite         *
*           x0, y0 = Coordonnées du sprite en page 0             *
*           x1, y1 = Coordonnées du sprite en page 1             *
*  Info    : Cette fonction doit être déclenchée avant le premier *
*            appel à MoveSprite()                               *
*****************************************************************/

void SetSprite( SPID *spidp, int x0, int y0, int x1, int y1 )
{
 spidp->x[0] = x0;      /* Mémorise les coordonnées dans la structure */
 spidp->x[1] = x1;
 spidp->y[0] = y0;
 spidp->y[1] = y1;

 GetSpriteBg( spidp, 0 );                   /* Lit le fond du sprite en */
 GetSpriteBg( spidp, 1 );                   /*   pages 0 et 1 */
 PrintSprite( spidp, 0 );                   /* Dessine le sprite en page 0 */
 PrintSprite( spidp, 1 );                   /*   et 1 */
}

/*****************************************************************
*  RemoveSprite: Retire un sprite de l'emplacement qu'il occupe  *
*                et le rend invisible                            *
*****************************************************************
*  Entrée : SPIDP = Pointeur sur la structure du sprite          *
*  Info   : A l'issue de cette fonction, il faut appeler SetSprite()  *
*           avant de déplacer le sprite par MoveSprite()         *
*****************************************************************/

void RemoveSprite( SPID *spidp )
{
 RestoreSpriteBg( spidp, 0 );               /* Rétablit le fond du sprite */
 RestoreSpriteBg( spidp, 1 );               /*   en pages 0 et 1 */
}

/*****************************************************************
*  Demo: Démontre l'usage des différentes fonctions de ce module *
*****************************************************************/

void Demo( void )
{
 static char *VaisseauMontant[20] =
                 {  "                AA              ",
                    "               AAAA             ",
                    "               AAAA             ",
                    "                AA              ",
                    "               GGBBGG           ",
                    "              GBBCCBBG          ",
                    "             GBBCCBBBG          ",
                    "            GBBBBBBBBBG         ",
                    "            GBBBBBBBBBG         ",
                    "          G GBBBBBBBBBBBG     G ",
                    "         GCG  GGDBBBBBBBBBDGG  GCG",
                    "         GCG  GGBBBCBBB  BBBCBBBGG  GCG",
                    "         GCBGGGBBBBBCBB    BBCBBBBBGGGBCG",
                    "         GCBBBBBBBBBBDB    BDBBBBBBBBBBCG",
                    "         BBBBBBBBBBBBDB BB BDBBBBBBBBBBBB",
                    "         GGCBBBBBBBBDBBBBBBBBBDBBBBBBCG ",
                    "          GGCCBBBDDDDDDDDDDDDDDBBBCCG   ",
                    "           GGBBDDDDDGGGGGDDDDDDBBG      ",
                    "             GDDDDGGG    GGGDDDDG       ",
                    "              DDDD          DDDD        " };

 static char *VaisseauDescendant[20] =
                 {
                    "              DDDD          DDDD        ",
                    "             GDDDDGGG    GGGDDDDG       ",
                    "           GGBBDDDDDGGGGGDDDDDDBBG      ",
```

```
                "  GGCC888DOOOOOOOOOOOO8B8CCG   ",
                "GGC88888888D888888888D8888888CG ",
                "B8888888888888DB BB 8D8888888888888",
                "GC888888888888DB    8D888888888BCG",
                "GC8GGG88888D8B    88D8888888GGGBCG",
                "GCG   GG888D888  888D8B8GG   GCG",
                "GCG     GGD8888888888DGG     GCG",
                " G       G888888888888G       G ",
                "          G888888888BG          ",
                "          G888888888BG          ",
                "           G88BCC888G           ",
                "            G8BCC8BG            ",
                "             GGBGG              ",
                "               AA               ",
                "              AAAA              ",
                "              AAAA              ",
                "               AA               " );

#define NBSPR   6                       /* Nombre de sprites */
#define LARGEUR 38  /* Nombre de caractères de l'avis de copyright */
#define HAUTEUR 6               /* Nombre de lignes du même avis */
#define SX      (MAXX-(LARGEUR*8)) / 2      /* Abscisse de départ X */
#define SY      (MAXY-(HAUTEUR*8)) / 2      /* Ordonnée de départ Y */

struct {                                /* décrit les sprites gérés */
     SPID *spidp;                       /* Pointeur sur l'identificateur */
     int  deltax[2],     /* Déplacement horizontal pour pages 0 et 1 */
          deltay[2];     /* Déplacement vertical pour pages 0 et 1 */
   } sprites[ NBSPR ];
BYTE   page,                            /* Page présentement traitée */
       out;                     /* Mémorise l'indicateur de collision */
int    x, y, i,                         /* Compteurs d'itérations */
       dx, dy;                                  /* Déplacements */
char   lc;
SPLOOK *vaisseauupp, *vaisseaudnp;      /* Pointeurs sur les sprites */

srand( *(int far *) 0x0040006cl );
                /* Initialise le générateur de nombres aléatoires */

     /*-- Construit les motifs binaires des sprites ----------------*/

vaisseauupp = CompileSprite( VaisseauMontant,  20, 0, 'A', 1 );
vaisseaudnp = CompileSprite( VaisseauDescendant, 20, 20, 'A', 1 );

      /*-- Remplit de caractères les deux pages graphiques----------*/

for( page = 0; page < 2; ++ page )
{
  setpage( page );
  for( lc = 0, y = 0; y < (MAXY+1)-8; y += 12 )
    for( x = 0; x < (MAXX+1)-8; x += 8 )
      GrafPrintf( x, y, lc % 255, 255, "%c", lc++ & 127 );

  /*-- Affiche l'avis de copyright -------------------------------*/

  Line( SX-1, SY-1, SX+LARGEUR*8, SY-1, 15 );
  Line( SX+LARGEUR*8, SY-1, SX+LARGEUR*8, SY+HAUTEUR*8,15 );
  Line( SX+LARGEUR*8, SY+HAUTEUR*8, SX-1, SY+HAUTEUR*8, 15 );
  Line( SX-1, SY+HAUTEUR*8, SX-1, SY-1, 15 );
  GrafPrintf( SX, SY,    15, 4,
              "                                    " );
  GrafPrintf( SX, SY+8,  15, 4,
              "  S3240C (c) 1990, 92 Michael Tischer " );
  GrafPrintf( SX, SY+16, 15, 4,
              "                                    " );
  GrafPrintf( SX, SY+24, 15, 4,
              "  Démonstration de sprites en mode  " );
  GrafPrintf( SX, SY+32, 15, 4,
              "       VGA 320x400 256 couleurs     " );
  GrafPrintf( SX, SY+40, 15, 4,
              "                                    " );
}

/*-- Fabrique les différents sprites  ----------------------------*/

for( i = 0; i < NBSPR  ; ++ i)
{
  sprites[ i ].spidp = CreateSprite( vaisseauupp );
  do                                    /* Sélectionne les déplacements */
  {
    dx = 0;
    dy = random(8) - 4;
  }
  while( dx==0  &&  dy==0 );

  sprites[ i ].deltax[0] = sprites[ i ].deltax[1] = dx * 2;
  sprites[ i ].deltay[0] = sprites[ i ].deltay[1] = dy * 2;

  x = ( 320 / NBSPR  * i ) + (320 / NBSPR  - 40) / 2 ;
  y = random( (MAXY+1) - 40 );
  SetSprite(
            sprites[ i ].spidp,
            x,
            y,
            x - dx,
            y - dy
           );
}

/*-- Déplace sprites et les fait rebondir aux extrémités de l'écran */

page = 1;                                       /* Commence en page 1 */
while( !kbhit() )       /* Une frappe de touche interrompt la boucle */
{
  showpage( (BYTE) (1 - page) );                /* Affiche l'autre page */

  for( i = 0; i < NBSPR; ++ i)                  /* Affiche les sprites */
  {                     /* Déplace les sprites et teste les collisions */
    out = MoveSprite( sprites[i].spidp, page, sprites[i].deltax[page],
                      sprites[i].deltay[page] );
    if ( out & OUT_TOP  ||  out & OUT_BOTTOM )          /* Contact ? */
    {                   /* Oui inverse le déplacement et change l'image */
      sprites[i].deltay[page] = 0 - sprites[i].deltay[page];
      sprites[i].spidp->splookp = ( out & OUT_TOP ) ? vaisseaudnp
                                                    : vaisseauupp;
    }
    if ( out & OUT_LEFT  ||  out & OUT_RIGHT )
      sprites[i].deltax[page] = 0 - sprites[i].deltax[page];
  }
  page = (page+1) & 1;                  /* Passe de 1 à 0 et vice-versa */
}
!}

!/*-------------------------------------------------------------------*/
!/*-- PROGRAMME PRINCIPAL                                           --*/
!/*-------------------------------------------------------------------*/

!void main( void )
!{
! union REGS regs;

! if ( IsVga() )                                 /* A-t-on une carte VGA? */
! {                                                    /* Oui c'est parti */
!  init320400();                              /* Initialise le mode graphique */
!  Demo();
!  getch();                                   /* Attend une frappe de touche */
!  regs.x.ax = 0x0003;                            /* Rétablit le mode texte */
!  int86( 0x10, &regs, &regs );
! }
! else
!  printf( "S3240C.C - (c) 1990, 92 by MICHAEL TISCHER\n\nATTENTION "\
!          "Ce programme nécessite une carte VGA");
!}
```

Listing : S3240CA.ASM

```
;*****************************************************************;
;*                      S 3 2 4 0 C A . A S M                    *;
;*---------------------------------------------------------------*;
;*   Fonction       : contient des routines pour travailler avec les  *;
;*        sprites dans le mode 320*400-256  de la carte VGA      *;
;*---------------------------------------------------------------*;
;*   Auteur         : MICHAEL TISCHER                            *;
;*   Développé le   :  8.09.1990                                 *;
;*   Dernière MAJ   : 14.02.1992                                 *;
;*---------------------------------------------------------------*;
;*   Modèle mémoire : SMALL                                      *;
;*---------------------------------------------------------------*;
;*   Assemblage     : MASM /mx S3240CA;    ou     TASM -mx S3240CA  *;
;*                    ... puis lier à S3240C.C                   *;
;*****************************************************************;

IGROUP group _text                      ;Regroupe les segments de programme
DGROUP group _bss, _data                ;Regroupe les segments de données
```

```
        assume CS:IGROUP, DS:DGROUP, ES:DGROUP, SS:DGROUP

_BSS    segment word public 'BSS'          ;Segment réservé aux variables
_BSS    ends                               ;statiques non initialisées

_DATA   segment word public 'DATA'         ;Segment des variables globales
                                           ;et statiques non initialisées

_DATA   ends

;-- Constantes ----------------------------------------------------------

SC_INDEX        = 3c4h      ;Registre d'index du contrôleur du séquenceur
SC_MAP_MASK     = 2                        ;Numéro de registre Map Mask
SC_MEM_MODE     = 4                    ;Numéro du registre de mode mémoire

GC_INDEX        = 3ceh          ;Registre d'index du contrôleur graphique
GC_READ_MAP     = 4                        ;Numéro du registre Read Map

PIXX            = 320                          ;Résolution horizontale

;-- Programme -----------------------------------------------------------

_TEXT   segment byte public 'CODE'                  ;Segment de programme
;-- Déclarations publiques ----------------------------------------------

public    _copybuf2plane
public    _copyplane2buf

;--------------------------------------------------------------------------
;-- CopyBuf2Plane: copie contenu d'un buffer dans une zone rectangulaire
;--                d'un plan de bits
;-- Déclaration: CopyBuf2Plane( byte *bufptr,
;--                             byte verspage,
;--                             int  versx,
;--                             int  versy,
;--                             byte largeur,
;--                             byte hauteur,
;--                             bool bg );

_copybuf2plane proc near

sfr0    struc                          ;Structure d'accès à la pile
bp0     dw ?                                       ;Mémorise BP
ret_adr0 dw ?                      ;Adresse de retour à l'appelant
bufptr0 dw ?                               ;Pointeur sur buffer
verspage dw ?                              ;Page de destination
versx   dw ?                           ;Abscisse de destination
versy   dw ?                           ;Ordonnée de destination
largeur0 dw ?                                      ;Largeur
hauteur0 dw ?                          ;Hauteur de la zone
bg      dw ?                          ;Tient compte du fond
sfr0    ends                                   ;Fin de la structure

fr      equ [ bp - bp0 ]          ;adresse les éléments de la structure
bfr     equ byte ptr [ bp - bp0 ]    ;adresse éléments pile comme octets

        push  bp                       ;Prépare l'adressage des paramètres
        mov   bp,sp                            ; par le registre BP
        push  di
        push  si

        ;-- Calcule le segment d'accès à la mémoire d'écran -------------

        mov   ah,0A0h             ;ES au début de la page de destination
        cmp   bfr.verspage,0                       ;Est-ce la page 0?
        je    cv0                              ;Oui AL est tout bon

        mov   ah,0A8h                        ;Non, page 1 en A800h

cv0:    xor   al,al                      ;Octet faible toujours nul
        mov   es,ax

        ;-- Calcule l'offset de la position de destination dans la page -

        mov   ax,PIXX / 4              ;DI sur position de destination
        mul   fr.versy
        mov   di,fr.versx
        mov   cx,di                              ;mémorise l'abscisse X
        shr   di,1
        shr   di,1
        add   di,ax

        ;-- Prépare l'adressage du plan de bits  ------------------------

        mov   ah,1                          ;Le numéro du plan est pris
        and   cl,3                                 ; comme masque binaire
        shl   ah,cl
        mov   dx,SC_INDEX                   ;Assure l'accès au plan
        mov   al,SC_MAP_MASK
        out   dx,ax

        ;-- Charge les compteurs pour la boucle de copie ----------------

        mov   dh,bfr.hauteur0                          ;DH = lignes
        mov   dl,bfr.largeur0                          ;DL = octets
        mov   bx,PIXX / 4              ;BX = offset sur ligne suivante
        sub   bl,dl
        xor   ch,ch                 ;Octet fort du compteur toujours nul

        mov   si,fr.bufptr0          ;DS:SI doit pointer sur le buffer

        cmp   bfr.bg,0                          ;Fond à respecter ?
        jne   cv2                           ;Oui, utilise une autre routine

        ;-- Routine de copie pour un plan de bits, ne respecte ----------
                                                       ;-- pas le fond
cv1:    mov   cl,dl                              ;Nombre d'octets en CL

        rep   movsb                                 ;Copie une ligne
        add   di,bx                              ;DI sur ligne suivante
        dec   dh                                   ;Reste une ligne ?
        jne   cv1                                ;Oui---> on continue

        jmp   short cvend                          ;Non, buffer copié

        ;-- Routine de copie pour plan de bits avec exploitation --------
                                                       ;-- du buffer transmis
cv2:    mov   cl,dl                              ;Nombre d'octets en CL

cv3:    lodsb                                  ;Charge un octet du buffer
        cmp   al,255                                  ;Octet de fond ?
        je    cv5                                    ;Oui ---> ne ps copier
        stosb                          ;Non, le transfère en mémoire d'écran
        loop  cv3                                 ;Traite l'octet suivant

cv4:    ;- Avance le pointeur de la mémoire d'écran sur la ligne suivante

        add   di,bx                              ;DI sur ligne suivante
        dec   dh                                 ;Reste-t-il une ligne ?
        jne   cv2                                  ;Oui ---> on continue
        jmp   short cvend                            ;Non, buffer copié

cv5:    ;-- Octet de fond à ne pas copier ----------------------------

        inc   di                    ;Ne pas mettre cet octet en mémoire d'écran
        loop  cv3                       ;Reste-t-il un octet dans cette ligne ?
        jmp   cv4                                  ;Non ---> ligne suivante

cvend:  pop   si                          ;Reprend les registres sur la pile
        pop   di
        pop   bp
        ret                                       ;Retourne à l'appelant

_copybuf2plane endp

;--------------------------------------------------------------------------
;-- CopyPlane2Buf: Copie une zone zone rectangulaire d'un plan de bits
;--                dans un buffer
;-- Déclaration : CopyPlane2Buf( byte *bufptr,
;--                              byte depage,
;--                              int  dex,
;--                              int  dey,
;--                              byte largeur,
;--                              byte hauteur );

_copyplane2buf proc near

sfr1    struc                          ;Structure d'accès à la pile
bp1     dw ?                                       ;Mémorise BP
ret_adr1 dw ?                      ;Adresse de retour à l'appelant
bufptr1 dw ?                               ;Pointeur sur le buffer
depage  dw ?                                  ;Page d'origine
dex     dw ?                              ;Abscisse d'origine
dey     dw ?                              ;Ordonnée d'origine
largeur1 dw ?                      ;Largeur de la zone en pixels
hauteur1 dw ?                      ;Hauteur de la zone en pixels
sfr1    ends                                   ;Fin de la structure

fr      equ [ bp - bp1 ]          ;adresse les éléments de la structure
bfr     equ byte ptr [ bp - bp1 ]    ;adresse pile comme octet

        push  bp                       ;Prépare l'adressage des paramètres
        mov   bp,sp                            ; par le registre BP
        push  di
        push  si
        push  ds                                   ;Sauvegarde DS
        push  ds                           ;Deuxième exemplaire pour ES

        ;-- Calcule le segment d'accès à la mémoire d'écran -------------

        mov   ah,0A0h             ;ES au début de la page d'origine
        cmp   bfr.depage,0                         ;Est-ce la page 0 ?
        je    cc0                              ;Oui AL est bon
```

```
        mov   ah,0A8h                           ;Non page 1 en A800h

cc0:    xor   al,al                         ;Octet faible toujours nul
        mov   ds,ax

        ;-- Forme l'offset dans la page à lire ------------------------

        mov   ax,PIXX / 4                   ;SI sur position d'origine
        mul   fr.dey
        mov   si,fr.dex
        mov   cx,si                         ;Mémorise l'abscisse en CX
        shr   si,1
        shr   si,1
        add   si,ax

        ;-- Prépare l'adressage du plan de bits ------------------------

        and   cl,3                  ;Calcule en AH le masque binaire pour
        mov   ah,cl                                 ; le plan concerné
        mov   al,GC_READ_MAP                ;Numéro du registre en AL
        mov   dx,GC_INDEX           ;Charge index du contrôleur graphique
        out   dx,ax                         ;Lit le registre Read Map

        ;-- Charge les compteurs pour la boucle de copie ---------------

        mov   dh,bfr.hauteur1                           ;DH = lignes
        mov   dl,bfr.largeur1                           ;DL = octets
        mov   bx,PIXX / 4               ;BX = offset sur ligne suivante
        sub   bl,dl

        xor   ch,ch                     ;Octet fort du compteur toujours 0

        pop   es                            ;ES:DI doit pointer sur le buffer
        mov   di,fr.bufptr1

        ;-- Routine de copie pour un plan de bits sans tenir ------------
        ;-- compte du fond ----------------------------------------------

ccl:    mov   cl,dl                                 ;Nombre d'octets en CL

        rep   movsb                                         ;Copie une ligne
        add   si,bx                                 ;SI sur ligne suivante
        dec   dh                                    ;reste-t-il une ligne ?
        jne   ccl                                           ;Oui, on continue

        pop   ds                                            ;Récupère DS
        pop   si
        pop   di
        pop   bp

        ret                                         ;Retourne à l'appelant

_copyplane2buf endp

;== Fin ========================================================

_text   ends                                ;Fin du segment de programme
        end                             ;Fin de la source en assembleur
```

Listing : V3240CA.ASM (déjà mentionné en 4.8.7)

Listing : S3240P.PAS

```
{*********************************************************************
*                          S 3 2 4 0 P . P A S                       *
**--------------------------------------------------------------------**
* Fonction : Monte comment travailler avec des sprites dans le mode   *
*            graphique VGA 320*200 avec 256 couleurs et deux pages    *
*            d'écran. Ce programme utilise les routines en assembleur *
*            des modules V3240PA.ASM et S3240PA.ASM                   *
**--------------------------------------------------------------------**
* Auteur       : MICHAEL TISCHER                                      *
* Développé le : 12.09.1990                                           *
* Dernière MAJ : 14.01.1991                                           *
*********************************************************************}

program S3240P;

uses dos, crt;

{-- Références externes aux routines en assembleur ------------------}

{$L v3240pa}      { Intègre le module en assembleur }

procedure init320400; external;
procedure setpix( x, y : integer; couleur : byte ); external;
function  getpix( x, y: integer ) : byte ; external;
procedure setpage( page : byte ); external;
procedure showpage( page : byte ); external;

{$L s3240pa}   { Intègre le module en assembleur }

procedure CopyPlane2Buf( bufptr : pointer;
                         page   : byte;
                         dex,
                         dey    : integer;
                         largeur,
                         hauteur  : byte    ); external;

procedure CopyBuf2Plane( bufptr : pointer;
                         page   : byte;
                         ax,
                         ay  : integer;
                         largeur,
                         hauteur  : byte;
                         bg     : boolean ); external;

{-- Constantes ------------------------------------------------------}

const MAXX = 319;                               { Coordonnées maximales }
      MAXY = 399;

      OUT_LEFT   = 1;     { Indicateurs de collision dans SpriteMove() }
      OUT_TOP    = 2;
      OUT_RIGHT  = 4;
      OUT_BOTTOM = 8;
      OUT_NO     = 0;                               { Pas de collision }

{-- Déclarations de types -------------------------------------------}

type PIXBUF = record          { Informations pour GetVideo et PutVideo }
       bitptr : array[0..3] of pointer;          {Ptr sur plans de bits}
       oparl  : array[0..3] of byte;           { Nb d'octets à copier }
       hauteur: byte;                                { Nombre de lignes }
                                {-- Ici les octets des plans de bits --}
     end;
     PIXPTR = ^PIXBUF;              { Pointeur sur un buffer de pixels }

     SPLOOK = record                                  { Image d'un sprite }
       largeur,                                          { Largeur totale }
       hauteur  : byte;                        { Hauteur en lignes de pixels }
       pixbp    : PIXPTR;                      { Pointeur sur bloc de pixels }
     end;
     SPLP = ^SPLOOK;                                { Pointeur sur une image }

     SPID = record                               { Descripteur de sprite (ID) }
       splookp : SPLP;                                 { Pointeur sur l'image }
       x, y    : array [0..1] of integer;        { Coord pages 0 et 1 }
       fondptr : array [0..1] of PIXPTR;         { Ptr sur le buffer }
     end;                                                          { du fond }
     SPIP = ^SPID;                  { Pointeur sur le descripteur du sprite }

     PTRREC = record   { Pour décomposer un pointeur ou un entier long }
                ofs,
                seg : word;
```

```
!              end;
!
!     BYTEAR = array[0..10000] of byte;  { Pour adresser les différents }
!     BARPTR = ^BYTEAR;                                  { buffers    }
!
!{*********************************************************************
!*  IsVga : Teste la présence d'une carte VGA.                       *
!**-----------------------------------------------------------------**
!*  Entrée  : néant                                                  *
!*  Sortie  : TRUE ou FALSE                                          *
!*********************************************************************}
!
!function IsVga : boolean;
!
!var Regs : Registers;                    { Registres pour l'interruption}
!
!begin
!  Regs.AX := $1a00;                { La fonction 1Ah n'existe qu'en VGA }
!  Intr( $10, Regs );
!  IsVga := ( Regs.AL = $1a );
!end;
!
!{*********************************************************************
!*  PrintChar : Affiche un caractère en mode graphique               *
!**-----------------------------------------------------------------**
!*  Entrée : caractere = le caractère à afficher                     *
!*           x, y      = Coordonnées du coin sup gauche              *
!*           cc        = Couleur du caractère                        *
!*           cf        = Couleur du fond                             *
!*  Info   : Le caractère est dessiné dans une matrice de 8*8 pixels *
!*           sur la base du jeu de caractères 8*8 en ROM             *
!*********************************************************************}
!
!procedure PrintChar( caractere : char; x, y : integer; cc, cf : byte );
!
!type CARADEF = array[0..255,0..7] of byte;{ Structure jeu de caractères}
!     CARAPTR = ^CARADEF;                 { Pointe sur le jeu de caractères }
!
!var  Regs  : Registers;        { Registres pour gérer les interruptions }
!     ch    : char;                                  { Pixel du caractère }
!     i, k,                                      { Compteur d'itérations }
!     Masque : byte;          { Masque binaire pour dessiner le caractère }
!
!const fptr : CARAPTR = NIL;   { Pointe sur le jeu de caractères en ROM }
!
!begin
!  if fptr = NIL then              { A-t-on déjà déterminé ce pointeur ? }
!    begin                                                       { Non}
!      Regs.AH := $11;                  { Appelle l'option $1130 de la    }
!      Regs.AL := $30;                  { fonction vidéo du BIOS          }
!      Regs.BH := 3;       { pour obtenir un pointeur sur le jeu 8*8      }
!      intr( $10, Regs );
!      fptr := ptr( Regs.ES, Regs.BP );            { Compose le pointeur }
!    end;
!
!  if ( cf = 255 ) then                        { Caractère transparent ? }
!    for i := 0 to 7 do { Oui ne dessine que les pixels du premier plan }
!      begin
!        Masque := fptr^[ord(caractere),i];{Motif binaire pour une ligne}
!        for k := 0 to 7 do
!          begin
!            if ( Masque and 128 <> 0 ) then       { Pixel à dessiner? }
!              setpix( x+k, y+i, cc );                          { Oui }
!            Masque := Masque shl 1;
!          end;
!      end
!  else                                        { Non, tient compte du fond }
!    for i := 0 to 7 do                          { Parcourt les lignes }
!      begin
!        Masque := fptr^[ord(caractere),i];{Motif binaire pour une ligne}
!        for k := 0 to 7 do
!          begin
!            if ( Masque and 128 <> 0 ) then          { Premier plan ? }
!              setpix( x+k, y+i, cc )                           { Oui }
!            else
!              setpix( x+k, y+i, cf );                  { Non, fond }
!            Masque := Masque shl 1;
!          end;
!      end;
!end;
!
!{*********************************************************************
!*  Line : Trace un segment dans la fenêtre graphique en appliquant  *
!*         l'algorithme de Bresenham                                 *
!**-----------------------------------------------------------------**
!*  Entrée : X1, Y1  = Coordonnées de l'origine (0 - ...)            *
!*           X2, Y2  = Coordonnées de l'extrémité terminale          *
!*           COULEUR = couleur du segment                            *
!*********************************************************************}
!
!procedure Line( x1, y1, x2, y2 : integer; couleur : byte );
!
!var d, dx, dy,
!    aincr, bincr,
!    xincr, yincr,
!    x, y              : integer;
!
!{-- Procédure accessoire pour échanger deux variables entières --------}
!
!procedure SwapInt( var i1, i2: integer );
!
!var dummy : integer;
!
!begin
!  dummy := i2;
!  i2    := i1;
!  i1    := dummy;
!end;
!
!{-- Procédure principale ----------------------------------------------}
!
!begin
!  if ( abs(x2-x1) < abs(y2-y1) ) then    { Parcours : par axe X ou Y ? }
!    begin                                                 { par l'axe des Y }
!      if ( y1 > y2 ) then                                 { y1 supérieur à y2? }
!        begin
!      SwapInt( x1, x2 );                          { Oui, échange X1 et X2, }
!      SwapInt( y1, y2 );                                  { Y1 et Y2 }
!        end;
!
!      if ( x2 > x1 ) then xincr := 1          { Fixe le pas horizontal }
!                     else xincr := -1;
!
!      dy := y2 - y1;
!      dx := abs( x2-x1 );
!      d  := 2 * dx - dy;
!      aincr := 2 * (dx - dy);
!      bincr := 2 * dx;
!      x := x1;
!      y := y1;
!
!      Setpix( x, y, couleur );               { Dessine le premier point }
!      for y:=y1+1 to y2 do                     { Parcourt l'axe des Y }
!        begin
!          if ( d >= 0 ) then
!            begin
!              inc( x, xincr );
!              inc( d, aincr );
!            end
!          else
!            inc( d, bincr );
!          Setpix( x, y, couleur );
!        end;
!    end
!  else                                                  { par l'axe des X }
!    begin
!      if ( x1 > x2 ) then                          { x1 plus grand que x2? }
!        begin
!      SwapInt( x1, x2 );                          { Oui, échange X1 et X2 }
!      SwapInt( y1, y2 );                                  { Y1 et Y2 }
!        end;
!
!      if ( y2 > y1 ) then yincr := 1          { Fixe le pas vertical }
!                     else yincr := -1;
!
!      dx := x2 - x1;
!      dy := abs( y2-y1 );
!      d  := 2 * dy - dx;
!      aincr := 2 * (dy - dx);
!      bincr := 2 * dy;
!      x := x1;
!      y := y1;
!
!      Setpix( x, y, couleur );               { Dessine le premier point }
!      for x:=x1+1 to x2 do                     { Parcourt l'axe des X }
!        begin
!          if ( d >= 0 ) then
!            begin
!              inc( y, yincr );
!              inc( d, aincr );
!            end
!          else
!            inc( d, bincr );
!          Setpix( x, y, couleur );
!        end;
!    end;
!end;
!
!{*********************************************************************
!*  GrafPrint : Affiche une chaine formatée sur l'écran graphique    *
!**-----------------------------------------------------------------**
!*  Entrées : X, Y   = Coordonnées de départ (0-...)                 *
!*            CC     = Couleur des caractères                        *
!*            CF     = Couleur du fond (255 = transparent)           *
!*            STRING = Chaine avec indications de formatage          *
!*********************************************************************}
```

```
!
!procedure GrafPrint( x, y : integer; cc, cf : byte; strt : string );
!
!var i : integer;                                   ( Compteur d'itérations )
!
!begin
! for i:=1 to length( strt ) do
!   begin
!     printchar( strt[i], x, y, cc, cf );          ( Affiche par printchar )
!     inc( x, 8 );                  ( x à la position du caractère suivant )
!   end;
!end;
!
!
!{*****************************************************************
!* GetVideo : Charge le contenu d'une zone rectangulaire de la mémoire *
!*            d'écran dans un buffer                                 *
!*---------------------------------------------------------------**
!* Entrées : PAGE     = Page d'écran  (0 ou 1)                     *
!*           X1, Y1   = Coordonnées de départ                      *
!*           LARGEUR  = Largeur de la zone rectangulaire en pixels *
!*           HAUTEUR  = Hauteur de la zone rectangulaire en pixels *
!*           BUFPTR   = Pointeur sur un buffer de pixels qui va    *
!*                      mémoriser les informations                 *
!* Sortie  : Pointeur sur le buffer de pixels qui contient la zone *
!*           indiquée                                              *
!* Info    : Si on donne au paramètre BUFPTR la valeur NIL , un    *
!*           nouveau buffer de pixels est alloué sur le tas et retourné*
!*           Ce buffer peut être transmis lors d'un nouvel appel si *
!*           l'ancien contenu est effaçable et si la taille de la zone *
!*           n'a pas changé                                        *
!*****************************************************************}
!
!function GetVideo( page : byte; x1, y1 : integer;
!                   largeur, hauteur : byte; bufptr : PIXPTR ) : PIXPTR;
!
!var i,                                              ( Compteur d'itérations )
!    plancour,                                       ( Plan de bits courant )
!    sb,                                ( Plan de bits des coordonnées de début )
!    eb,                                 ( Plan de bits des coordonnées de fin )
!    b,                                 ( Nombre d'octets dans un plan de bits )
!    am  : byte;  { Nombre d'octet au milieu des deux groupes de quatre )
!    rptr : pointer;                          ( Pointeur courant sur l'offset )
!
!begin
!  if ( bufptr = NIL ) then                      ( Pas de buffer transmis ?)
!    getmem( bufptr, sizeof(PIXBUF) + largeur*hauteur);( Non, on alloue )
!
!    {-- Calcule le nombre d'octets par plan de bits ------------------}
!
!  am := ( ( (x1+largeur-1) and not(3) ) -  {Nombre d'octets au milieu  }
!       ( (x1+4) and not(3) ) ) div 4;
!  sb := x1 mod 4;                                  ( Plan de bit de début )
!  eb := (x1+largeur-1) mod 4;                       ( Plan de bit de fin )
!
!  rptr := ptr( seg(bufptr^), ofs(bufptr^) + sizeof( PIXBUF ));
!
!  {-- Parcourt les quatre plans de bits ----------------------------}
!
!  for i:=0 to 3 do
!    begin
!      plancour := (sb+i) mod 4;
!      b := am;                      ( Nombre de base des octets à copier   )
!      if ( plancour >= sb ) then
!                        ( également dans le bloc des quatre du début ? )
!        inc( b );                    ( Oui, ajoute un octet dans ce plan   )
!        if ( plancour <= eb ) then
!                        ( également dans le bloc des quatre de la fin ? )
!        inc( b );                     ( Oui, ajoute un octet dans ce plan )
!        bufptr^.bitptr[i] := rptr;
!                    ( Mémorise dans le buffer un pointeur sur le début )
!        bufptr^.oparl[i] := b;              ( Mémorise le nombre d'octets )
!        CopyPlane2Buf( rptr, page, x1+i,                ( Lit le contenu  )
!             y1, b, hauteur );                                ( des plans )
!        inc( PTRREC(rptr).ofs, b * hauteur );  ( Positionne le pointeur   )
!                                      ( sur le plan suivant dans le buffer )
!    end;
!  bufptr^.hauteur := hauteur;                      ( mémorise la hauteur )
!
!  GetVideo := bufptr; { Renvoie à l'appelant le pointeur sur le buffer )
!end;
!
!{*****************************************************************
!* PutVideo : Réécrit dans la mémoire d'écran le contenu d'une zone *
!*            d'écran rectangulaire préalablement sauvegardée par   *
!*            GetVideo                                              *
!*---------------------------------------------------------------**
!* Entrée : BUFPTR = Pointeur renvoyé par GetVideo et référençant un *
!*                   buffer de pixels                                *
!*          PAGE   = Page d'écran (0 ou 1)                           *
!*          X1, Y1 = Coordonnées de début                            *
!*          BG     = Indique si les pixels du fond (code couleur 255) *
!*                   doivent être écrits dans la mémoire d'écran    *
!* Info    : Le buffer de pixels n'est pas effacé par cette procédure *
!*           cette tâche étant remplie par FreePixBuf()            *
!*****************************************************************}
!
!procedure PutVideo( bufptr : PIXPTR; page : byte; x1, y1 : integer;
!                    bg     : boolean );
!
!var plancour,                                       ( Plan de bits courant )
!    hauteur : byte;
!
!begin
!  hauteur := bufptr^.hauteur;                          ( Hauteur de la zone )
!  for plancour:=0 to 3 do             ( Parcourt les quatre plans de bits )
!    CopyBuf2Plane( bufptr^.bitptr[plancour], page, x1+plancour,
!                   y1, bufptr^.oparl[plancour], hauteur, bg );
!end;
!
!{*****************************************************************
!* FreePixBuf : Efface un buffer de pixels alloué sur le tas par    *
!*              GetVideo                                            *
!*---------------------------------------------------------------**
!* Entrée : BUFPTR  = Pointeur renvoyé par Getvideo et référençant  *
!*                    un buffer de pixels                           *
!*          LARGEUR = Largeur de la zone rectangulaire en pixels    *
!*          HAUTEUR = Hauteur de la zone rctangulaire en pixels     *
!*****************************************************************}
!
!procedure FreePixBuf( bufptr : PIXPTR; largeur, hauteur : byte );
!
!begin
!  freemem( bufptr, sizeof( PIXBUF ) + largeur*hauteur );
!end;
!
!{*****************************************************************
!* CreateSprite : Crée un sprite à l'aide d'un motif de pixels      *
!*                préalablement compilé                             *
!*---------------------------------------------------------------**
!* Entrée  : SPLOOKP = Pointeur sur la structure de données produite *
!*                     par CompileSprite()                          *
!* Sortie  : Pointeur sur la structure du sprite créée              *
!*****************************************************************}
!
!function CreateSprite( splookp : SPLP ) : SPIP;
!
!var spidp : SPIP;                          ( Pointe sur la structure du sprite   )
!
!begin
!  new( spidp );                 ( Alloue de la mémoire pour la structure      )
!  spidp^.splookp := splookp;
!
!  {-- Crée deux buffers de fond dans lesquels GetVideo va sauvegarder   )
!                                        (-- des zones de la mémoire d'écran )
!  spidp^.fondptr[0] := GetVideo( 0, 0, 0, splookp^.largeur,
!                                 splookp^.hauteur, NIL );
!  spidp^.fondptr[1] := GetVideo( 0, 0, 0, splookp^.largeur,
!                                 splookp^.hauteur, NIL );
!  CreateSprite := spidp;    ( Renvoie un ptr sur la structure du sprite )
!end;
!
!{*****************************************************************
!* CompileSprite : Crée le motif binaire d'un sprite à l'aide d'une *
!*                 définition connue au moment de l'exécution       *
!*---------------------------------------------------------------**
!* Entrée  : BUFP     = Pointeur sur un tableau de pointeurs        *
!*                      référençant des chaînes de caractères qui   *
!*                      représente le motif du sprite               *
!*           HAUTEUR  = Hauteur du sprite et nombre de chaînes de   *
!*                      caractères                                  *
!*           PAGE     = Page graphique pour construire le sprite    *
!*           CLR      = Caractère ASCII associé à la plus petite coul.*
!*           COULEURPP = Premier code de couleur pour CLR           *
!* Info    : Les sprites sont dessinés à partir du bord gauche de la *
!*           ligne indiquée                                        *
!*****************************************************************}
!
!function CompileSprite( var buf; hauteur, page : byte;
!          clr: char; couleurpp : byte ) : SPLP;
!
!type BYPTR  = ^byte;                                ( Pointeur sur un octet )
!
!var largeur,                  ( Longueur des chaînes = largeur du motif )
!    c,                                          ( Mémorise un caractère )
!    couleur,                                      ( Couleur d'un pixel )
!    i, k, l  : byte;                           ( Variables d'itération )
!    splookp  : SPLP;{ Ptr sur la structure de l'image du sprite créée )
!    bptr     : barptr; {Sert à adresser le buffer de l'image du sprite)
!    pbptr    : PIXPTR;                        ( Mémorise le fond du sprite )
!
!begin
!  {-- Crée la structure de l'image du sprite et la remplit ------------}
```

```
{
  new( splookp );
  bptr := @buf;                          { Pointe sur le buffer du logo }
  largeur := bptr^[0]; { Lit la longueur des chaines = largeur du logo }
  splookp^.largeur := largeur;
  splookp^.hauteur  := hauteur;

      {-- Construit le sprite dans la page indiquée à partir de (0,0)--}

  setpage( page );                              { Fixe la page de dessin   }
  pbptr := GetVideo( page, 0, 0, largeur, hauteur, nil );{ Lit le fond }

  for i := 0 to hauteur-1 do                    { Parcourt les lignes }
    for k := 0 to largeur-1 do                  { Parcourt les colonnes }
      begin
    c := bptr^[i*(largeur+1)+k+1];                          { Lit un pixel}
    if ( c = 32 ) then                                      { Fond ? }
      setpix( k, i, 255 )           { Oui, choisit le code couleur 255}
    else                                        { Non, met la couleur prévue }
          setpix( k, i, couleurpp+(c-ord(clr)) );
      end;

  {-- Lit le sprite dans le buffer et restaure le fond  ------------}

  splookp^.pixbp := GetVideo( page, 0, 0, largeur, hauteur, NIL );
  PutVideo( pbptr, page, 0, 0, false );
  FreePixBuf( pbptr, largeur, hauteur );            { Libère le buffer }

  CompileSprite := splookp; { Renvoie une ptr sur le buffer de l'image }
end;

{*********************************************************************
*  PrintSprite : Affiche un sprite dans une page donnée              *
**--------------------------------------------------------------------**
*  Entrée  : SPIDP = Pointeur sur la structure du sprite             *
*            PAGE  = Page concernée  (0 ou 1)                        *
*********************************************************************}

procedure PrintSprite( spidp : SPIP; page : byte );

begin
  PutVideo( spidp^.splookp^.pixbp,
            page, spidp^.x[page], spidp^.y[page], true );
end;

{*********************************************************************
*  GetSpriteBg : Lit le fond du sprite et le mémorise à l'emplacement *
*                prévu                                               *
**--------------------------------------------------------------------**
*  Entrées : SPIDP = Pointeur sur la structure du sprite             *
*            PAGE  = Page d'où est tiré le fond (0 ou 1)             *
*********************************************************************}

procedure GetSpriteBg( spidp : SPIP; page : BYTE );

var dummy : PIXPTR;

begin
  dummy := GetVideo( page, spidp^.x[page],  spidp^.y[page],
                     spidp^.splookp^.largeur, spidp^.splookp^.hauteur,
                     spidp^.fondptr[page] );
end;

{*********************************************************************
*  RestoreSpriteBg : Rétablit dans la page d'origine le fond d'un    *
*                    sprite sauvegardé au préalable                  *
**--------------------------------------------------------------------**
*  Entrée  : SPIDP = Pointeur sur la structure du sprite             *
*            PAGE  = Page où doit être recopié le fond (0 ou 1)      *
*********************************************************************}

procedure RestoreSpriteBg( spidp : SPIP; page : BYTE );

begin
  PutVideo( spidp^.fondptr[page], page,
            spidp^.x[page],  spidp^.y[page], false );
end;

{*********************************************************************
*  MoveSprite : Déplace un sprite dans sa page                       *
**--------------------------------------------------------------------**
*  Entrées : SPIDP  = Pointeur sur la structure du sprite            *
*            PAGE   = Page où doit être recopié le fond (0 ou 1)     *
*            DELTAX = Déplacement dans les directions X etY          *
*            DELTAY                                                  *
*  Sortie  : Indicateur de collision, cf constantes OUT_...          *
*********************************************************************}

function MoveSprite( spidp : SPIP; page : byte;
                     deltax, deltay : integer    ) : byte;

var nouvx, nouvy : integer;          { Nouvelles coordonnées du sprite }
    out          : byte; { Indique une collision avec le bord de l'écran }

begin
                  {-- Décale l'abscisse X et détecte les collisions --}

  nouvx := spidp^.x[page] + deltax;
  if ( nouvx  < 0 ) then
    begin
      nouvx := 0 - deltax - spidp^.x[page];
      out := OUT_LEFT;
    end
  else
    if ( nouvx > MAXX - spidp^.splookp^.largeur ) then
      begin
        nouvx := (2*(MAXX+1))-nouvx-2*(spidp^.splookp^.largeur);
        out := OUT_RIGHT;
      end
    else
      out := OUT_NO;

                        { Décale l'ordonnée Y et détecte les collisions }

  nouvy := spidp^.y[page] + deltay;
  if ( nouvy < 0 ) then                                  { Bord supérieur ? }
    begin                                  { Oui , deltay doit être négatif }
      nouvy := 0 - deltay - spidp^.y[page];
      out := out or OUT_TOP;
    end
  else
    if ( nouvy + spidp^.splookp^.hauteur > MAXY+1  ) then   { Bord inf ?}
      begin                                    { Oui, deltay doit être positif }
        nouvy := (2*(MAXY+1))-nouvy-2*(spidp^.splookp^.hauteur);
        out := out or OUT_BOTTOM;
      end;

{Ne fixe une nouvelle position que si elle est différente de l'ancienne}

  if ( nouvx <> spidp^.x[page] ) or  ( nouvy <> spidp^.y[page] ) then
    begin                                                { Nouvelle position }
      RestoreSpriteBg( spidp, page );                     { Restaure le fond }
      spidp^.x[page] := nouvx;      { Mémorise les nouvelles coordonnées }
      spidp^.y[page] := nouvy;
      GetSpriteBg( spidp, page );                    { Lit le nouveau fond }
      PrintSprite( spidp, page );{ Dessine sprite dans la page indiquée }
    end;

  MoveSprite := out;
end;

{*********************************************************************
*  SetSprite : Place le sprite à une position donnée                 *
**--------------------------------------------------------------------**
*  Entrées : SPIDP  = Pointeur sur la structure du sprite            *
*            x0, y0 = Coordonnées du sprite en page 0                *
*            x1, y1 = Coordonnées du sprite en page 1                *
*  Info    : Cette fonction doit être déclenchée avant le premier    *
*            appel à MoveSprite()                                    *
*********************************************************************}

procedure SetSprite( spidp : SPIP; x0, y0, x1, y1 : integer );

begin
  spidp^.x[0] := x0;        { Mémorise les coordonnées dans la structure }
  spidp^.x[1] := x1;
  spidp^.y[0] := y0;
  spidp^.y[1] := y1;

  GetSpriteBg( spidp, 0 );                          { Lit le fond du sprite }
  GetSpriteBg( spidp, 1 );                               { en pages 0 et 1 }
  PrintSprite( spidp, 0 );                           { Dessine le sprite}
  PrintSprite( spidp, 1 );                           { en pages 0 et 1   }
end;

{*********************************************************************
*  Demo : Démontre l'usage des différentes fonctions de ce module    *
*********************************************************************}

procedure Demo;

const VaisseauMontant :array [1..20] of string[32] =
                  ( '            ÄÄ            ',
                    '           ÄÄÄÄ           ',
                    '           ÄÄÄÄ           ',
                    '            ÄÄ            ',
                    '           ßßßßß          ',
                    '          ßßßßßßß         ',
                    '         ßßßßßßßßß        ',
                    '        ßßßßßßßßßßß       ',
                    '       ßßßßßßßßßßßßß      ',
                    '      ßßßßßßßßßßßßßßß     ',
                    '  ß   ßßßßßßßßßßßßßßßßß   ß ',
                    ' ßßß  ßßßßßßßßßßßßßßßßßßß  ßßß',
                    'ßßßßßßßßßßßßßßßßßßßßßßßßßßßßßßß',
                    'ßßßßßßßßßßßßßßßßßßßßßßßßßßßßßßß',
                    'ßßßßßßßßßßßßßßßßßßßßßßßßßßßßßßß',
                    'ßßßßßßßßßßßßßßßßßßßßßßßßßßßßßßß',
                    'ßßßßßßßßßßßßßßßßßßßßßßßßßßßßßßß',
                    'ßßßßßßßßßßßßßßßßßßßßßßßßßßßßßßß',
                    'ßßßßßßßßßßßßßßßßßßßßßßßßßßßßßßß',
                    'ßßßßßßßßßßßßßßßßßßßßßßßßßßßßßßß');
```

```
                'GCBGGGBBBBBDBB    BBDBBBBBGGGBCG',
                'GCBBBBBBBBBBDB    BDBBBBBBBBBCG',
                'BBBBBBBBBBBBDB BB BDBBBBBBBBBBB',
                'GBCBBBBBBBDBBBBBBBBBBDBBBBBBCG ',
                '  GGCCBBBDDDDDDDDDDDDDBBBCCG   ',
                '    GGBBDDDDDGGGGGDDDDDBBG     ',
                '      GDDDDGGG    GGGDDDDG      ',
                '       DDDD        DDDD       ' );

const VaisseauDescendant :array [1..20] of string[32] =
              ( '       DDDD        DDDD       ',
                '      GDDDDGGG    GGGDDDDG     ',
                '    GGBBDDDDGGGGGDDDDDBBG      ',
                '  GGCCBBBDDDDDDDDDDDDDBBBCCG   ',
                'GBCBBBBBBBDBBBBBBBBBBDBBBBBBCG ',
                'BBBBBBBBBBBDB BB BDBBBBBBBBBBB',
                'GCBBBBBBBBBDB    BDBBBBBBBBBCG',
                'GCBGGGBBBBBDBB    BBDBBBBGGGBCG',
                'GCG   GGBBBDBBB  BBBDBBBQG   GCG',
                'GCG    GGDBBBBBBBBBDGG    GCG',
                ' G      GBBBBBBBBBBBBG      G ',
                '         GBBBBBBBBBG          ',
                '         GBBBBBBBBBG          ',
                '          GBBBCCBBBG          ',
                '           GBBCCBBG           ',
                '            GBBGG             ',
                '             AA               ',
                '            AAAA              ',
                '            AAAA              ',
                '             AA               ' );

  NBSPR = 6;                                        ( Nombre de sprites  )
  LARGEUR = 37;          ( Nombre de caractères de l'avis de copyright )
  HAUTEUR  = 6;                    ( Nombre de lignes de ce même avis )
  SX     = (MAXX-(LARGEUR*8)) div 2;              ( Abscisse de départ X )
  SY     = (MAXY-(HAUTEUR*8)) div 2;              ( Ordonnée de départ Y )

type SPRITE = record                              ( Pour gérer les sprites  )
                spidp : SPIP;                  ( Pointe sur l'identificateur )
                deltax,      ( Déplacement horizontal pour pages 0 et 1)
                deltay : array [0..1] of integer;( Déplacement vertical)
              end;

var sprites    : array [1..nbspr] of SPRITE;
    page,                                          ( Page courante )
    lc,                                         ( Caractère courant )
    out        : byte;                 ( Mémorise l'indicateur de collision )
    x, y, i,                                    ( Compteurs d'itérations )
    dx, dy     : integer;                            ( Déplacements )
    vaisseauupp,
    vaisseaudnp : SPLP;                      ( Pointeur sur les sprites )
    ch         : char;
begin

  Randomize;           ( Initialise le générateur de nombres aléatoires )

          {-- Construit les motifs binaires des sprites --------------}

  vaisseauupp := CompileSprite( VaisseauMontant,   20, 0, 'A', 1 );
  vaisseaudnp := CompileSprite( VaisseauDescendant, 20, 0, 'A', 1 );

       {-- Remplit de caractères les deux pages graphiques -----------}

  for page := 0 to 1 do
    begin
      setpage( page );
      lc := 0;
      y := 0;
      while ( y < (MAXY+1)-8 ) do
        begin
          x := 0;
          while ( x < (MAXX+1)-8 ) do
            begin
              PrintChar( chr(lc and 127), x, y, lc mod 255, 0 );
              inc( lc );
              inc( x, 8 );
            end;
          inc( y, 12 );
        end;

     {-- Affiche l'avis de copyright ---------------------------------}

      Line( SX-1, SY-1, SX+LARGEUR*8, SY-1, 15 );
      Line( SX+LARGEUR*8, SY-1, SX+LARGEUR*8, SY+HAUTEUR*8,15 );
      Line( SX +LARGEUR*8, SY+HAUTEUR*8, SX-1, SY+HAUTEUR*8, 15 );
      Line( SX-1, SY+HAUTEUR*8, SX-1, SY-1, 15 );
      GrafPrint( SX, SY,    15, 4,
                                                          '  );
      GrafPrint( SX, SY+8,  15, 4,
                          ' S3240P.PAS - (c) 1990 by M. TISCHER ' );
      GrafPrint( SX, SY+16, 15, 4,
                                                          '  );
      GrafPrint( SX, SY+24, 15, 4,
                          '  Démonstration de sprites dans le   ' );
      GrafPrint( SX, SY+32, 15, 4,
                          '      mode 320*400 de la carte VGA   ' );
      GrafPrint( SX, SY+40, 15, 4,
                                                          '  );
    end;

    {-- Fabrique les différents sprites ------------------------------}

  for i := 1 to NBSPR do
    begin
      sprites[ i ].spidp := CreateSprite( vaisseauupp );
      repeat                                    ( Choisir les déplacements )
        dx := 0;
        dy := random(10) - 5;
      until ( dx <> 0 ) or ( dy <> 0 );

      sprites[ i ].deltax[0] := dx * 2;
      sprites[ i ].deltay[0] := dy * 2;
      sprites[ i ].deltax[1] := dx * 2;
      sprites[ i ].deltay[1] := dy * 2;

      x := ( 320 div NBSPR) * (i-1) ) + ((320 div NBSPR)-40) div 2;
      y := random( 200 - 40 );
      SetSprite( sprites[ i ].spidp, x, y, x - dx, y - dy );
    end;

{-- Déplace les sprites et les fait rebondir aux extrémités de l'écran )

  page := 1;                                      ( Commence en page 1 )
  while ( not keypressed ) do{Une frappe de touche interrompt la boucle}
    begin
      showpage( 1 - page );                      ( Affiche l'autre page )

      for i := 1 to nbspr do                     ( Parcourt les sprites )
        begin          ( Deplace les sprites et détecte les collisions )
          out := MoveSprite( sprites[i].spidp, page,
                             sprites[i].deltax[page],
                             sprites[i].deltay[page] );
          if ( ( out and OUT_TOP ) <> 0 ) or( Contact en haut/en bas ? )
             ( ( out and OUT_BOTTOM ) <> 0 ) then
            begin
                  {-- Oui, inverse le déplacement et change l'image --}

              sprites[i].deltay[page] := -sprites[i].deltay[page];
              if ( ( out and OUT_TOP ) <> 0 ) then
                sprites[i].spidp^.splookp := vaisseaudnp
              else
                sprites[i].spidp^.splookp := vaisseauupp;
            end;
          if ( ( out and OUT_LEFT ) <> 0 ) or(Contact à gauche/droite ?)
             ( ( out and OUT_RIGHT ) <> 0 ) then
            sprites[i].deltax[page] := -sprites[i].deltax[page];
        end;
      page := (page+1) and 1;          ( Passe de 1 à 0 et vice versa )
    end;

  ch := readkey;                        ( Attend une frappe de touche )
end;

{------------------------------------------------------------------------}
{--                        PROGRAMME PRINCIPAL                         --}
{------------------------------------------------------------------------}

begin
  if ( IsVga ) then                     ( Dispose-t-on d'une carte VGA )
    begin                                                ( Oui, on y va )
      Init320400;                       ( Initialise le mode graphique )
      Demo;
      Textmode( COBO );                          ( Ratblit le mode texte )
    end
  else
    writeln( 'S3240P.PAS  -  (c) 1990 by MICHAEL TISCHER'#13#10#10 +
             'Attention! Ce programme nécessite une carte VGA ');
end.
```

Listing : S3240PA.ASM

```
;*****************************************************************;
;*                        S 3 2 4 0 P A . A S M                  *;
;*---------------------------------------------------------------*;
;*   Fonction : Contient les routines pour travailler avec le    *;
;*              sprites dans le mode 320*400-256 de la carte VGA *;
;*---------------------------------------------------------------*;
;*   Auteur         : MICHAEL TISCHER                            *;
;*   Développé le   : 08.09.1990                                 *;
;*   Dernière MAJ   : 14.01.1991                                 *;
;*---------------------------------------------------------------*;
;*   Assemblage     : MASM /mx S3240PA;    ou   TASM -mx S3240PA *;
;*                    ... puis lier à S3240P.PAS                 *;
;*****************************************************************;


;== Constantes ==================================================

SC_INDEX      = 3c4h            ;Registre du contrôleur du séquenceur
SC_MAP_MASK   = 2                       ;Numéro du registre Map Mask
SC_MEM_MODE   = 4                 ;Numéro de registre de mode mémoire

GC_INDEX      = 3ceh          ;Registre d'index du contrôleur graphique
GC_READ_MAP   = 4                       ;Numéro du registre Read Map

PIXX          = 320                         ;Résolution horizontale

;== Segment de données ==========================================

DATA  segment word public

DATA  ends

;== Programme ===================================================

CODE      segment byte public                   ;Segment de programme

      assume cs:code, ds:data

;-- Déclarations publiques ---------------------------------------

public    copybuf2plane
public    copyplane2buf

;-----------------------------------------------------------------
;-- CopyBuf2Plane: Copie le contenu d'un buffer dans une zone
;--                rectangulaire d'un plan de bits
;-- Appel depuis TP: CopyBuf2Plane( bufptr   : pointer;
;--                                 verspage : byte;
;--                                 versx,
;--                                 versy    : integer;
;--                                 largeur,
;--                                 hauteur  : byte;
;--                                 bg       : bool  );

copybuf2plane proc near

sfr0      struc                         ;Structure d'accès à la pile
bp0       dw ?                                    ;Mémorise BP
ret_adr0  dw ?                        ;Adresse de retour à l'appelant
bg        dw ?                              ;Tient compte du fond
hauteur0  dw ?                              ;Hauteur de la zone
largeur0  dw ?                                    ;Largeur
versy     dw ?                          ;Ordonnée de destination
versx     dw ?                          ;Abscisse de destination
verspage  dw ?                             ;Page de destination
bufptr0   dd ?                             ;Pointeur sur buffer
sfr0      ends                                  ;Fin de la structure

fr    equ [ bp - bp0 ]          ;Adresse les éléments de la structure
bfr   equ byte ptr [ bp - bp0 ]   ;Adresse un élément de la pile octet

      push  bp                      ;Prépare l'adressage des paramètres
      mov   bp,sp                           ; par le registre BP

      push  ds
      cld                         ;Fixe le sens des traitements sur chaînes

            ;-- Calcule le segment d'accès à la mémoire d'écran ------

      mov   ah,0A0h               ;ES au début de la page de destination
      cmp   bfr.verspage,0                      ;Est-ce la page 0 ?
      je    cv0                                   ;Oui, AL est bon

      mov   ah,0A8h                           ;Non, page 1 en A800h

cv0:  xor   al,al                       ;Octet faible toujours nul
      mov   es,ax

      ;-- Calcule l'offset de la position de destination dans la page -

      mov   ax,PIXX / 4               ;DI sur position de destination
      mul   fr.versy
      mov   di,fr.versx
      mov   cx,di                             ; Mémorise l'abscisse X
      shr   di,1
      shr   di,1
      add   di,ax

            ;-- Prépare l'adressage du plan de bits ---------------

      mov   ah,1                          ;Le numéro du plan est pris
      and   cl,3                                ;comme masque binaire
      shl   ah,cl
      mov   dx,SC_INDEX                   ;Assure l'accès au plan
      mov   al,SC_MAP_MASK                          ; à traiter
      out   dx,ax

;-- Charge les compteurs pour la boucle de copie --------------------

      mov   dh,bfr.hauteur0                           ;DH = Lignes
      mov   dl,bfr.largeur0                           ;DL = Octets
      mov   bx,PIXX / 4               ;BX = offset sur ligne suivante
      sub   bl,dl
      xor   ch,ch                 ;Octet fort du compteur toujours 0

      lds   si,fr.bufptr0         ;DS:SI doit pointer sur le buffer

      cmp   bfr.bg,0                          ;Fond à respecter ?
      jne   cv2                         ;Oui, utilise une autre routine

            ;-- Routine de copie pour un plan de bits, ne respecte ---
                                                  ;-- pas le fond
cv1:  mov   cl,dl                             ;Nombre d'octets en CL

      rep movsb                                     ;Copie une ligne
      add   di,bx                           ;DI sur ligne suivante
      dec   dh                                 ;Reste une ligne?
      jne   cv1                              ;Oui---> on continue

      jmp   short cvend                       ;Non, buffer copié

                  ;-- Routine de copie pour plan de bits avec    -
                                    ;-- exploitation du buffer transmis
cv2:  mov   cl,dl                             ;Nombre d'octets en CL

cv3:  lodsb                                 ;Charge un octet du buffer
      cmp   al,255                              ;Octet de fond ?
      je    cv5                               ;Oui ---> ne pas copier
      stosb                           ;Non, le transfère en mémoire d'écran
      loop  cv3                               ;Traite l'octet suivant

cv4:  ;-- Avance le pointeur de la mémoire d'écran sur la ligne suivante

      add   di,bx                           ;DI sur ligne suivante
      dec   dh                                 ;Reste une ligne?
      jne   cv2                              ;Oui---> on continue
      jmp   short cvend                       ;Non, buffer copié

cv5:        ;-- Octet de fond à ne pas copier     ------------------

      inc   di                    ;Ne pas mettre cet octet en mémoire d'écran
      loop  cv3                       ;Reste-t-il un octet dans cette ligne ?
      jmp   cv4                                 ;Non ---> Ligne suivante

cvend: pop  ds
       pop  bp

      ret   16                          ;Retire les paramètres de la pile
                                                  ; et rend la main
copybuf2plane endp

;-----------------------------------------------------------------
;-- CopyPlane2Buf: Copie une zone rectangulaire d'un plan de bits dans
;--                un buffer
;-- Appel depuis TP: CopyPlane2Buf( bufptr : pointer;
;--                                 depage: byte;
;--                                 dex,
;--                                 dey    : integer;
;--                                 largeur,
;--                                 hauteur  : byte );

copyplane2buf proc near
```

```
;
;sfr1    struc                          ;Structure d'accès à la pile
;bp1     dw ?                                    ;Mémorise BP
;ret_adr1 dw ?                         ;ADresse de retour à l'appelant
;hauteur1 dw ?                          ;Hauteur de la zone en pixels
;largeur1 dw ?                          ;Largeur de la zone en pixels
;dey      dw ?                                  ;Ordonnée d'origine
;dex      dw ?                                  ;Abscisse d'origine
;depage   dw ?                                  ;Page d'origine
;bufptr1  dd ?                                  ;Pointeur sur buffer
;sfr1    ends                                   ;Fin de la structure
;
;fr      equ [ bp - bp1 ]              ;Adresse les éléments de la structure
;bfr     equ byte ptr [ bp - bp1 ]      ;Adr éléments de pile comme octet
;
;       push  bp                      ;Prépare l'adressage des paramètres
;       mov   bp,sp                           ; par le registre BP
;
;       push  ds
;
;             ;-- Calcule le segment d'accès à la mémoire d'écran -----
;
;       mov   ah,0A0h                 ;ES au début de la page d'origine
;       cmp   bfr.depage,0                    ;Est-ce la page 0 ?
;       je    cc0                             ;Oui AL est bon
;
;       mov   ah,0A8h                         ;Non, la page 1 en A800h
;
;cc0:   xor   al,al                    ;Octet faible toujours nul
;       mov   ds,ax
;
;             ;-- Forme l'offset dans la page à lire ----------------
;
;       mov   ax,PIXX / 4               ;SI sur la position d'origine
;       mul   fr.dey
;       mov   si,fr.dex
;       mov   cx,si                   ;Mémorise également l'abscisse en CX
;       shr   si,1
;       shr   si,1
;       add   si,ax
;
;                   ;-- Fixe le plan de bits à adresser ---------------
;       and   cl,3                          ;Calcule en AH les masque binaire
;       mov   ah,cl                                 ;pour le plan concerné
;       mov   al,GC_READ_MAP        ;Transfère en AL le numéro du registre
;       mov   dx,GC_INDEX ;Charge l'adresse index du contrôleur graphique
;       out   dx,ax                           ;Charge le registre Read Map
;
;             ;-- Charge les compteurs pour la boucle de copie----------
;
;       mov   dh,bfr.hauteur1                               ;DH = Lignes
;       mov   dl,bfr.largeur1                               ;DL = octets
;       mov   bx,PIXX / 4                 ;BX = offset sur ligne suivante
;       sub   bl,dl
;       xor   ch,ch                       ;Octet haut du compteur toujours 0
;
;       les   di,fr.bufptr1               ;ES:DI doit pointer sur le buffer
;
;                        ;-- Routine de copie pour un plan de bits - ----
;                                          ;-- sans tenir compte du fond
;cc1:   mov   cl,dl                                  ;Nombre d'octets en CL
;
;       rep movsb                                         ;Copie une ligne
;       add   si,bx                               ;SI sur ligne suivante
;       dec   dh                                    ;Reste une ligne ?
;       jne   cc1                                     ;Oui on continue
;
;       pop   ds
;       pop   bp
;
;          ret  14
;
;copyplane2buf endp
;
;;-- Fin ---------------------------------------------------------------
;
;CODE   ends                                       ;Fin du segment de code
;       end                                          ;Fin du programme
;
```

Listing : V3240PA.ASM (déjà mentionné en 4.8.7)

Sprites en mode graphique EGA et VGA avec une résolution de 640*350 points et 16 couleurs

Celui qui attache plus d'importance à une haute résolution plutôt qu'au nombre de couleurs affichables sur l'écran se trouve fort bien servi avec le mode graphique 640*350 points. Ce mode est à la fois reconnu par les cartes VGA et EGA ce qui n'est pas le cas du mode haute résolution en 640*480 points. Mais il est de toute façon à écarter de la programmation Sprite puisqu'il n'est plus possible d'afficher deux pages écran dans ce mode.

En revanche en mode 640*350 points, la capacité de la RAM vidéo reste inépuisable malgré l'utilisation de deux pages écran. Dans ce mode, les deux pages écran réunies n'occupent que 219 Ko de la RAM vidéo qui offre en tout 256 Ko. Il reste exactement 38 144 octets inutilisés ce qui correspond à près de 76 000 points ou une résolution supplémentairede 640*120 points. Selon le même procédé que celui du mode 320*200 points, on peut naturellement utiliser la partie inoccupée de la RAM vidéo pour stocker les modèles de Sprites et les buffers d'arrière-plan.

Selon le nombre de modèles Sprite et les Sprites affichés effectivement sur l'écran, il se peut que l'espace devienne toutefois restreint dans la RAM vidéo. Cela s'explique par le fait qu'on a besoin maintenant de huit copies différentes par modèle Sprite au lieu des quatre en mode 320*200 points.

Il ne faut pas s'étonner si les Sprites du mode 640*350 points suivent les traces de leurs prédécesseurs du mode 320*400 points et gèrent les modèles et fonds de Sprites dans la mémoire principale. Mais les routines servant à disposer les Sprites et échanger les zones écran entre la mémoire principale et la RAM vidéo sont nettement différentes de celles de leurs prédécesseurs.

En modes 16 couleurs, l'organisation de la RAM vidéo est complètement différente de celle des modes 256 couleurs comme vous avez pu le constater dans les sections précédentes. Au lieu de représenter un point par un octet, ici chaque octet de la RAM vidéo contient huit points. Sachant que chacun des huit points d'un tel octet ne dispose que d'un bit, il est évident que cela ne suffit pas à recevoir un code couleur compris entre 0 et 15. C'est ainsi que se crée la fameuse commutation "3D" des quatre plans de bits où quatre octets consécutifs issus des quatre plans de bits reçoivent les informations de couleur pour huit points.

Cette organisation de la RAM vidéo qui rend quelque peu contradictoire et ralentit par exemple l'accès individuel à des points déterminés comporte naturellement de nombreuses implications dans le fonctionnement des routines des modules S6435C.C (version C) et S6435P.PAS (version Pascal) qui affichent les Sprites sur l'écran en mode 640*350 points.

Sur le plan interne, la largeur des Sprites est toujours arrondie à un multiple de huit pour que les bits des marges gauche et droite d'un Sprite ne soient pas exclus de l'opération lors des divers accès à la RAM vidéo. Il faut donc d'abord charger chaque octet vidéo dans la marge d'un Sprite puis isoler les bits souhaités et ensuite le réécrire lors d'un accès en écriture. Tout cela risque de durer un long moment. Il en est de même de la programmation permanente du registre Bit Mask, la seule et unique alternative à cette procédure.

Cette action d'arrondir reste transparente à l'utilisateur des routines telles que COMPILESPRITE, CREATESPRITE et MOVESPRITE parce qu'elle s'exécute sur le plan interne sans apparaître à l'extérieur. De même, il ne remarque pas que huit copies d'un modèle Sprite ont été créées sur le plan interne en raison d'un problème qui intervient ici comme dans le mode 320*200 points.

Si on commence notamment à coder le modèle Sprite en système binaire à partir du bit 7 du premier octet par exemple, ce modèle ne peut alors être copié tel quel que dans la zone écran commençant à la coordonnée X et qui est un multiple de huit. Pour toute zone écran commençant à une coordonnée X dont le modulo par huit donne une valeur différente de 0, les différents bits doivent être alignés à droite par rapport à cette valeur. Une telle action n'est pas souple parce qu'elle tend à ralentir l'exécution d'un programme en nécessitant surtout une programmation poussée.

Dans la routine COMPILESPRITE de S6435C.C et S6435P.PAS, un modèle Sprite a donc été créé huit fois en le décalant à chaque fois d'un point vers la droite. Si un Sprite doit être affiché en un endroit précis de l'écran, on fait toujours appel au même modèle qui convient à la coordonnée X spécifiée.

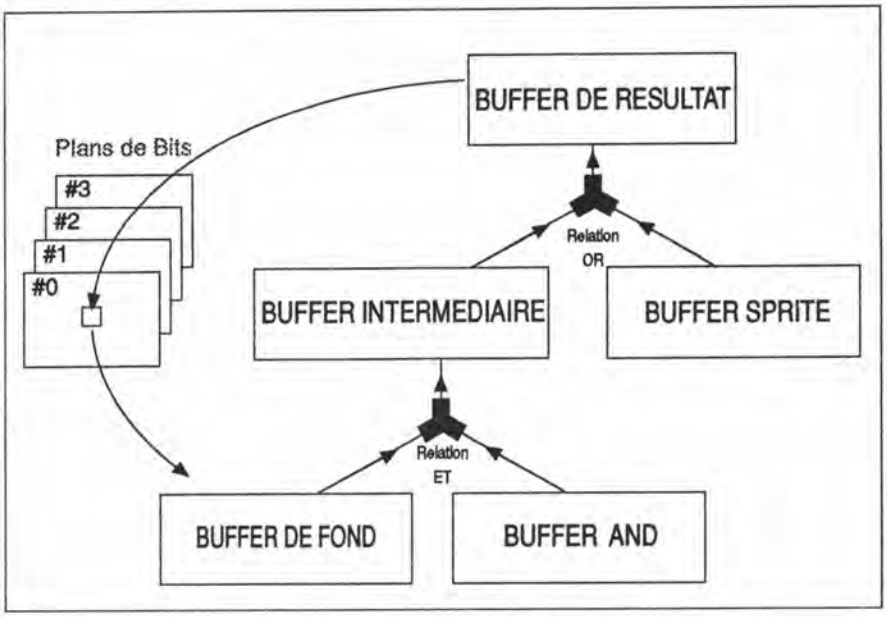
Ces huit modèles Sprite sont stockés sous la forme de buffers pixel c'est-à-dire que le modèle Sprite est d'abord conçu dans la RAM vidéo tout comme les modules pour la programmation Sprite en mode 320*400 points. Il est ensuite chargé dans un buffer pixel au moyen d'un appel de GETVIDEO. Contrairement aux programmes Sprite du mode 320*400 points, cette action s'opère ici huit fois pour chaque Sprite ce qui explique le besoin important en mémoire principale.

Pour un Sprite d'une ampleur de 20*30 points, ce qui est insignifiant dans un mode à si haute résolution, on obtient une largeur interne de

```
int((20 + 7 + 7)/8)*8 = 32
```

points après avoir revalorisé la largeur à sept points puis arrondi à un multiple de huit. Sachant que chaque point nécessite quatre bits, on obtient 16 octets par ligne de points. En multipliant cela par 20, il en résulte d'ores et déjà 640 octets rien que pour le buffer pixel sans prendre en compte l'information d'étaten PIXBUF. Si on considère maintenant qu'un tel buffer est réclamé huit fois par modèle Sprite, on atteint facilement 5 Ko. Il ne faut oublier que les deux buffers d'arrière-plan qui consomment plus d'1 Ko par Sprite ne sont pas pris en compte.

Ce n'est pas tout. Il s'y ajoute les buffers AND qui sont à définir pour chacun des huit modèles Sprite. Ils sont également utiles en mode 640*350 points parce qu'un Sprite ne doit être que partiellement transparent pour qu'il ne soit pas reconnu comme une zone écran rectangulaire. En raison de l'organisation de la RAM vidéo dans ce mode graphique, il s'agit là d'une occasion particulièrement délicate qui ne peut être mise à profit qu'à l'aide d'un processus en trois étapes. Ce dernier doit être répété pour chaque octet de la RAM vidéo situé dans la zone du Sprite. Ce processus comprend d'abord le chargement de l'octet à traiter, puis la programmation du registre Bit Mask pour ne lire que les points souhaités et ensuite l'écriture de la valeur souhaitée.



*Fusion d'un arrière-plan Sprite avec un modèle Sprite en mode 640*350 points à travers un buffer AND*

Cette procédure risque de durer longtemps en raison du nombre important d'octets par Sprite et de Sprites par page écran. Nous avons donc choisi une autre méthode dans les programmes qui suivent.

On suppose que non seulement le modèle mais aussi l'arrière-plan du Sprite existent déjà avant l'écriture d'un Sprite parce que l'arrière-plan a dû naturellement être sauvegardé au préalable. On peut alors fusionner ces deux buffers octet par octet avant même l'accès à la RAM vidéo pour obtenir une mixture composée de points d'arrière-plan et de points de Sprite pour chaque octet. Ainsi se crée l'octet qui peut finalement être inscrit dans la RAM vidéo sans être obligé de masquer des bits déterminés.

Dans les octets du buffer d'arrière-plan, il ne faut cependant modifier que les bits ne correspondant pas à un point d'arrière-plan. Ainsi, seule la couleur des points d'arrière-plan reste inchangée, tous les autres points étant recouverts par le Sprite. A cet effet, il faut établir une relation avec le masque AND.

Prenez par exemple un octet quelconque du modèle Sprite avec ses huits points. Dans cet octet, il faut que le point situé complètement à gauche (représenté par le bit 7) soit transparent. Dans le masque AND de cet octet, ce bit est réglé sur 1 alors que tous les autres affichent 0. Les deux étapes suivantes expliquent cette situation. Lors de la combinaison des trois buffers (modèle Sprite, buffer d'arrière-plan et buffer AND), l'octet d'arrière-plan est en effet relié en premier avec l'octet correspondant du buffer AND à travers une opération logique AND.

Le résultat est que seul le bit d'arrière-plan (7) reste inchangé alors que tous les autres sont mis sur 0. A l'étape suivante, ce résultat est combiné avec l'octet de premier plan à travers une opération logique OR. Dans un tel octet de premier plan, toutes les positions de bits correspondant à un point transparent contiennent en principe la valeur 0. Les autres bits contiennent un des quatre bits composant la couleur du point Sprite.

La couleur du point Sprite concerné est ainsi incluse dans les points écran recouverts à présent par le Sprite. Quant aux bits correspondant aux points d'arrière-plan, ils restent inchangés.

Il faut en outre que l'opération soit menée intégralement sur les quatre plans de bits. Le masque AND est d'ailleurs identique pour les divers plans de bits parce que ce sont toujours les mêmes points qui doivent faire partie du Sprite ou rester transparents. La taille d'un buffer AND qui n'est valable que pour l'un des huit modèles Sprite, s'élève à

```
Largeur * Hauteur /8 octets.
```

Les buffers AND nécessaires aux modèles Sprite se créent comme les modèles Sprite eux-mêmes en COMPILESPRITE. Dans ce cas, les buffers sont traités comme un énorme champ de bits qui est d'abord vide. Lorsque le point concerné doit rester transparent, un 1 est inséré à la fin de ce champ lors du parcours du tableau Sprite. En revanche, un 0 représente un point Sprite devant réécrire l'arrière-plan.

La combinaison de l'arrière-plan Sprite, du modèle Sprite et du buffer AND est réalisée grâce à une routine Assembleur des modules S6435CA.ASM ou S6435PA.ASM. Elle porte un nom assez long mais significatif MERGEANDCOPYBUF2VIDEO parce qu'elle fusionne les buffers et les copie ensuite dans la RAM vidéo. Cette opération s'effectue néanmoins en parallèle avec l'accès à la RAM vidéo. Autrement dit, un octet issu de ces trois buffers est d'abord combiné pour être ensuite inscrit directement dans la RAM vidéo.

L'opération se réalise en principe automatiquement quatre fois dans MERGEANDCOPYBUF2VIDEO pour pouvoir traiter successivement les quatre plans de bits. Il en est de même des routines Assembleur COPYBUF2VIDEO et COPYVIDEO2BUF qui sont utiles en mode 320*400 points au lieu des routines COPYBUF2PLANE et COPYVIDEO2PLANE des modules Assembleur.

Il vous est possible de traiter automatiquement les quatre plans de bits puisque le nombre d'octets traités reste toujours identique en raison de l'utilisation permanente des zones d'écran dont la largeur est toujours un multiple de huit. Par conséquent, COPYVIDEO2BUF copie automatiquement la zone écran spécifiée depuis les quatre plans de bits vers le buffer défini. De même, COPYBUF2VIDEO charge la zone écran dans les quatre plans de bits avec le contenu du buffer spécifié.

En dehors des différences de concept, il n'y a rien de plus à ajouter à propos des divers programmes Sprite en mode 640*350 points. Ils ressemblent beaucoup à leurs prédécesseurs qui ont été largement décrits au cours de ce chapitre.

Pour conclure ce chapitre, voici les listings des programmes Sprite en mode graphique 640*350 points avec 16 couleurs. Si vous ne parvenez pas à élucider certaines questions, n'hésitez pas à consulter les listings qui vous apporteront certainement les réponses nécessaires. Vous y trouverez également de nombreuses routines utiles pour programmer vos prores Sprite.

Listing : S6435C.C

```
/*****************************************************************
*                          S6435C.C                              *
******------------------------------------------------------------**
* Fonction          : Montre comment travailler avec des sprites  *
*                     dans le mode graphique 640*350 des cartes EGA *
*                     et VGA avec 16 couleurs et deux pages d'écran *
*                     Le programme utilise les routines en assembleur *
*                     des modules S6435CA.ASM et V16COLCA.ASM     *
******------------------------------------------------------------**
* Auteur       : MICHAEL TISCHER                                  *
* Développé le :  5.12.1990                                       *
* Dernière MAJ : 14.02.1992                                       *
******------------------------------------------------------------**
* Modèle mémoire   : SMALL                                        *
******------------------------------------------------------------**
* (MICROSOFT C)                                                   *
* Compilation      : CL /AS s6435c.c s6435ca v16colca              *
******------------------------------------------------------------**
* (BORLAND TURBO C)                                               *
* Compilation      : Utiliser un fichier de projet avec le contenu *
*                    suivant:                                     *
*                    s6435c.c                                     *
*                    v16colca.obj                                 *
*                    s6435ca.obj                                  *
******------------------------------------------------------------**
* Appel            : s6435c                                       *
*****************************************************************/

#include <dos.h>
#include <stdarg.h>
#include <stdio.h>
#include <string.h>
#include <stdlib.h>
#include <conio.h>

/*-- Déclarations dépendantes du compilateur-------------------------*/

#ifdef __TURBOC__
  #include <alloc.h>
#else
  #include <malloc.h>
  #define random(x) ( rand() % (x+1) )          /* Fonction aléatoire */
#endif

/*-- Déclarations de types ------------------------------------------*/

typedef unsigned char BYTE;
typedef BYTE BOOL;

typedef struct {    /* Buffer de pixels pour GetVideo() et PutVideo() */
        BYTE largeurbyte,           /* Largeur de la zone en octets */
             hauteur;                          /* Nombre de lignes */
        int  pixblen;                /* taille du buffer de pixels */
        void *pixbptr;           /* Pointeur sur le buffer de pixels */
             } PIXBUF;
typedef PIXBUF *PIXPTR;          /* Pointeur sur un buffer de pixels */

typedef struct {                              /* Image d'un sprite */
        BYTE  largeur,                         /* Largeur totale */
              hauteur;                /* Hauteur en lignes de pixels */
        void  *bmskp[8];       /* Pointeur sur le masque binaire AND */
        PIXPTR pixmp[8];        /* Pointeur sur définition de pixels */
             } SPLOOK;

typedef struct {                        /* Descripteur de sprite (ID) */
        SPLOOK *splookp;                     /* Pointeur sur l'image */
        int    x[2], y[2];            /* Coordonnées en pages 0 et 1 */
        PIXPTR hgptr[2];            /* Pointeur sur le buffer du fond */
             } SPID;

typedef struct {                          /* décrit un champ de bits */
        BYTE *champptr, /* Pointe sur le buffer avec le champ de bits */
             *courptr,                /* Pointeur sur l'octet courant */
              bitcour,                /* Bit courant dans octet courant */
              bytecour;                  /* valeur de l'octet courant */
             } CHAMPBITS;
typedef CHAMPBITS *CBPTR;             /* Pointeur sur un champ de bits */

/*-- Références externes aux routines en assembleur-----------------*/

extern void init640350( void );
extern void setpix( int x, int y, unsigned char couleur);
extern BYTE getpix( int x, int y );
extern void setpage( int page );
extern void showpage( int page );
extern void far * getfontptr( void );

extern void copybuf2video( BYTE *bufptr, BYTE page,
                           int ax, int ay, BYTE largeur,
                           BYTE hauteur );
extern void copyvideo2buf( BYTE *bufptr, BYTE page,
                           int dex, int dey, BYTE largeur,
                           BYTE hauteur );
extern void mergeandcopybuf2video( void * spribufptr, void * hgbufptr,
                                   void * andbufptr, BYTE page,
                                   int  ax, int  ay,
                                   BYTE largeur, BYTE hauteur );

/*-- Constantes -----------------------------------------------------*/

#define TRUE  ( 0 == 0 )
#define FALSE ( 0 == 1 )

#define MAXX 639                          /* Coordonnées maximales */
#define MAXY 349

#define OUT_LEFT   1    /* Indicateurs de collision dans SpriteMove() */
#define OUT_TOP    2
#define OUT_RIGHT  4
#define OUT_BOTTOM 8
#define OUT_NO     0                          /* Pas de collision */

#define EGA        0                           /* Types de cartes */
#define VGA        1
#define NINI       2

#define ALLOCBUF ((PIXPTR) 0)          /* GetVideo(): alloue un buffer */

/*****************************************************************
*  IsEgaVga : Teste la présence d'une carte EGA ou VGA           *
******------------------------------------------------------------**
*  Entrée   : néant                                              *
*  Sortie   : EGA, VGA ou NINI                                   *
*****************************************************************/

BYTE IsEgaVga( void )
{
 union REGS  Regs;                /* Registes pour gérer l'interruption */

 Regs.x.ax = 0x1a00;              /* La fonction 1Ah n'existe qu'en VGA */
 int86( 0x10, &Regs, &Regs );
 if( Regs.h.al == 0x1a )                    /* Est-elle disponible ? */
   return VGA;
 else
 {
   Regs.h.ah = 0x12;                          /* Appelle l'option 10h */
   Regs.h.bl = 0x10;                           /* de la fonction 12h */
   int86(0x10, &Regs, &Regs );        /*Déclenche l'interruption vidéo */
   return (BYTE) (( Regs.h.bl != 0x10 ) ? EGA : NINI);
 }
}

/*****************************************************************
*  PrintChar : Ecrit un caractère en dehors de la zone visible   *
*              de la mémoire d'écran                             *
******------------------------------------------------------------**
```

```
/*  Entrée  :  caractere = caractère à afficher                    */
/*             x, y      = Coordonnées du coin supérieur gauche    */
/*             cc        = Couleur du caractère                    */
/*             cf        = Couleur du fond                         */
/*  Info    : Le caractère est dessiné dans une matrice de 8*8 pixels */
/*            sur la base du jeu de caractères 8*8 en ROM          */
/*****************************************************************/

void PrintChar( char caractere, int x, int y, BYTE cc, BYTE cf )
{
 typedef BYTE   CARDEF[256][8];     /* Structure du jeu de caractères */
 typedef CARDEF far *CARPTR;        /* Pointe sur un jeu de caractères */

 BYTE     i, k,                             /* Compteur d'itérations */
          masque;        /* Masque binaire pour dessiner le caractère */

 static   CARPTR fptr = (CARPTR) 0;       /* Jeu de caractères en ROM */

 if( fptr == (CARPTR) 0 )       /* A-t-on déjà déterminé ce pointeur ? */
   fptr = getfontptr();      /*Non, détermine avec fonction assembleur */

  /*- Dessine le caractère pixel par pixel ---------------------------*/

 if( cf == 255 )                              /* Caractère transparent ? */
   for ( i = 0; i < 8; ++i )   /* Oui, dessine que pixels du 1er plan */
   {
     masque = (*fptr)[caractere][i];          /* Motif bin pour ligne */
     for( k = 0; k < 8; ++k, masque <<= 1 )  /* Parcourt les colonnes */
       if( masque & 128 )                      /* Pixel à dessiner ? */
         setpix( x+k, y+i, cc );                          /* Oui */
   }
 else                                      /* Non dessine chaque pixel */
   for ( i = 0; i < 8; ++i )                  /* Parcourt les lignes */
   {
     masque = (*fptr)[caractere][i];          /* Motif bin pour ligne */
     for( k = 0; k < 8; ++k, masque <<= 1 )  /* Parcourt les colonnes */
       setpix( x+k, y+i, (BYTE) (( masque & 128 ) ? cc : cf) );
   }
}

/*****************************************************************/
/*  Line: Trace un segment dans la fenêtre graphique en appliquant */
/*        l'algorithme de Bresenham                                */
/*---------------------------------------------------------------*/
/*  Entrées : X1, Y1  = Coordonnées de l'origine                 */
/*            X2, Y2  = Coordonnées de l'extrémité terminale     */
/*            COULEUR = couleur dusegment                        */
/*****************************************************************/

/*-- Fonction accessoire pour échanger deux variables entières -------*/

void SwapInt( int *i1, int *i2 )
{
 int   dummy;

 dummy = *i2;  *i2  = *i1;  *i1  = dummy;
}

/*-- Procédure principale ---------------------------------------------*/

void Line( int x1, int y1, int x2, int y2, BYTE couleur )
{
 int d, dx, dy,
     aincr, bincr,
     xincr, yincr,
     x, y;

 if( abs(x2-x1) < abs(y2-y1) )     /* Sens du parcours : axe X ou Y ? */
 {                                                      /* Par Y */
   if ( y1 > y2 )                          /* y1 plus grand que y2? */
   {
     SwapInt( &x1, &x2 );                  /* Oui échange X1 et X2, */
     SwapInt( &y1, &y2 );                             /* Y1 et Y2 */
   }

   xincr = ( x2 > x1 ) ?  1 : -1;          /* Fixe le pas horizontal */

   dy     = y2 - y1;
   dx     = abs( x2-x1 );
   d      = 2 * dx - dy;
   aincr  = 2 * (dx - dy);
   bincr  = 2 * dx;
   x      = x1;
   y      = y1;

   setpix( x, y, couleur );              /* dessine le premier pixel */
   for( y=y1+1; y<= y2; ++y )               /* Parcourt l'axe des Y */
   {
     if( d >= 0 )
     {
       x += xincr;
       d += aincr;
     }
     else
       d += bincr;
     setpix( x, y, couleur );
   }
 }
 else                                                   /* par X */
 {
   if( x1 > x2 )                              /* x1 plus grand que x2? */
   {
     SwapInt( &x1, &x2 );                     /* Oui, échange X1 et X2 */
     SwapInt( &y1, &y2 );                                /* Y1 et Y2 */
   }

   yincr = ( y2 > y1 ) ? 1 : -1;             /* Fixe le pas vertical */

   dx     = x2 - x1;
   dy     = abs( y2-y1 );
   d      = 2 * dy - dx;
   aincr  = 2 * (dy - dx);
   bincr  = 2 * dy;
   x      = x1;
   y      = y1;

   setpix( x, y, couleur );              /* Dessine le premier pixel */
   for( x=x1+1; x<=x2; ++x )                /* Parcourt l'axe des X */
   {
     if ( d >= 0 )
     {
       y += yincr;
       d += aincr;
     }
     else
       d += bincr;
     setpix( x, y, couleur );
   }
 }
}

/*****************************************************************/
/*  GrafPrintf: Affiche une chaine formatée sur l'écran graphique */
/*---------------------------------------------------------------*/
/*  Entrées : X, Y   = Coordonnées de départ (0 - ...)           */
/*            CC     = Couleur des caractères                    */
/*            CF     = Couleur du fond  (255 = transparent)      */
/*            STRING = Chaine avec indications de formatage      */
/*            ...    = Expressions comme pour printf             */
/*****************************************************************/

void GrafPrintf( int x, int y, BYTE cc, BYTE cf, char * string, ... )
{
 va_list  parameter;     /* Liste de paramètres pour les macros VA_...*/
 char     affichage[255],           /* Buffer pour la chaine formatée */
          *cp;

 va_start( parameter, string );          /* Convertit les paramètres */
 vsprintf( affichage, string, parameter );              /* Formate */
 for( cp = affichage; *cp; ++cp, x+= 8 )       /* Affiche la chaine */
   PrintChar( *cp, x, y, cc, cf );          /* formatée par PrintChar */
}

/*****************************************************************/
/*  GetVideo: Charge le contenu d'une zone rectangulaire de la mémoire */
/*            d'écran dans un buffer                             */
/*---------------------------------------------------------------*/
/*  Entrées : PAGE   = Page d'écran   (0 ou 1)                   */
/*            X1, Y1 = Coordonnées de départ                     */
/*            LARGEUR= Largeur de la zone rectangulaire en pixels */
/*            HAUTEUR= Hauteur de la zone rectangulaire en pixels */
/*            BUFPTR = Pointeur sur le buffer de pixels qui va   */
/*                     mémoriser les informations                */
/*  Sortie  : Pointeur sur le buffer créé qui contient la zone indiquée */
/*  Info    : Si on donne au paramètre BUFPTR la valeur ALLOCBUF,  */
/*            un nouveau buffer de pixels est alloué sur le tas et */
/*            retourné. Ce buffer peut être transmis lors d'un nouvel */
/*            appel si l'ancien contenu ne mérite pas d'être préservé */
/*            et si la taille de la zone n'a pas changé          */
/*            La zone spécifiée doit commencer à une abscisse divisible */
/*            par huit et s'étendre sur un nombre de pixels multiple */
/*            de huit                                            */
/*****************************************************************/

PIXPTR GetVideo( BYTE page, int x1, int y1, BYTE largeur, BYTE hauteur,
                 PIXPTR bufptr )
{
 if( bufptr == ALLOCBUF )                 /* Pas de buffer transmis ? */
 {                                                /* Non, on alloue */
   bufptr = malloc( sizeof( PIXBUF ) );           /* Crée le buffer */
   bufptr->pixbptr = malloc( (largeur*hauteur) / 2 );
   bufptr->hauteur = hauteur;             /* Hauteur du buffer en lignes */
   bufptr->largeurbyte = largeur / 8;  /* Largeur d'1 ligne en octets */
   bufptr->pixblen = (largeur*hauteur) / 2;   /* Taille totale buffer */
 }
```

```
| copyvideo2buf( bufptr->pixbptr, page, x1, y1, largeur / 8, hauteur );
| return bufptr;                          /* Renvoie un pointeur sur le buffer */
|}
|
|/*****************************************************************
|*  PutVideo: Réécrit dans la mémoire d'écran le contenu d'une zone  *
|*            rectangulaire préalablement sauvegardée par GetVideo()  *
|**---------------------------------------------------------------**
|*  Entrées : BUFPTR = Pointeur renvoyé par Getvideo() et référençant  *
|*                     un buffer de pixels                           *
|*            PAGE   = Page d'écran(0 ou 1)                          *
|*            X1, Y1 = Coordonnées de début                          *
|*  Info    : Le buffer de pixels n'est pas effacé par cette procédure. *
|*            cette tâche étant menée à bien par FreePixBuf()        *
|*            L'abscisse X spécifiée doit être un multiple de huit   *
|*****************************************************************/
|
|void PutVideo( PIXPTR bufptr, BYTE page, int x1, int y1 )
|{
| copybuf2video( bufptr->pixbptr, page, x1, y1,
|                bufptr->largeurbyte, bufptr->hauteur );
|}
|
|/*****************************************************************
|*  FreePixBuf: Efface un buffer de pixels alloué sur le tas       *
|*              par GetVideo                                       *
|**---------------------------------------------------------------**
|*  Entrée : BUFPTR = Pointeur renvoyé par Getvideo() et référençant  *
|*                    un buffer de pixels                          *
|*****************************************************************/
|
|void FreePixBuf( PIXPTR bufptr )
|{
| free( bufptr->pixbptr );
| free( bufptr );
|}
|
|/*****************************************************************
|*  CreateSprite: Crée un sprite à l'aide d'un motif de pixels     *
|*                préalablement compilé                            *
|**---------------------------------------------------------------**
|*  Entrée  : SPLOOKP = Pointeur sur la structure de donnée produite  *
|*                      par CompileSprite()                        *
|*  Sortie  : Pointeur sur la structure du sprite créée            *
|*  Info    : la mémorisation du fond du sprite nécessite          *
|*            deux zones contiguës de la taille du sprite          *
|*****************************************************************/
|
|SPID *CreateSprite( SPLOOK *splookp )
|{
| SPID   *spidp;            /* Pointeur sur la structure du sprite créée */
|
| spidp = (SPID *) malloc( sizeof(SPID) );     /* Alloue de la mémoire */
| spidp->splookp = splookp;   /* Reporte les données dans la structure */
|
| /*- Crée desx buffers de fond en sauvegardant par GetVideo une zone -*/
|                                          /*- de la mémoire d'écran -*/
|
| spidp->hgptr[0] = GetVideo( 0, 0, 0, splookp->largeur,
|                             splookp->hauteur, ALLOCBUF );
| spidp->hgptr[1] = GetVideo( 0, 0, 0, splookp->largeur,
|                             splookp->hauteur, ALLOCBUF );
| return spidp;       /* Renvoie un pointeur sur la structure du sprite */
|}
|
|/*****************************************************************
|*  CBInit: Crée un champ de bits et prépare son traitement        *
|**---------------------------------------------------------------**
|*  Entrée  : NBBIT = Nombre de bits à placer dans le champ        *
|*  Sortie  : Pointeur sur un descripteur de champ de bits         *
|*****************************************************************/
|
|CBPTR CBInit( int NbBit )
|{
| CBPTR   cbptr;                  /* Pointeur sur le descripteur généré */
|
| cbptr = malloc( sizeof( CHAMPBITS ) );          /* Crée le descripteur */
|
|          /*-- Création et initialisation du champ de bits ----------*/
|
| cbptr->champptr = cbptr->courptr  = malloc( ( NbBit + 7 ) / 8 );
| cbptr->bitcour  = cbptr->bytecour = 0;
|
| return cbptr;                  /* Retourne un pointeur sur le descripteur */
|}
|
|/*****************************************************************
|*  CBAppendBit: Ajoute un bit à un champ de bits                  *
|**---------------------------------------------------------------**
|*  Entrée  : BFID = Pointeur sur le descripteur de champ de bits  *
|*                   renvoyé par CBInit()                          *
|*            BIT  = Valeur du bit à ajouter (0 ou 1)             *
|*  Sortie  : néant                                                *
|*****************************************************************/
|
|void CBAppendBit( CBPTR bfid, BYTE bit )
|{
| bfid->bytecour |= bit;                    /* Place le bit e, position 0 */
| if( bfid->bitcour == 7 )                        /* octet rempli ? */
| {                                                        /* Oui */
|   *(bfid->courptr++) = bfid->bytecour;       /* Octet dans le buffer */
|   bfid->bytecour = bfid->bitcour = 0;                 /* Remise à 0 */
| }
| else                                        /* L'octet n'est pas rempli */
| {
|   ++bfid->bitcour;                            /* Traite un bit de plus */
|   bfid->bytecour <<= 1;                     /* Décale le masque binaire */
| }
|}
|
|/*****************************************************************
|*  CBEnd : Clôture l'exploitation d'un champ de bits et efface son  *
|*          descripteur sans effacer le champ proprement dit       *
|**---------------------------------------------------------------**
|*  Entrée  : BFID = Pointeur sur le descripteur de champ de bit renvoyé*
|*                   par CBInit()                                  *
|*  Sortie  : Pointeur sur le champ proprement dit dont le buffer peut  *
|*            être libéré par FREE().                              *
|*****************************************************************/
|
|void *CBEnd( CBPTR bfid )
|{
| void *retptr;        /* Pointeur sur le champ de bits proprement dit */
|
| if( bfid->bitcour )                          /* Dernier octet rempli ? */
|   *bfid->courptr = bfid->bytecour << (7 - bfid->bitcour );    /* Non */
|
| retptr = bfid->champptr;              /* Mémorise ptr sur champ de bits */
| free( bfid );                                  /* Efface le descripteur */
|
| return retptr;          /* Retourne un pointeur sur le champ de bits */
|}
|
|/*****************************************************************
|*  CompileSprite: Crée le motif binaire d'un sprite à l'aide d'une  *
|*                 définition connus au moment de l'exécution      *
|**---------------------------------------------------------------**
|*  Entrées : BUFP    = Pointeur sur un tableau de pointeurs référençant*
|*                      des chaînes de caractères qui représentent le  *
|*                      sprite                                     *
|*            HAUTEUR = Hauteur du sprite et nombre de chaînes de    *
|*                      caractères                                 *
|*  Info    : Dans le motif transmis, un espace correspond à un pixel du*
|*            fond, A au code de couleur 0, B à 1, C à 2, etc...   *
|*****************************************************************/
|
|SPLOOK *CompileSprite( char **bufp, BYTE hauteur )
|{
| BYTE  slargeur,          /* Longueur des chaînes = largeur du motif */
|       largeurspr,                          /* Largeur des sprites */
|       c,                                   /* Mémorise un caractère */
|       i, k, l, y;                          /* Compteurs d'itérations */
| SPLOOK *splookp;       /* Pointeur sur la structure du sprite créée */
| PIXPTR tpix;                    /* Mémorise temporairement le fond */
| CBPTR  cbptr;
|          /*-- Crée une structure d'image et la remplit ------------*/
|
| splookp    = (SPLOOK *) malloc( sizeof(SPLOOK) );
| slargeur   = (BYTE) strlen( *bufp );   /* Long. chaînes = larg. logo */
| largeurspr = ( ( slargeur + 7 + 7 ) / 8 ) * 8;     /* Largeur totale */
| splookp->largeur = largeurspr;
| splookp->hauteur = hauteur;
|
| setpage( 1 );                          /* Construit les sprites en page 1 */
| showpage( 0 );                                /* mais affiche la page 0 */
|
| tpix = GetVideo( 1, 0, 0, largeurspr, hauteur, ALLOCBUF );   /* Fond */
|
|       /*-- Elabore et code huit fois le sprite ----------------------*/
|
| for( l = 0; l < 8; ++l )
| {                                /* Remplit d'abord le fond de pixels noirs */
|   for( y = 0; y < hauteur; ++y )
|     Line( 0, y, largeurspr-1, y, 0 );
|
|   cbptr = CBInit( largeurspr*hauteur );     /* Mémoire pour buf. AND */
|
|   for( i = 0; i < hauteur ; ++i )              /* Parcourt les lignes */
|   {
|     for( y = l; y; --y )    /* Crée les bits AND pour le bord gauche */
|       CBAppendBit( cbptr, 1 );
|
|     for( k = 0; k < slargeur; ++k )          /* Parcourt les colonnes */
|     {
|       if( ( c = *(*(bufp+i)+k) ) == 32 )          /* Pixel de fond ? */
|       {                                        /* Oui, code couleur 0 */
```

```
        setpix( k+1, i, 0 );
        CBAppendBit( cbptr, 1 );          /* Le pixel de fond reste */
       }
       else                          /* Non, met le code couleur indiqué */
       {
        setpix( k+1, i, c-64 );
        CBAppendBit( cbptr, 0 );          /* Enlève le pixel de fond */
       }
     }

     for( y = largeurspr-slargeur-1; y ; --y )     /* Ajoute bits AND */
       CBAppendBit( cbptr, 1 );                  /* pour le bord droit */
   }
   splookp->bmskp[ i ] = CBEnd( cbptr );

  /*-- Cherche le motif de pixels du sprite dans la mémoire d'écran --*/
   splookp->pixmp[ i ] = GetVideo( i, 0, 0,
                                   largeurspr, hauteur, ALLOCBUF );
 }                                     /* Passe au sprite suivant */

 PutVideo( tpix, 1, 0, 0 );   /* Restaure le fond du sprite en page 1 */
 FreePixBuf( tpix );                          /* et efface le buffer */

 return splookp;       /* Renvoie un pointeur sur le buffer du sprite */
}

/*****************************************************************
*  PrintSprite : Affiche un sprite dans une page donnée          *
**---------------------------------------------------------------**
*  Entrées : SPIDP = Pointeur sur la structure du sprite         *
*            PAGE = Page concernée (0 ou 1)                      *
*****************************************************************/

void PrintSprite( register SPID *spidp, BYTE page )
{
 int   x;                                  /* Abscisse X du sprite */

 x = spidp->x[page];
 mergeandcopybuf2video( spidp->splookp->pixmp[x % 8]->pixbptr,
                        spidp->hgptr[page]->pixbptr,
                        spidp->splookp->bmskp[x % 8],
                        page,
                        x & (7),
                        spidp->y[page],
                        spidp->splookp->largeur / 8,
                        spidp->splookp->hauteur );
}

/*****************************************************************
*  GetSpriteBg: Lit le fond du sprite et le mémorise à l'emplacement  *
*               prévu                                            *
**---------------------------------------------------------------**
*  Entrées : SPIDP = Pointeur sur la structure du sprite         *
*            PAGE  = Page d'où est tiré le fond (0 ou 1)         *
*****************************************************************/

void GetSpriteBg( register SPID *spidp, BYTE page )
{
 GetVideo( page, spidp->x[page] & (7),  spidp->y[page],
           spidp->splookp->largeur, spidp->splookp->hauteur,
           spidp->hgptr[page] );
}

/*****************************************************************
*  RestoreSpriteBg: Rétablit dans la page d'origine le fond d'un sprite*
*                   sauvegardé au préalable                      *
**---------------------------------------------------------------**
*  Entrées : SPIDP = Pointeur sur la structure du sprite         *
*            PAGE  = Page où doit être recopié le fond (0 ou 1)  *
*****************************************************************/

void RestoreSpriteBg( register SPID *spidp, BYTE page )
{
 PutVideo( spidp->hgptr[page], page,
           spidp->x[page] & (7), spidp->y[page] );
}

/*****************************************************************
*  MoveSprite: Déplace le sprite                                 *
**---------------------------------------------------------------**
*  Entrées : SPIDP  = Pointeur sur la structure du sprite        *
*            PAGE   = Page où doit être recopié le fond (0 ou 1) *
*            DELTAX = Déplacements dans les directions X et Y    *
*            DELTAY                                              *
*  Sortie  : Indicateur de collision , cf constantes OUT_...     *
*****************************************************************/

BYTE MoveSprite( SPID *spidp, BYTE page, int deltax, int deltay )
{
 int  nouvx, nouvy;                /* Nouvelles coordonnées du sprite */
 BYTE out;                        /* Mémorise un indicateur de collision */

        /*-- Décale l'abscisse X et détecte les collisions ---------*/

 if( ( nouvx = spidp->x[page] + deltax ) < 0 )
 {
   nouvx = 0 - deltax - spidp->x[page];
   out = OUT_LEFT;
 }
 else
   if( nouvx > MAXX - spidp->splookp->largeur )
   {
     nouvx = (2*(MAXX+1))-nouvx-2*(spidp->splookp->largeur);
     out = OUT_RIGHT;
   }
   else
     out = OUT_NO;

        /*-- Décale l'ordonnée Y et détecte les collisions ---------*/
 if((nouvy=spidp->y[page]+deltay) <0)                  /* Bord sup ? */
 {                                      /* oui deltay doit être négatif */
   nouvy = 0 - deltay - spidp->y[page];
   out |= OUT_TOP;
 }
 else
   if( nouvy + spidp->splookp->hauteur > MAXY+1  )      /* bord inf ? */
   {                                    /* Ja, deltay doit être positif */
     nouvy = (2*(MAXY+1))-nouvy-2*(spidp->splookp->hauteur);
     out |= OUT_BOTTOM;
   }

/*-- Fixe une nouvelle position que si différente de l'ancienne */

 if( nouvx != spidp->x[page]  ||  nouvy != spidp->y[page] )
 {                                                 /* Nouvelle position */
   RestoreSpriteBg( spidp, page );                  /* Restaure le fond */
   spidp->x[page] = nouvx;                        /* Mémorise les nouvelles */
   spidp->y[page] = nouvy;                                /* coordonnées */
   GetSpriteBg( spidp, page );                    /* Lit le nouveau fond */
   PrintSprite( spidp, page );   /* Dessine sprite dans page indiquée */
 }
 return out;
}

/*****************************************************************
*  SetSprite: Place le sprite à une position donnée              *
**---------------------------------------------------------------**
*  Entrées : SPIDP = Pointeur sur la structure du sprite         *
*            x0, y0 = Coordonnées du sprite en page 0            *
*            x1, y1 = Coordonnées du sprite en page 1            *
*  Info    : Cette fonction doit être déclenchée avant le premier *
*            appel à MoveSprite()                                *
*****************************************************************/

void SetSprite( SPID *spidp, int x0, int y0, int x1, int y1 )
{
 spidp->x[0] = x0;       /* Mémorise les coordonnées dans la structure */
 spidp->x[1] = x1;
 spidp->y[0] = y0;
 spidp->y[1] = y1;

 GetSpriteBg( spidp, 0 );                        /* Lit le fond du sprite */
 GetSpriteBg( spidp, 1 );                             /* en page 0 et 1 */
 PrintSprite( spidp, 0 );                       /* Dessine le sprite */
 PrintSprite( spidp, 1 );                             /* en page 0 et 1 */
}

/*****************************************************************
*  RemoveSprite: Retire le sprite de l'emplacement qu'il occupe  *
*                le rendant ainsi invisible                      *
**---------------------------------------------------------------**
*  Entrée : SPIDP = Pointeur sur la structure du sprite          *
*  Info   : A l'issue de cette fonction il faut appeler SetSprite()  *
*           avant de déplacer le sprite par MoveSprite()         *
*****************************************************************/

void RemoveSprite( SPID *spidp )
{
 RestoreSpriteBg( spidp, 0 );                    /* Rétablit le fond */
 RestoreSpriteBg( spidp, 1 );                         /* en page 0 et 1 */
}

/*****************************************************************
*  Demo: Démontre l'usage des différentes fonctions de ce module *
*****************************************************************/

void Demo( void )
{
 static char *VaisseauMontant[20] =
                ( "            AA            ",
                  "           AAAA           ",
                  "           AAAA           ",
                  "            AA            ",
                  "           GGBBGG         ",
                  "          GBBCCBBG        ",
                  "         GBBBCCBBBG       ",
```

```
                "          GBBBBBBBBBG          ",
                "          GBBBBBBBBBG          ",
                "  G       GBBBBBBBBBBBG      G ",
                " GCG      GGDBBBBBBBBBDGG     GCG",
                " GCG   GGBBBDBBBB  BBBDBBBGG   GCG",
                " GCBGGGBBBBBDBB    BBDBBBBBGGGBCG",
                " GCBBBBBBBBBDB     BDBBBBBBBBBBCG",
                " BBBBBBBBBBBDB BB BDBBBBBBBBBBBB",
                " GGCBBBBBBBBDBBBBBBBBBBDBBBBBBCG ",
                "  GGCCBBBDDDDDDDDDDDDDDBBBCCG    ",
                "    GGBBDDDDDGGGGGDDDDDBBG       ",
                "       GDDDDGGG    GGGDDDDG       ",
                "        DDDD          DDDD        " );

static char *VaisseauDescendant[20] =
            {
                "        DDDD          DDDD        ",
                "      GDDDDGGG    GGGDDDDG        ",
                "    GGBBDDDDDGGGGGDDDDDBBG        ",
                "  GBCCBBBDDDDDDDDDDDDDBBBCCG      ",
                "GGCBBBBBBBBDBBBBBBBBBDBBBBBBCG ",
                "BBBBBBBBBBBDB BB BDBBBBBBBBBBB",
                "GCBBBBBBBBBDB     BDBBBBBBBBBCG",
                "GCBGGGBBBBBDBB    BBDBBBBBGGGBCG",
                "GCG   GGBBBDBBB  BBBDBBBGG   GCG",
                "GCG    GGDBBBBBBBBBDGG     GCG",
                " G      GBBBBBBBBBBG       G ",
                "          GBBBBBBBBBG          ",
                "          GBBBBBBBBBG          ",
                "           GBBBCBBBG           ",
                "            GBBCBBG            ",
                "             GBBGG             ",
                "              AA               ",
                "             AAAA              ",
                "             AAAA              ",
                "              AA               " };

#define NBSPR 6                                 /* Nombre de sprites */
#define LARGEUR 42    /* Nombre de caractères de l'avis de copyright */
#define HAUTEUR  6                    /* Hauteur du même avis en lignes */
#define SX      (MAXX-(LARGEUR*8)) / 2              /* Coordonnés de */
#define SY      (MAXY-(HAUTEUR*8)) / 2                      /* départ */

struct {                                /* Décrit les sprites gérés */
    SPID *spidp;                       /* Pointeur sur l'identificateur */
    int  deltax[2],          /* Déplacement horizontale en page 0 et 1 */
         deltay[2];            /* Déplacement vertical en page 0 et 1 */
    } sprites[ NBSPR ];
BYTE   page,                                    /* Page courante */
       out;                     /* Mémorise un indicateur de collision */
int    x, y, i,                              /* Compteur d'itérations */
       dx, dy;                                      /* Déplacements */
char   lc;
SPLOOK *vaisseauupp, *vaisseaudnp;       /* Pointeurs sur les sprites */

srand( *(int far *) 0x0040006cl );   /* Init. génér. nbr. aléatoires */

      /*-- Construit les motifs binaires des sprites ---------------*/

vaisseauupp = CompileSprite( VaisseauMontant,    20 );
vaisseaudnp = CompileSprite( VaisseauDescendant, 20 );

      /*-- Remplit de caractères les deux pages graphqiues ---------*/

for( page = 0; page < 2; ++ page )
{
  setpage( page );
  showpage( page );
  for( lc = 0, y = 0; y < (MAXY+1)-8; y += 12 )
    for( x = 0; x < (MAXX+1)-8; x += 8 )
      GrafPrintf( x, y, lc & 15, 255, "%c", lc++ & 127 );

    /*-- Affiche l'avis de copyright-----------------------------*/

  Line( SX-1, SY-1, SX+LARGEUR*8, SY-1, 15 );
  Line( SX+LARGEUR*8, SY-1, SX+LARGEUR*8, SY+HAUTEUR*8,15 );
  Line( SX+LARGEUR*8, SY+HAUTEUR*8, SX-1, SY+HAUTEUR*8, 15 );
  Line( SX-1, SY+HAUTEUR*8, SX-1, SY-1, 15 );
  GrafPrintf( SX, SY,    15, 4,
              "                                          " );
  GrafPrintf( SX, SY+8,  15, 4,
              "  S6435C (c) 1990, 92 by Michael Tischer  " );
  GrafPrintf( SX, SY+16, 15, 4,
              "                                          " );
  GrafPrintf( SX, SY+24, 15, 4,
              "  Démonstration de sprites dans le mode   " );
  GrafPrintf( SX, SY+32, 15, 4,
              "       EGA/VGA 640x350 16 couleurs        " );
  GrafPrintf( SX, SY+40, 15, 4,
              "                                          " );
}

            /*-- Réalise les différents sprites ------------------*/

for( i = 0; i < NBSPR ; ++ i)
{
  sprites[ i ].spidp = CreateSprite( vaisseauupp );
  do                                /* Sélectionne les déplacements */
  {
    dx = 0;
    dy = random(8) - 4;
  }
  while ( dx==0  &&  dy==0 );

  sprites[ i ].deltax[0] = sprites[ i ].deltax[1] = dx * 2;
  sprites[ i ].deltay[0] = sprites[ i ].deltay[1] = dy * 2;

  x = ( MAXX / NBSPR * ( i ) + (MAXX / NBSPR - 40) / 2 ;
  y = random( (MAXY+1) - 40 );
  SetSprite( sprites[ i ].spidp, x, y, x - dx, y - dy );
}

/* Déplace les sprites et les fait rebondir aux extrémités de l'écran */

page = 1;                                  /* Commence en page 1 */
while( !kbhit() )       /* Une frappe de touche interrompt la boucle */
{
  showpage( (BYTE) (1 - page) );             /* Affiche l'autre page */

  for( i = 0; i < NBSPR; ++ i)               /* Parcourt les sprites */
  {                     /* Déplace les sprites et détecte les collisions */
    out = MoveSprite( sprites[i].spidp, page,
                      sprites[i].deltax[page],
                      sprites[i].deltay[page] );
    if( out & OUT_TOP  ||  out & OUT_BOTTOM )          /* Contact ? */
    {                   /* Oui change le sens du déplacement et l'image */
      sprites[i].deltay[page] = 0 - sprites[i].deltay[page];
      sprites[i].spidp->splookp = ( out & OUT_TOP ) ? vaisseaudnp
                                                    : vaisseauupp;
    }
    if( out & OUT_LEFT  ||  out & OUT_RIGHT )
      sprites[i].deltax[page] = 0 - sprites[i].deltax[page];
  }
  page = (page+1) & 1;                 /* Passe de 1 à 0 et vice versa */
}

/*-------------------------------------------------------------------*/
/*--                      PROGRAMME PRINCIPAL                      --*/
/*-------------------------------------------------------------------*/

void main( void )
{
  union REGS  regs;

  if( IsEgaVga() != NINI )       /*Dispose-t-on d'une carte EGA ou VGA */
  {                                          /* Oui, c'est parti */
    init640350();                       /* Initialise le mode graphique */
    Demo();
    getch();                             /*Attend une frappe de touche */
    regs.x.ax = 0x0003;                      /* Revient au mode texte */
    int86( 0x10, &regs, &regs );
  }
  else
    printf( "S6439C.C - (c) 1990, 92 by MICHAEL TISCHER\n\nATTENTION "\
            "Ce programme nécessite une carte EGA ou VGA.\n\n" );
}
```

Listing : S6435CA.ASM

```
;*****************************************************************;
;*                      S 6 4 3 5 C A . A S M                    *;
;*---------------------------------------------------------------*;
;*   Fonction        : contient des routines pour travailler avec les*;
;*                     sprites dans le mode 640*350 des cartes EGA   *;
;*                     et VGA                                        *;
;*---------------------------------------------------------------*;
;*   Auteur          : MICHAEL TISCHER                             *;
;*   Développé le    :  8.12.1990                                  *;
;*   Dernière MAJ    : 14.02.1992                                  *;
;*---------------------------------------------------------------*;
;*   Modèle mémoire : SMALL                                        *;
;*---------------------------------------------------------------*;
;*   Assemblage      : MASM /mx S6435CA;    ou   TASM -mx S6435CA  *;
;*                     ... puis lier à S6435C.C                    *;
;*****************************************************************;

IGROUP group _text                  ;Regroupe les segments de programme
DGROUP group _bss, _data            ;regroupe les segments de données
       assume CS:IGROUP, DS:DGROUP, ES:DGROUP, SS:DGROUP

_BSS   segment word public 'BSS'        ;Segment des variables statiques
_BSS   ends                                        ;non initialisées

_DATA  segment word public 'DATA'       ;Segment des variables globales
                                              ;et statiques initialisées

_DATA  ends

;== Constantes ================================================

SC_INDEX       = 3c4h      ;Registre d'index du contrôleur du séquenceur
SC_MAP_MASK    = 2                        ;Numéro du registre Map Mask
SC_MEM_MODE    = 4                  ;Numéro du registre de mode mémoire

GC_INDEX       = 3ceh          ;Registre d'index du contrôleur graphique
GC_READ_MAP    = 4                        ;Numéro du registre Read Map
GC_BIT_MASK    = 8              ;Numéro du registre de masquage binaire

PIXX           = 640                        ;Résolution horizontale

;== Programme =================================================

_TEXT  segment byte public 'CODE'                ;Segment de programme

;-- Déclarations publiques

public    _copybuf2video
public    _mergeandcopybuf2video
public    _copyvideo2buf

;---------------------------------------------------------------
;-- CopyBuf2Video: Recopie en mémoire d'écran le contenu d'une zone
;--                rectangulaire préalablement sauvée par CopyVideo2Buf
;-- Déclaration: CopyBuf2Plane( byte *bufptr,
;--                             byte verspage,
;--                             int  versx,
;--                             int  versy,
;--                             byte largeur,
;--                             byte hauteur );

_copybuf2video proc near

sfr0     struc                          ;Structure d'accès à la pile
bp0      dw ?                                    ;Mémorise BP
stofs0   dw ?          ;Var loc : Offset de début dans mémoire d'écran
ret_adr0 dw ?                    ;Adresse de retour à l'appelant
bufptr0  dw ?                                ;Pointeur sur buffer
verspage dw ?                                ;Page de destination
versx    dw ?                             ;Abscisse de destination
versy    dw ?                             ;Ordonnée de destination
largeur0 dw ?                                        ;Largeur
hauteur0 dw ?                                ;Hauteur de la zone
sfr0     ends                                ;Fin de la structure

fr     equ [ bp - bp0 ]              ;adr. les éléments de la structure
bfr    equ byte ptr [ bp - bp0 ]        ;adr. élément pile comme octet

       sub   sp,2               ;De la place pour les variables locales

       push  bp                      ;Prépare l'adressage des paramètres
       mov   bp,sp                                ; par le registre BP

       push  si                                   ;Sauvegarde SI et DI
       push  di

       ;-- Calcule le segment d'accès à la mémoire d'écran ------------

       mov   ax,0A000h           ;ES au début de la page de destination
       cmp   bfr.verspage,0                          ;Est-ce la page 0 ?
       je    cv0                                        ;Oui, AL est bon

       mov   ax,0A6D6h                              ;Non Page 1 en A6D6h

cv0:   mov   es,ax

       ;-- calcule l'offset de la position de destination dans la page -

       mov   ax,PIXX / 8                ;AX sur position de destination
       mul   fr.versy
       mov   bx,fr.versx
       shr   bx,1
       shr   bx,1
       shr   bx,1
       add   bx,ax
       mov   fr.stofs0,bx            ;Mémorise le résultat dans var loc

       mov   si,fr.bufptr0                  ;DS:SI pointe sur le buffer

       ;-- Initialise les compteurs pour la boucle de copie ------------

       mov   dl,bfr.largeur0                                  ;DL = octets
       mov   bx,PIXX / 8                      ;BX = offset sur ligne suivante
       sub   bl,dl
       xor   ch,ch                          ;Octet haut du compteur tjs 0

       ;-- Prépare l'adressage du plan de bits -------------------------

       mov   ah,1                ;Le numéro du plan est pris comme masque
       mov   al,SC_MAP_MASK          ;Charge le numéro du registre en AL

cv1:   mov   dx,SC_INDEX                    ;Clôture la demande d'accès
       out   dx,ax

       ;-- Routine de copie pour un plan de bits sans respect----------
                                                              ;--  du fond

       mov   di,fr.stofs0                       ;DI sur offset de départ
       mov   dh,bfr.hauteur0                                  ;DH = lignes
       mov   dl,bfr.largeur0                                  ;DL = octets

cv2:   mov   cl,dl                             ;Nombre d'octets en CL

       rep   movsb                                      ;Copie une ligne
       add   di,bx                                ;DI sur ligne suivante
       dec   dh                                ;Reste-t-il une ligne ?
       jne   cv2                                      ;Oui---> on continue

       shl   ah,1                                   ;Passe au plan suivant
       test  ah,16                           ;A-t-on traité tous les plans ?
       je    cv1                                    ;Non, on traite le suivant

       mov   ax,(0Fh shl 8)+ SC_MAP_MASK          ;Autorise l'accès à
       mov   dx,SC_INDEX                                 ;tous les plans
       out   dx,ax

       pop   di                                    ;Récupère DI et SI
       pop   si
       pop   bp                                             ;Récupère BP

       add   sp,2                               ;Retire les variables locales
       ret                                             ;Retourne à l'appelant

_copybuf2video endp

;---------------------------------------------------------------
;-- MergeAndCopyBuf2Video: Combine contenu buffer de fond avec celui
;--                        d'un buffer de sprite par l'intermédiaire d'un
;--                        masque binaire et copie le résultat dans la
;--                        mémoire d'écran
;-- Déclaration : MergeAndCopyBuf2Video( void * spribufptr,
;--                                      void * fondbufptr,
;--                                      void * andbufptr,
;--                                      BYTE page,
;--                                      int  versx,
;--                                      int  versy,
;--                                      BYTE largeur,
;--                                      BYTE hauteur );
;-- Info        : cf CopyVideo2Buf

_mergeandcopybuf2video proc near

sfr2     struc                          ;Structure d'accès à la pile
bp2      dw ?                                    ; mémorise BP
andptr2  dw ?                       ;Var loc: Pointeur dans buffer AND
stofs2   dw ?                  ;Var loc: Offset de début en mémire d'écran
```

```
!ret_adr2  dw ?                          ;Adresse de retour à l'appelant
!spribufptr dw ?                         ;Pointeur sur buffer de sprite
!fondbufptr dw ?                         ;Pointeur sur buffer de fond
!andbufptr dw ?                          ;Pointeur sur buffer AND
!verspage2 dw ?                          ;Page de destination
!versx2    dw ?                          ;Abscisse de destination
!versy2    dw ?                          ;Ordonnée de destination
!largeur2  dw ?                          ;Largeur
!hauteur2  dw ?                          ;Hauteur de la zone
!sfr2      ends                          ;Fin de la structure
!
!fr    equ [ bp - bp2 ]                  ;adr. les éléments de la structure
!bfr   equ byte ptr [ bp - bp2 ]         ;adr. éléments pile comme octets
!
!      sub  sp,4                 ;De la place pour les variables locales
!
!      push bp                   ;Prépare l'adressage des paramètres
!      mov  bp,sp                          ; par le registre BP
!
!      push ds
!      push si
!      push di
!
!      ;-- Calcule le segment d'accès à la mémoire d'écran ------------
!
!      mov  ax,0A000h          ;ES au début de la page de destination
!      cmp  bfr.verspage2,0                ;Est-ce la page 0?
!      je   cm0                            ;Oui, AL est o.k
!
!      mov  ax,0A6D6h                      ;Non , page 1 en A6D6h
!
!cm0:  mov  es,ax
!
!      ;-- calcule l'offset de la position de destination dans la page -
!
!      mov  ax,PIXX / 8          ;AX sur position de destination
!      mul  fr.versy2
!      mov  bx,fr.versx2
!      shr  bx,1
!      shr  bx,1
!      shr  bx,1
!      add  bx,ax
!      mov  fr.stofs2,bx         ;Mémorise le résultat en var loc
!
!      ;-- Initialise les compteurs pour la boucle d'itération ---------
!
!      mov  dl,bfr.largeur2                ;DL = octets
!      mov  bx,PIXX / 8          ;BX = offset sur ligne suivante
!      sub  bl,dl
!      xor  ch,ch                ;Octet fort du compteur toujours 0
!
!      ;-- Prépare l'adressage du plan de bits --------------------------
!
!      mov  ah,1                 ;Numéro du plan comme masque binaire
!
!cm1:  mov  al,SC_MAP_MASK       ;Charge en AL le numéro du registre
!      mov  dx,SC_INDEX                  ;Clôture la demande d'accès
!      out  dx,ax
!
!      ;-- Routine de copie pour un plan de bits sans tenir compte -----
!      ;-- du fond ------------------------------------------------------
!
!      mov  dx, word ptr fr.andbufptr            ;Copie en var loc
!      mov  word ptr fr.andptr2,dx       ;l'offset du pointeur AND
!
!      mov  di,fr.stofs2                 ;DI sur offset de départ
!      mov  dh,bfr.hauteur2                      ;DH = Lignes
!      mov  dl,bfr.largeur2                      ;DL = octets
!
!cm2:  mov  cl,dl                        ;Nombre d'octets en CL
!
!cm3:  mov  si,fr.fondbufptr             ;Charge pointeur de  fond
!      lodsb                         ;Lit un octet du buffer du fond
!      mov  word ptr fr.fondbufptr,si  ;Mémorise l'offset incrémenté
!
!      mov  si,fr.andptr2          ;Charge pointeur sur buffer AND
!      and  al,[si]              ;Combine le fond avec le masque AND
!      inc  si                 ;Incrémente l'offset dans le buffer AND
!      mov  word ptr fr.andptr2,si             ;puis le mémorise
!
!      mov  si,fr.spribufptr       ;Charge ptr sur buffer du sprite
!      or   al,[si]            ;Combine un octet de ce buffer par OR
!      inc  si           ;Incrémente l'offset dans le buffer du sprite
!      mov  word ptr fr.spribufptr,si          ;puis le mémorise
!
!      stosb                         ;Dépose l'octet en mémoire d'écran
!      loop cm3                            ;Traite l'octet suivant
!
!      add  di,bx                          ;DI sur ligne suivante
!      dec  dh                           ;Reste-t-il une ligne ?
!      jne  cm2                            ;Oui ---> on continue
!
!      shl  ah,1                           ;Passe au plan suivant
!      test ah,16                ;Tous les plans ont-ils été traités ?
!      je   cm1                                  ;Non, on continue
!
!      mov  ax,(0FFh shl 8)+ SC_MAP_MASK         ;Autorise l'accès
!      mov  dx,SC_INDEX                          ;à tous les plans
!      out  dx,ax
!
!      pop  di                                   ;Récupère DI et SI
!      pop  si
!      pop  ds                                         ;Récupère DS
!      pop  bp
!
!      add  sp,4                       ;Retire les variables locales
!      ret                                   ;Retourne à l'appelant
!
!_mergeandcopytxuf2video endp
!
!;-----------------------------------------------------------------------
!;-- CopyVideo2Buf: Copie dans un buffer une zone rectangulaire de la
!;--                mémoire d'écran
!;-- Déclaration : CopyPlane2Buf( byte *bufptr,
!;--                              byte depage,
!;--                              int  dex,
!;--                              int  dey,
!;--                              byte largeur,
!;--                              byte hauteur );
!;-- Info        : Dans cette version de la routine la zone à copier
!;--               doit commencer à une colonne de pixels divisible par
!;--               huit, et s'étendre sur un nombre de pixels multiple
!;--               de huit.
!;--               LARGEUR est ici le nombre d'octets par ligne dans un
!;--               plan de bits !
!
!_copyvideo2buf proc near
!
!sfr1     struc                          ;Structure d'accès à la pile
!bp1      dw ?                                     ;Mémorise BP
!stofs1   dw ?              ;Var loc: Offset de début en mémoire d'écran
!ret_adr1 dw ?                        ;Adresse de retour à l'appelant
!bufptr1  dw ?                                 ;Pointeur sur buffer
!depage   dw ?                                     ;Page d'origine
!dex      dw ?                                 ;Abscisse d'origine
!dey      dw ?                                 ;Ordonnée d'origine
!largeur1 dw ?                        ;Largeur de la zone en pixels
!hauteur1 dw ?                        ;Hauteur de la zone en pixels
!sfr1     ends                                 ;Fin de la structure
!
!fr    equ [ bp - bp1 ]                ;adr. les éléments de la structure
!bfr   equ byte ptr [ bp - bp1 ]         ;adr. élément pile comme octet
!
!      sub  sp,2                  ;de la place pour les variables locales
!
!      push bp                       ;Prépare l'adressage des paramètres
!      mov  bp,sp                                      ; par BP
!
!      push ds                                          ;mémorise DS
!      push si                                  ;ainsi que SI et DI
!      push di
!
!      push ds                         ;Charge ES avec le contenu de DS
!      pop  es                                  ;pour accéder au buffer
!
!      ;-- Calcule le segment d'accès à la mémoire d'écran ------------
!
!      mov  ax,0A000h                  ;ES au début de la page d'origine
!      cmp  bfr.depage,0                       ;Est-ce la page 0 ?
!      je   cc0                                      ;Oui, AL est bon
!
!      mov  ax,0A6D6h                          ;Non , page 1 en  A6D6h
!
!cc0:  mov  ds,ax
!
!      ;-- Forme l'offset dans la page à lire -------------------------
!
!      mov  ax,PIXX / 8                ;AX sur position d'origine
!      mul  fr.dey
!      mov  bx,fr.dex
!      shr  bx,1
!      shr  bx,1
!      shr  bx,1
!      add  bx,ax
!      mov  fr.stofs1,bx           ;mémorise le résultat dans var loc
!
!      mov  di,fr.bufptr1              ;ES:DI pointe sur le buffer
!
!      ;-- Initialise les compteurs pour la boucle de copie ------------
!
!      mov  dl,bfr.largeur1                      ;DL = octets
!      mov  bx,PIXX / 8          ;BX =  Offset sur ligne suivante
!      sub  bl,dl
!      xor  ch,ch                        ;Octet fort du compteur tjs 0
!
!      ;-- Prépare l'adressage du plan de bits --------------------------
```

```
;
        xor   ah,ah                             ;Commence par le plan 0
        mov   al,GC_READ_MAP            ;Charge le numéro du registre en AL
;
cc1:    mov   dx,GC_INDEX               ;Adresse index du contrôleur graphique
        out   dx,ax                             ;Charge le registre Read Map
;
        ;-- Routine de copie d'un plan de bits, ne tient pas compte------
        ;-- du fond -----------------------------------------------------
;
        mov   dh,bfr.hauteur1                           ;DH = lignes
        mov   dl,bfr.largeur1                           ;DL = octets
        mov   si,fr.stofs1                      ;Offset de départ en SI
;
cc2:    mov   cl,dl                             ;Nombre d'octets en CL
        rep   movsb                                     ;Copie une ligne
        add   si,bx                             ;SI sur ligne suivante
        dec   dh                                ;Reste-t-il une ligne ?
        jne   cc2                                       ;Oui on poursuit
;
        inc   ah                                ;Passe au plan suivant
;
        cmp   ah,4                              ;A-t-on traité ts les plans ?
        jne   cc1                               ;Non, c'est le tour du suivant
        pop   di                                        ;Récupère DI et SI
        pop   si
        pop   ds                                                ;Récupère DS
;
        pop   bp
;
        add   sp,2                              ;Retire les variables locales
        ret                                             ;et retourne à l'appelant
;
_copyVideo2buf endp
;
;-- Fin ---------------------------------------------------------------
;
_text   ends                                    ;Fin du segment de programme
        end                                     ;Fin de la source en assembleur
```

Listing : V16COLCA.ASM (déjà décrit en 4.8.6)

Listing : S6435P.PAS

```
{*********************************************************************
*                          S 6 4 3 5 P . P A S                       *
**-----------------------------------------------------------------**
*  Fonction : Montre comment travailler avec des sprites dans le mode *
*             graphique 640*350 EGA ou VGA en 16 couleurs sur deux    *
*             pages d'écran.                                          *
*             Ce programme utilise les routines en assembleur des     *
*             modules  V16COLPA.ASM et S6435PA.ASM.                   *
**-----------------------------------------------------------------**
*  Auteur       : MICHAEL TISCHER                                     *
*  Développé le :  5.12.1990                                          *
*  Dernière MAJ : 14.01.1991                                          *
*********************************************************************}

program S6435P;

uses dos, crt;

{-- Références externes aux routines en assembleur ------------------}

{$L v16colpa}                           { Intègre le module en assembleur }

procedure init640350; external;
procedure setpix( x, y : integer; couleur : byte ); external;
function  getpix( x, y: integer ) : byte ; external;
procedure setpage( page : integer ); external;
procedure showpage( page : integer ); external;

{$L s6435pa}                            { Intègre le module en assembleur }

procedure CopyVideo2Buf( bufptr : pointer;
                         page   : byte;
                         dex,
                         dey    : integer;
                         largeur,
                         hauteur  : byte    ); external;

procedure CopyBuf2Video( bufptr : pointer;
                         page   : byte;
                         ax,
                         ay  : integer;
                         largeur,
                         hauteur  : byte ); external;

procedure MergeAndCopyBuf2Video( spribufptr,
                                 hgbufptr,
                                 andbufptr  : pointer;
                                 page       : byte;
                                 ax,
                                 ay      : integer;
                                 largeur,
                                 hauteur    : byte ); external;

{-- Constantes ------------------------------------------------------}

const MAXX = 639;                               { Coordonnées maximales }
      MAXY = 349;

      OUT_LEFT   = 1;     { Indicateurs de collision pour SpriteMove() }
      OUT_TOP    = 2;
      OUT_RIGHT  = 4;
      OUT_BOTTOM = 8;
      OUT_NO     = 0;                               { Pas de collision }

{-- Déclarations de types -------------------------------------------}

type PIXBUF = record            { Informations pour GetVideo et PutVideo }
        largeurbyte,                    { Largeur de la zone en octets }
        hauteur  : byte;                        { Nombre de lignes }
        pixblen: integer;               { Taille du buffer de pixels }
        pixbptr: pointer;           { Pointeur sur le buffer de pixels }
           end;
     PIXPTR = ^PIXBUF;              { Pointeur sur un buffer de pixels }

     SPLOOK = record                                { Image d'un sprite }
        largeur,                                    { Largeur totale }
        hauteur  : byte;                            { Hauteur en pixels }
        bmskp  : array [0..7] of pointer;       { Ptr sur buffer AND }
        pixmp  : array [0..7] of PIXPTR;        {Ptr sur défin pixels }
        end;
     SPLP = ^SPLOOK;                            { Pointeur sur l'image }

     SPID = record                          { Descripteur de sprite (ID) }
          splookp : SPLP;                       { Pointeur sur l'image }
          x, y    : array [0..1] of integer;    { Coord. pages 0 et 1 }
          fondptr : array [0..1] of PIXPTR;     { Pointeur sur buffer
        end;                                                  du fond }
     SPIP = ^SPID;                  { Pointeur sur descripteur de sprite }

     BYTEAR = array[0..10000] of byte;  { Pour adresser les différents }
     BARPTR = ^BYTEAR;                                    { buffers }

     CARTE = ( EGA, VGA, NINI );

{*********************************************************************
*  IsEgaVga : Teste la présence d'une carte EGA ou VGA               *
**-----------------------------------------------------------------**
*  Entrée  : néant                                                   *
*  Sortie  : EGA, VGA ou NINI                                        *
*********************************************************************}
```

```
{
function IsEgaVga : CARTE;

var Regs : Registers;          { Registres pour gérer les interruptions }

begin
  Regs.AX := $1a00;              { La fonction 1AH n'existe qu'en VGA }
  Intr( $10, Regs );
  if ( Regs.AL = $1a ) then      { La fonction est-elle disponible ? }
    IsEgaVga := VGA
  else
    begin
      Regs.ah := $12;                          { Appelle l'option $10 de }
      Regs.bl := $10;                                { la fonction $12 }
      intr($10, Regs);          { Déclenche l'interruption vidéo du BIOS }
      if ( Regs.bl <> $10 ) then IsEgaVga := EGA
                            else IsEgaVga := MINI;
    end;
end;

{*****************************************************************
 *  PrintChar : Affiche un caractère en mode graphique           *
 *---------------------------------------------------------------*
 *  Entrée :  caractere = le caractère à afficher                *
 *            x, y      = Coordonnées du coin sup gauche         *
 *            cc        = Couleur du caractère                   *
 *            cf        = Couleur du fond                        *
 *  Info   : Le caractère est dessiné dans une matrice de 8*8 pixels *
 *           sur la base du jeu de caractères 8*8 en ROM         *
 *****************************************************************}

procedure PrintChar( caractere : char; x, y : integer; cc, cf : byte );

type CARADEF = array[0..255,0..7] of byte;{Structure jeu de caractères }
     CARAPTR = ^CARADEF;              { Pointe sur le jeu de caractères }

var  Regs  : Registers;        { Registres pour gérer les interruptions }
     ch    : char;                              { Pixel du caractère }
     f, k,                                  { Compteur d'itérations }
     Masque : byte;          { Masque binaire pour dessiner le caractère }

const fptr : CARAPTR = NIL;   { Pointe sur le jeu de caractères en ROM }

begin
  if fptr = NIL then             { A-t-on déjà déterminé ce pointeur ? }
    begin                                                   { Non }
      Regs.AH := $11;                    { Appelle l'option $1130 de la }
      Regs.AL := $30;                       { fonction vidéo du BIOS }
      Regs.BH := 3;           { pour obtenir un pointeur sur le jeu 8*8 }
      intr( $10, Regs );
      fptr := ptr( Regs.ES, Regs.BP );          { Compose le pointeur}
    end;

  if ( cf = 255 ) then                        { Caractère transparent ? }
    for f := 0 to 7 do { Oui ne dessine que les pixels du premier plan }
      begin
        Masque := fptr^[ord(caractere).f];{Motif binaire pour une ligne}
        for k := 0 to 7 do
          begin
            if ( Masque and 128 <> 0 ) then       { Pixel à dessiner? }
              setpix( x+k, y+f, cc );                          { Oui }
            Masque := Masque shl 1;
          end;
      end
  else                                      { Non, tient compte du fond }
    for f := 0 to 7 do                        { Parcourt les lignes }
      begin
        Masque := fptr^[ord(caractere).f];{Motif binaire pour une ligne}
        for k := 0 to 7 do
          begin
            if ( Masque and 128 <> 0 ) then          { Premier plan ? }
              setpix( x+k, y+f, cc )                           { Oui }
            else
              setpix( x+k, y+f, cf );                    { Non, fond }
            Masque := Masque shl 1;
          end;
      end;
end;

{*****************************************************************
 *  Line : Trace un segment dans la fenêtre graphique en appliquant *
 *         l'algorithme de Bresenham                             *
 *---------------------------------------------------------------*
 *  Entrée : X1, Y1  = Coordonnées de l'origine (0 - ...)        *
 *           X2, Y2  = Coordonnées de l'extrémité terminale      *
 *           COULEUR = couleur du segment                        *
 *****************************************************************}

procedure Line( x1, y1, x2, y2 : integer; couleur : byte );

var d, dx, dy,
    aincr, bincr,
    xincr, yincr,
    x, y                : integer;

{-- Procédure accessoire pour échanger deux variables entières --------}

procedure SwapInt( var i1, i2: integer );

var dummy : integer;

begin
  dummy := i2;
  i2    := i1;
  i1    := dummy;
end;

{-- Procédure principale ----------------------------------------------}

begin
  if ( abs(x2-x1) < abs(y2-y1) ) then  { Parcours : par l'axe X ou Y ? }
    begin                                                { par l'axe des Y }
      if ( y1 > y2 ) then                          { y1 supérieur à y2 ? }
        begin
        SwapInt( x1, x2 );                      { Oui, échange X1 et X2. }
        SwapInt( y1, y2 );                                { Y1 et Y2 }
        end;

      if ( x2 > x1 ) then xincr := 1          { Fixe le pas horizontal }
                     else xincr := -1;

      dy := y2 - y1;
      dx := abs( x2-x1 );
      d  := 2 * dx - dy;
      aincr := 2 * (dx - dy);
      bincr := 2 * dx;
      x := x1;
      y := y1;

      Setpix( x, y, couleur );              { Dessine le premier point }
      for y:=y1+1 to y2 do                    { Parcourt l'axe des Y }
        begin
          if ( d >= 0 ) then
            begin
              inc( x, xincr );
              inc( d, aincr );
            end
          else
            inc( d, bincr );
          Setpix( x, y, couleur );
        end;
    end
  else                                                 { par l'axe des X }
    begin
      if ( x1 > x2 ) then                       { x1 plus grand que x2 ? }
        begin
        SwapInt( x1, x2 );                      { Oui, échange X1 et X2 }
        SwapInt( y1, y2 );                                { Y1 et Y2 }
        end;

      if ( y2 > y1 ) then yincr := 1          { Fixe le pas vertical }
                     else yincr := -1;

      dx := x2 - x1;
      dy := abs( y2-y1 );
      d  := 2 * dy - dx;
      aincr := 2 * (dy - dx);
      bincr := 2 * dy;
      x := x1;
      y := y1;

      Setpix( x, y, couleur );              { Dessine le premier point }
      for x:=x1+1 to x2 do                    { Parcourt l'axe des X }
        begin
          if ( d >= 0 ) then
            begin
              inc( y, yincr );
              inc( d, aincr );
            end
          else
            inc( d, bincr );
          Setpix( x, y, couleur );
        end;
    end;
end;

{*****************************************************************
 *  GrafPrint: Affiche une chaîne formatée sur l'écran graphique *
 *---------------------------------------------------------------*
 *  Entrées : X, Y   = Coordonnées de départ (0-...)             *
 *            CC     = Couleur des caractères                    *
 *            CF     = Couleur du fond (255 = transparent)       *
 *            STRING = Chaîne avec indications de formatage      *
 *****************************************************************}
```

```
{
procedure GrafPrint( x, y : integer; cc, cf : byte; strt : string );

var i : integer;                                    { Compteur d'itérations }

begin
 for i:=1 to length( strt ) do
   begin
     printchar( strt[i], x, y, cc, cf );        { Affiche par printchar }
     inc( x, 8 );                  { x à la position du caractère suivant }
   end;
end;

{*****************************************************************
 * GetVideo : Charge le contenu d'une zone rectangulaire de la mémoire *
 *            d'écran dans un buffer                             *
 *---------------------------------------------------------------*
 * Entrées : PAGE    - Page d'écran (0 ou 1)                     *
 *           X1, Y1  - Coordonnées de départ                     *
 *           LARGEUR - Largeur de la zone rectangulaire en pixels *
 *           HAUTEUR - Hauteur de la zone rectangulaire en pixels *
 *           BUFPTR  - Pointeur sur le buffer de pixels qui va   *
 *                     mémoriser les informations                *
 * Sortie  : Pointeur sur le buffer créé qui contient la zone indiquée *
 * Info    : Si on donne au paramètre BUFPTR la valeur NIL un nouveau *
 *           buffer de pixels est alloué sur le tas et retourné. *
 *           Ce buffer peut être transmis lors d'un nouvel appel si *
 *           l'ancien contenu est effaçable et si la taille de la zone *
 *           n'a pas changé. La zone indiquée doit commencer à une *
 *           abscisse divisible par huit et s'étendre sur un nombre de *
 *           pixels multiple de huit.                            *
 *****************************************************************}

function GetVideo( page : byte; x1, y1 : integer;
                   largeur, hauteur : byte; bufptr : PIXPTR ) : PIXPTR;

begin
 if ( bufptr = NIL ) then                    { Pas de buffer transmis ? }
   begin                                                { Non, on alloue }
     new( bufptr );                          { Crée le buffer de pixels }
     getmem( bufptr^.pixbptr, (largeur*hauteur) div 2 );
     bufptr^.hauteur := hauteur;          { Hauteur du buffer en lignes }
     bufptr^.largeurbyte := largeur div 8;{Largeur une ligne en octets}
     bufptr^.pixblen:=(largeur*hauteur) div 2;{Taille totale du buffer}
   end;

 CopyVideo2Buf( bufptr^.pixbptr, page, x1, y1, largeur div 8, hauteur );
 GetVideo := bufptr;                  { Retourne un pointeur sur le buffer }
end;

{*****************************************************************
 * PutVideo : Réécrit dans la mémoire d'écran le contenu d'une zone *
 *            rectangulaire préalablement sauvegardée par GetVideo() *
 *---------------------------------------------------------------*
 * Entrée : BUFPTR  - Pointeur renvoyé par Getvideo et référençant un *
 *                    buffer de pixels                           *
 *          PAGE    - Page d'écran (0 ou 1)                     *
 *          X1, Y1 - Coordonnées de début                       *
 * Info    : Le buffer de pixels n'est pas effacé par cette procédure *
 *           cette tâche étant remplie par FreePixBuf()        *
 *           L'abscisse X indiquée doit être un multiple de huit *
 *****************************************************************}

procedure PutVideo( bufptr : PIXPTR; page : byte; x1, y1 : integer );

begin
 CopyBufZVideo( bufptr^.pixbptr, page, x1, y1,
                bufptr^.largeurbyte, bufptr^.hauteur );
end;

{*****************************************************************
 * FreePixBuf : Efface un buffer de pixels alloué sur le tas par *
 *              GetVideo                                         *
 *---------------------------------------------------------------*
 * Entrée  : BUFPTR = Pointeur renvoyé par Getvideo et référençant un *
 *                    buffer de pixels                           *
 *****************************************************************}

procedure FreePixBuf( bufptr : PIXPTR );

begin
 freemem( bufptr^.pixbptr, bufptr^.pixblen );
 dispose( bufptr );
end;

{*****************************************************************
 * CreateSprite : Crée un sprite à l'aide d'un motif de pixels   *
 *                préalablement compilé                          *
 *---------------------------------------------------------------*
 * Entrée : SPLOOKP = Pointeur sur la structure de données produite *
 *                    par CompileSprite()                        *
 * Sortie : Pointeur sur la structure du sprite créée            *
 *****************************************************************}

function CreateSprite( splookp : SPLP ) : SPIP;

var spidp : SPIP;          { Pointeur sur la structure du sprite créée }

begin
 new( spidp );              { Alloue de la mémoire pour le descripteur }
 spidp^.splookp := splookp;                    { Y transfère les données }

          {-- Crée deux buffers de fond en sauvegardant par GetVideo --}
                                        {-- une zone de la mémoire d'écran --}

 spidp^.fondptr[0] := GetVideo( 0, 0, 0, splookp^.largeur,
                                splookp^.hauteur, NIL );
 spidp^.fondptr[1] := GetVideo( 0, 0, 0, splookp^.largeur,
                                splookp^.hauteur, NIL );
 CreateSprite := spidp;   { Renvoie un ptr sur la structure du sprite }
end;

{*****************************************************************
 * CompileSprite : Crée le motif binaire d'un sprite à l'aide d'une *
 *                 définition connue au moment de l'exécution    *
 *---------------------------------------------------------------*
 * Entrées : BUFP    - Pointeur sur un tableau de pointeurs référençant *
 *                     des chaînes de caractères qui représentent le *
 *                     motif du sprite                           *
 *           HAUTEUR - Hauteur du sprite et nombre de chaînes de *
 *                     caractères                                *
 * Info    : Dans le motif transmis un espace correspond à un pixel du *
 *           fond A ou code de couleur 0,  B à 1, C à 2 etc.     *
 *****************************************************************}

function CompileSprite( var buf: hauteur : byte ) : SPLP;

type BYPTR  = ^byte;                          { Pointeur sur un octet }

var  slargeur,              { Longueur des chaînes = largeur du motif }
     largeurt,                                  { Largeur des sprites }
     c,                                        { Mémorise un caractère }
     i, k, l, y,                              { Variables d'itérations }
     andc,                                      { Compteur de pixels }
     andm     : byte;                           { Masque de pixels }
     andindex : integer;                  { Index dans le buffer AND }
     splookp  : SPLP;           { Pointeur sur la structure générée }
     lspb     : BYPTR;     { Pointeur courant dans le buffer du sprite }
     andp,                                  { Pointeur sur le buffer AND }
     bptr     : BARPTR;          {Pour adresser le buffer de l'image }
     tpix     : PIXPTR;  { Pointeur sur un buffer de pixels temporaire }

{-- Sous procédure AndBufInit: Initialise un buffer AND ----------}

procedure AndBufInit( bufp : BARPTR );

begin
 andp := bufp;                    { mémorise un pointeur sur le buffer }
 andindex := 0;                          { Commence au début du buffer }
 andm := 0;                        { Au départ le masque binaire est 0 }
 andc := 0;                    { Pas encore de bit dans le premier octet }
end;

{-- Sous-procédure AndBufAppendBit: accroche un bit au buffer AND -----}

procedure AndBufAppendBit( bit : byte );

begin
 andm := andm or bit;                  { Introduit le bit en position 0 }
 if andc = 7 then                                   { Octet rempli ? }
   begin                                                      { Oui }
     andp^[andindex] := andm;          { Mémorise l'octet dans le buffer }
     inc( andindex );                          { Adresse l'octet suivant }
     andm := 0;                          { Remet le masque binaire à 0 }
     andc := 0;      { Poursuit avec le premier bit de l'octet suivant }
   end
 else                                                { Octet non rempli }
   begin
     inc( andc );                              { Traite un bit de plus }
     andm := andm shl 1;                      { Décale le masque binaire }
   end;
end;

{-- Sous-procédure AndBufEnd: Clôture le buffer AND--------------------}

procedure AndBufEnd;

begin
 if ( andc <> 0 ) then                        { Dernier octet rempli ? }
   andp^[andindex] := andm shl (7 - andc);          { Non, on termine }
end;

begin
          {-- Crée une structure d'image et la remplit -------------}

 new( splookp );
```

```
  bptr := @buf;                                ( Pointe sur le buffer du logo )
  slargeur := bptr^[0];{ Lit la longueur des chaînes = largeur du logo }
  largeurt := ( ( slargeur + 7 + 7 ) div 8 ) * 8;     { Largeur totale }
  splookp^.largeur := largeurt;              { Mémorise largeur et hauteur }
  splookp^.hauteur := hauteur;

  setpage( 1 );                          { Construit les sprites en page 1 }
  showpage( 0 );                                  { mais affiche la page 0 }
  tpix := GetVideo( 1, 0, 0, largeurt, hauteur, NIL );   { Lit le fond }

        (-- Elabore et code huit fois le sprite ----------------------)

  for l := 0 to 7 do
    begin                          { Remplit d'abord le fond de pixels noirs }
      for y := 0 to hauteur-1 do
        Line( 0, y, largeurt-1, y, 0 );

                    (-- Alloue un buffer AND et l'initialise ----------)

      getmem( splookp^.bmskp[ l ], (largeurt*hauteur) div 8 );
      AndBufInit( splookp^.bmskp[ l ] );

      for i := 0 to hauteur-1 do                  { Parcourt les lignes }
        begin
        for y := 1 to l do          { Crée les bits AND pour le bord gauche}
            AndBufAppendBit( 1 );

        for k := 0 to slargeur-1 do               { Parcourt les colonnes }
            begin
          c := bptr^[i*(slargeur+1)+k+1];                { Lit la couleur }
          if ( c = 32 ) then                          { Pixel de fond ? }
        begin                                  { Oui, met le code couleur 0 }
                setpix( k+1, i, 0 );
          AndBufAppendBit( 1 );                    { Le pixel de fond reste }
              end
          else                                 { Non, met le code couleur indiqué }
              begin
          setpix( k+1, i, c-ord('@') );
          AndBufAppendBit( 0 );                      { Enlève le bit de fond }
              end;
            end;
        for y := largeurt-slargeur-1 downto 1 do  { Ajoute les bits AND }
          AndBufAppendBit( 1 );                        { pour le bord droit }
        end;
        AndBufEnd;                                  { Referme le buffer AND }

      (-- Cherche le motif de pixels du sprite dans la mémoire d'écran --)
        splookp^.pixmp[ l ] := GetVideo( 1, 0, 0, largeurt, hauteur, nil);

      end;                                          { Passe au sprite suivant }

    PutVideo( tpix, 1, 0, 0 );     { Restaure le fond du sprite en page 1 }
    FreePixBuf( tpix );                                 { et efface le buffer }

    CompileSprite := splookp; { Renvoie un pointeur sur buffer du sprite }
end;

{*****************************************************************
*  PrintSprite : Affiche un sprite dans une page donnée          *
*----------------------------------------------------------------*
*  Entrée : SPIDP = Pointeur sur la structure du sprite          *
*           PAGE  = Page concernée (0 ou 1 )                     *
*****************************************************************}

procedure PrintSprite( spidp : SPIP; page : byte );

var x : integer;                                { Abscisse X du sprite }

begin
  x := spidp^.x[page];
  MergeAndCopyBufZVideo( spidp^.splookp^.pixmp[x mod 8]^.pixbptr,
                         spidp^.fondptr[page]^.pixbptr,
                         spidp^.splookp^.bmskp[x mod 8],
                         page,
                         x and not(7),
                         spidp^.y[page],
                         spidp^.splookp^.largeur div 8,
                         spidp^.splookp^.hauteur );
end;

{*****************************************************************
*  GetSpriteBg : Lit le fond du sprite et le mémorise à l'emplacement *
*                prévu                                           *
*----------------------------------------------------------------*
*  Entrée : SPIDP = Pointeur sur la structure du sprite          *
*           PAGE  = Page d'où est tiré le fond (0 ou 1)          *
*****************************************************************}

procedure GetSpriteBg( spidp : SPIP; page : BYTE );

var dummy : PIXPTR;

begin
  dummy := GetVideo( page, spidp^.x[page] and not(7), spidp^.y[page],
                     spidp^.splookp^.largeur, spidp^.splookp^.hauteur,
                     spidp^.fondptr[page] );
end;

{*****************************************************************
*  RestoreSpriteBg : Rétablit dans la page d'origine le fond d'un *
*                    sprite sauvegardé au préalable              *
*----------------------------------------------------------------*
*  Entrée : SPIDP = Pointeur sur la structure du sprite          *
*           PAGE  = Page où doit être recopié le fond (0 ou 1)   *
*****************************************************************}

procedure RestoreSpriteBg( spidp : SPIP; page : BYTE );

begin
  PutVideo( spidp^.fondptr[page], page,
            spidp^.x[page] and not(7), spidp^.y[page] );
end;

{*****************************************************************
*  MoveSprite : Déplace un sprite dans sa page d'écran           *
*----------------------------------------------------------------*
*  Entrée  : SPIDP  = Pointeur sur la structure du sprite        *
*            PAGE   = Page où doit être recopié le fond (0 ou 1) *
*            DELTAX = Déplacement dans les directions X et Y     *
*            DELTAY                                              *
*  Sortie  : Indicateur de collision, cf constantes OUT_...      *
*****************************************************************}

function MoveSprite( spidp : SPIP; page : byte;
                     deltax, deltay : integer    ) : byte;

var nouvx, nouvy : integer;          { Nouvelles coordonnées du sprite }
    out          : byte;  { Indique une collision avec le bord de l'écran}

begin
        (-- Décale l'abscisse X et détecte les collisions -----------)

  nouvx := spidp^.x[page] + deltax;
  if ( nouvx < 0 ) then
    begin
      nouvx := 0 - deltax - spidp^.x[page];
      out := OUT_LEFT;
    end
  else
    if ( nouvx > MAXX - spidp^.splookp^.largeur ) then
      begin
        nouvx := (2*(MAXX+1))-nouvx-2*(spidp^.splookp^.largeur);
        out := OUT_RIGHT;
      end
    else
      out := OUT_NO;

        (-- Décale l'ordonnée Y et détecte les collisions  -----------)

  nouvy := spidp^.y[page] + deltay;
  if ( nouvy < 0 ) then                                    { Bord sup? }
    begin                                  { Oui, deltay doit être négatif }
      nouvy := 0 - deltay - spidp^.y[page];
      out := out or OUT_TOP;
    end
  else
    if ( nouvy + spidp^.splookp^.hauteur > MAXY+1  ) then  { Bord inf ? }
      begin                                  { Oui , deltay doit être positif }
        nouvy := (2*(MAXY+1))-nouvy-2*(spidp^.splookp^.hauteur);
        out := out or OUT_BOTTOM;
      end;

{Ne fixe une nouvelle position que si elle est différente de l'ancienne}

  if ( nouvx <> spidp^.x[page] ) or  ( nouvy <> spidp^.y[page] ) then
    begin                                                { Nouvelle position }
      RestoreSpriteBg( spidp, page );                     { Restaure le fond }
      spidp^.x[page] := nouvx;                          { Mémorise les nouvelles }
      spidp^.y[page] := nouvy;                                  { coordonnées }
      GetSpriteBg( spidp, page );                      { Lit le nouveau fond }
      PrintSprite( spidp, page );{ Dessine le sprite dans page indiquée }
    end;

  MoveSprite := out;
end;

{*****************************************************************
*  SetSprite : Place un sprite à une position donnée             *
*----------------------------------------------------------------*
*  Entrées : SPIDP  = Pointeur sur la structure du sprite        *
*            x0, y0 = Coordonnées du sprite en page 0            *
*            x1, y1 = Coordonnées du sprite en page 1            *
*  Info    : Cette fonction doit être déclenchée avant le premier *
*            appel à MoveSprite()                                *
*****************************************************************}
```

```
procedure SetSprite( spidp : SPIP; x0, y0, x1, y1 : integer );

begin
 spidp^.x[0] := x0;       ( Mémorise les coordonnées dans la strcuture )
 spidp^.x[1] := x1;
 spidp^.y[0] := y0;
 spidp^.y[1] := y1;

 GetSpriteBg( spidp, 0 );                    ( Lit le fond du sprite en )
 GetSpriteBg( spidp, 1 );                                ( pages 0 et 1 )
 PrintSprite( spidp, 0 );                               ( Dessine le sprite )
 PrintSprite( spidp, 1 );                                ( en page 0 et 1 )
end;

{*****************************************************************
*  Demo : Démontre l'usage des différentes fonctions de ce module  *
*****************************************************************}

procedure Demo;

const VaisseauMontant :array [1..20] of string[32] =
                ( '                AA              ',
                  '               AAAA             ',
                  '               AAAA             ',
                  '                AA              ',
                  '              GGBBGG            ',
                  '             GBBCCBBG           ',
                  '            GBBCCBBBG           ',
                  '           GBBBBBBBBBG          ',
                  '           GBBBBBBBBBG          ',
                  ' G       GBBBBBBBBBBBBG       G ',
                  'GCG     GGBBBBBBBBBBBBGG     GCG',
                  'GCG   GGBBBBBBBB  BBBBBBBGG   GCG',
                  'GCBGGGGBBBBBBBB     BBBBBBBBGGGBCG',
                  'GCBBBBBBBBBBBB      BBBBBBBBBBBCG',
                  'BBBBBBBBBBBBBB BB BBBBBBBBBBBBBB',
                  'GCBBBBBBBBBBBB      BBBBBBBBBBBCG',
                  'GCBGGGBBBBBBBB      BBBBBBBBGGGBCG',
                  'GCG   GGBBBBBBBB  BBBBBBBGG   GCG',
                  'GCG     GGBBBBBBBBBBBBGG     GCG',
                  ' G       GBBBBBBBBBBBBG       G ',
                  '           GBBBBBBBBBG          ',
                  '           GBBBBBBBBBG          ',
                  '            GBBBCCBBG           ',
                  '             GBBCCBG            ',
                  '              GGBGG             ',
                  '               AA               ',
                  '               AAAA             ',
                  '               AAAA             ',
                  '                AA              ' );

const VaisseauDescendant :array [1..20] of string[32] =
                ( '                AA              ',
                  '               AAAA             ',
                  '               AAAA             ',
                  '              GGBBGG            ',
                  '             GBBCCBBG           ',
                  '            GBBCCBBBG           ',
                  '           GBBBBBBBBBG          ',
                  '           GBBBBBBBBBG          ',
                  ' G       GBBBBBBBBBBBBG       G ',
                  'GCG     GGBBBBBBBBBBBBGG     GCG',
                  'GCG   GGBBBBBBBB  BBBBBBBGG   GCG',
                  'GCBGGGGBBBBBBBB     BBBBBBBBGGGBCG',
                  'GCBBBBBBBBBBBB      BBBBBBBBBBBCG',
                  'BBBBBBBBBBBBBB BB BBBBBBBBBBBBBB',
                  'GCBBBBBBBBBBBB      BBBBBBBBBBBCG',
                  'GCBGGGBBBBBBBB      BBBBBBBBGGGBCG',
                  'GCG   GGBBBBBBBB  BBBBBBBGG   GCG',
                  'GCG     GGBBBBBBBBBBBBGG     GCG',
                  ' G       GBBBBBBBBBBBBG       G ',
                  '           GBBBBBBBBBG          ',
                  '           GBBBBBBBBBG          ',
                  '            GBBBCCBBG           ',
                  '             GBBCCBG            ',
                  '              GGBGG             ',
                  '               AA               ',
                  '               AAAA             ',
                  '               AAAA             ',
                  '                AA              ' );

  NBSPR = 6;                                        ( Nombre de sprites )
  LARGEUR = 42;          ( Nombre de caractères de l'avis de copyright )
  HAUTEUR = 6;                          ( Nombre de lignes du même avis )
  SX    = (MAXX-(LARGEUR*8)) div 2;          ( Coordonnées de départ )
  SY    = (MAXY-(HAUTEUR*8)) div 2;

type SPRITE = record                              ( Pour gérer les sprites )
       spidp : SPIP;                      ( Pointeur sur l'identificateur )
       deltax,              ( Déplacement horizontal  en page 0 und 1 )
       deltay : array [0..1] of integer;        ( Déplacement vertical )
     end;

var sprites    : array [1..nbspr] of SPRITE;
    page,                                          ( page courante )
    lc,                                        ( caractère courant )
    out        : byte;                     ( Indicateur de collision )
    x, y, i,                                ( Compteurs d'itérations )
    dx, dy     : integer;                             ( Déplacements )
    vaisseauupp,
    vaisseaudnp : SPLP;                  ( Pointeurs sur les sprites )
    ch         : char;

begin
  Randomize;          ( Initialise le générateur de nombres aléatoires )

       {-- Construit les motifs binaires des sprites ----------------}

  vaisseauupp := CompileSprite( VaisseauMontant,   20 );
  vaisseaudnp := CompileSprite( VaisseauDescendant, 20 );

      {-- Remplit de caractères les deux pages graphiques ------------}

  for page := 0 to 1 do
    begin
      setpage( page );
      showpage( page );
      lc := 0;
      y := 0;
      while ( y < (MAXY+1)-8 ) do
        begin
          x := 0;
          while ( x < (MAXX+1)-8 ) do
            begin
              PrintChar( chr(lc and 127), x, y, lc and 15, 0 );
              inc( lc );
              inc( x, 8 );
            end;
          inc( y, 12 );
        end;

    {-- Affiche l'avis de copyright --------------------------------}

      Line( SX-1, SY-1, SX+LARGEUR*8, SY-1, 15 );
      Line( SX+LARGEUR*8, SY-1, SX+LARGEUR*8, SY+HAUTEUR*8,15 );
      Line( SX+LARGEUR*8, SY+HAUTEUR*8, SX-1, SY+HAUTEUR*8, 15 );
      Line( SX-1, SY+HAUTEUR*8, SX-1, SY-1, 15 );
      GrafPrint( SX, SY,    15, 4,
                                                              ' );
      GrafPrint( SX, SY+8,  15, 4,
                          ' S64350.PAS - (c) 1990 by Michael Tischer ' );
      GrafPrint( SX, SY+16, 15, 4,
                                                              ' );
      GrafPrint( SX, SY+24, 15, 4,
                '   Démonstration de sprites dans le mode  ' );
      GrafPrint( SX, SY+32, 15, 4,
                '        640*350 des cartes EGA et VGA     ' );
      GrafPrint( SX, SY+40, 15, 4,
                                                              ' );
    end;

      {-- Réalise les différents sprites ----------------------------}

  for i := 1 to NBSPR do
    begin
      sprites[ i ].spidp := CreateSprite( vaisseauupp );
      repeat                          ( Choisit les valeurs de déplacement )
        dx := 0;
        dy := random(10) - 5;
      until ( dx <> 0 ) or ( dy <> 0 );

      sprites[ i ].deltax[0] := dx * 2;
      sprites[ i ].deltay[0] := dy * 2;
      sprites[ i ].deltax[1] := dx * 2;
      sprites[ i ].deltay[1] := dy * 2;

      x := ( ((MAXX+1) div NBSPR) * (i-1) )
          + (((MAXX+1) div NBSPR)-40) div 2;
      y := random( (MAXY+1) - 40 );
      SetSprite( sprites[ i ].spidp, x, y, x - dx, y - dy );
    end;

{- Déplace les sprites et les fait rebondir aux extrémités de l'écran -}

  page := 1;                                    ( Commence par la page 1 )
  while ( not keypressed ) do{Une frappe de touche interrompt la boucle}
    begin
      showpage( 1-page );                        ( Affiche l'autre page )
                                                        ( ch := readkey; )
                                                  ( Attend une frappe )
      for i := 1 to nbspr do                    ( Parcourt les sprites )
    begin                ( Déplace les sprites et détecte les collisions )
        out := MoveSprite( sprites[i].spidp, page,
                           sprites[i].deltax[page],
                           sprites[i].deltay[page] );
      if ( ( out and OUT_TOP ) <> 0 ) or     (Contact en haut/en bas ? )
           ( ( out and OUT_BOTTOM) <> 0) then
          begin
              {-- Oui inverse le déplacement et change l'image - ---}

            sprites[i].deltay[page] := -sprites[i].deltay[page];
            if ( ( out and OUT_TOP ) <> 0 ) then
              sprites[i].spidp^.splookp := vaisseaudnp
            else
              sprites[i].spidp^.splookp := vaisseauupp;
          end;
      if ( ( out and OUT_LEFT ) <> 0 ) or     {contact à gauche/droite ?}
           ( ( out and OUT_RIGHT ) <> 0 ) then
          sprites[i].deltax[page] := -sprites[i].deltax[page];
      end;
    page := (page+1) and 1;              ( Passe de 1 à 0 et vice versa)
    end;
  ch := readkey;                         ( Attend une frappe de touche )
end;
```

```
{-------------------------------------------------------------}
{--                    PROGRAMME PRINCIPAL                  --}
{-------------------------------------------------------------}

begin
 if ( IsEgaVga <> NINI ) then  { Dispose-t-on d'une carte EGA ou VGA ? }
   begin                                              { Oui, c'est parti }
     Init640350;                              { Initialise le mode graphique }
     Demo;
     Textmode( C080 );                                { Rétablit le mode texte }
   end
 else
   writeln( 'S6435P.PAS - (c) 1990 by MICHAEL TISCHER'#13#10#10 +
        'Attention! Ce programme nécessite une carte EGA ou VGA');
end.
```

Listing : S6435PA.ASM

```
;*****************************************************************;
;*                      S6435PA.ASM                              *;
;*---------------------------------------------------------------*;
;*    Fonction : contient des routines pour travailler avec des  *;
;*               sprites dans le mode  640*350 des cartes EGA et VGA *;
;*---------------------------------------------------------------*;
;*    Auteur       : MICHAEL TISCHER                             *;
;*    Développé le :  8.12.1990                                  *;
;*    Dernière MAJ : 14.01.1991                                  *;
;*---------------------------------------------------------------*;
;*    Assemblage : MASM /mx S6435PA;     ou   TASM -mx S6435PA   *;
;*                 puis lier à S6435P.PAS                        *;
;*****************************************************************;

;== Constantes ==================================================

SC_INDEX      = 3c4h      ;Registre d'index du contrôleur du séquenceur
SC_MAP_MASK   = 2                    ;Numéro du registre Map Mask
SC_MEM_MODE   = 4                ;Numéro du registre de mode mémoire

GC_INDEX      = 3ceh         ;Registre d'index du contrôleur graphique
GC_READ_MAP   = 4                    ;Numéro du registre Read Map
GC_BIT_MASK   = 8              ;Numéro du registre de masquage binaire

PIXX          = 640                          ;Résolution horizontale

;== Segment de données ==========================================

DATA   segment word public
DATA   ends

;== Programme ===================================================

CODE   segment byte public                          ;Segment de programme

       assume cs:code, ds:data

;-- Déclarations publiques ---------------------------------------

public     copybuf2video
public     mergeandcopybuf2video
public     copyvideo2buf

;-----------------------------------------------------------------
;-- CopyBuf2Video: Recopie en mémoire d'écran le contenu d'une zone
;--                rectangulaire préalablement sauvegardée par
;--                CopyVideo2Buf
;-- Appel depuis TP: CopyBuf2Plane( bufptr   : pointer;
;--                                 verspage : byte;
;--                                 versx,
;--                                 versy    : integer;
;--                                 largeur,
;--                                 hauteur  : byte );
;-- Info         : cf CopyVideo2Buf

copybuf2video proc near

sfr0    struc                              ;Structure d'accès à la pile
bp0     dw ?                                        ;Mémorise BP
stofs0  dw ?            ;Var loc: Offset de début dans mémoire d'écran
ret_adr0 dw ?                        ;Adresse de retour à l'appelant
hauteur0 dw ?                                ;Hauteur de la zone
largeur0 dw ?                                        ;Largeur
versy   dw ?                              ;Ordonnée de destination
versx   dw ?                              ;Abscisse de destination
verspage dw ?                                ;Page de destination
bufptr0 dd ?                                 ;Pointeur sur buffer
sfr0    ends                                         ;Fin de la structure

fr      equ [ bp - bp0 ]                ;adr. les éléments de la structure
bfr     equ byte ptr [ bp - bp0 ] ;adr. les éléments de la pile en octets

        sub   sp,2                ;De la place pour les variables locales

        push  bp                          ;Prépare l'adressage des paramètres
        mov   bp,sp                                  : par le registre BP

        push  ds

        cld                             ;Fixe le sens des opérations sur chaines

              ;-- Calcule le segment d'accès à la mémoire d'écran ------

        mov   ax,0A000h             ;ES au début de la page de destination
        cmp   bfr.verspage,0                          ;Est-ce la page 0 ?
        je    cv0                                        ;Oui, AL est bon

        mov   ax,0A6D6h                             ;Non, , page 1 en A6D6h

cv0:    mov   es,ax

          ;-- Calcule l'offset de la position de destination dans la page

        mov   ax,PIXX / 8                   ;AX sur position de destination
        mul   fr.versy
        mov   bx,fr.versx
        shr   bx,1
        shr   bx,1
        shr   bx,1
        add   bx,ax
        mov   fr.stofs0,bx          ;Mémorise le résultat dans var locale

        lds   si,fr.bufptr0                    ;DS:SI pointe sur le buffer

        ;-- Initialise les compteurs pour la boucle de copie ------------

        mov   dl,bfr.largeur0                                    ;DL = octets
        mov   bx,PIXX / 8                 ;BX = Offset sur ligne suivante
        sub   bl,dl
        xor   ch,ch                                ;Octet fort du compteur = 0

                                        ;-- Prépare l'adressage du plan de bits

        mov   ah,1                  ;Le numéro du plan est pris comme masque
        mov   al,SC_MAP_MASK            ;Charge en AL le numéro du registre

cv1:    mov   dx,SC_INDEX               ;Clôture la demande d'accès au plan
        out   dx,ax

            ;-- Routine de copie pour un plan de bits sans respect - ----
                                                              ;-- du fond

        mov   di,fr.stofs0                          ;DI sur offset de départ
        mov   dh,bfr.hauteur0                                    ;DH = lignes
        mov   dl,bfr.largeur0                                    ;DL = octets

cv2:    mov   cl,dl                                 ;Nombre d'octets en CL

        rep movsb                                          ;Copie une ligne
        add   di,bx                                      ;DI sur ligne suivante
        dec   dh                                        ;Reste-t-il une ligne ?
        jne   cv2                                          ;Oui---> on continue

        shl   ah,1                                    ;Passe au plan suivant
        test  ah,16                     ;Tous les plans ont-il été traités ?
        je    cv1                                          ;Non, on continue

        mov   ax,(0Fh shl 8)+ SC_MAP_MASK       ;Autorise l'accès à -
        mov   dx,SC_INDEX                                ;tous les plans
        out   dx,ax

        pop   ds                                        ;Récupère DS et BP
        pop   bp

        add   sp,2                              ;Retire les variables locales
        ret   14                          ;Retourne à l'appelant en enlevant
```

```
!                                               ; les paramètres de la pile
!copybuf2video endp
!
!;-----------------------------------------------------------------
!;-- MergeAndCopyBuf2Video: Combine le contenu d'un buffer de fond avec
!;--                        celui d'un buffer sprite par l'intermédiaire
!;--                        d'un masque binaire et copie le résultat dans
!;--                        la mémoire d'écran
!;-- Appel depuis TP: MergeAndCopyBuf2Video( spribufptr,
!;--                                         fondbufptr,
!;--                                         andbufptr  : pointer;
!;--                                         page       : byte;
!;--                                         versx,
!;--                                         versy      : integer;
!;--                                         largeur,
!;--                                         hauteur    : byte );
!;-- Info        : cf CopyVideo2Buf
!
!mergeandcopybuf2video proc near
!
!sfr2    struc                                  ;Structure d'accès à la pile
!bp2     dw ?                                           ; mémorise BP
!andptr2  dd ?                          ;Var loc: Pointeur dans buffer AND
!stofs2   dw ?                  ;Var loc: Offset de début en mémire d'écran
!ret_adr2 dw ?                          ;Adresse de retour à l'appelant
!hauteur2 dw ?                                  ;Hauteur de la zone
!largeur2 dw ?                                          ;Largeur
!versy2   dw ?                                  ;Ordonnée de destination
!versx2   dw ?                                  ;Abscisse de destination
!verspage2 dw ?                                 ;Page de destination
!andbufptr  dd ?                                ;Pointeur sur buffer AND
!fondbufptr dd ?                                ;Pointeur sur buffer de fond
!spribufptr dd ?                                ;Pointeur sur buffer de sprite
!sfr2    ends                                   ;Fin de la structure
!
!fr      equ [ bp - bp2 ]                ;adr. les éléments de la structure
!bfr     equ byte ptr [ bp - bp2 ] ;adr. les éléments de la pile en octets
!
!       sub   sp,6                      ;De la place pour les variables locales
!
!       push  bp                                ;Prépare l'adressage des paramètres
!       mov   bp,sp                                     ; par le registre BP
!
!       push  ds
!
!       cld                             ;Fixe le sens des opérations sur chaines
!
!                       :-- Calcule le segment d'accès à la mémoire d'écran
!
!       mov   ax,0A000h                 ;ES au début de la page de destination
!       cmp   bfr.verspage2,0                           ;Est-ce la page 0?
!       je    cm0                                       ;Oui, AL est o.k
!
!       mov   ax,0A6D6h                                 ;Non , page 1 en A6D6h
!
!cm0:   mov   es,ax
!
!         :-- calcule l'offset de la position de destination dans la page
!
!       mov   ax,PIXX / 8                       ;AX sur position de destination
!       mul   fr.versy2
!       mov   bx,fr.versx2
!       shr   bx,1
!       shr   bx,1
!       shr   bx,1
!       add   bx,ax
!       mov   fr.stofs2,bx              ;Mémorise le résultat en var loc
!
!             :-- Initialise les compteurs pour la boucle d'itération
!
!       mov   dl,bfr.largeur2                                   ;DL = octets
!       mov   bx,PIXX / 8                       ;BX = offset sur ligne suivante
!       sub   bl,dl
!       xor   ch,ch                     ;Octet fort du compteur toujours 0
!
!       mov   ax,word ptr fr.andbufptr+2                ;Copie en var loc
!       mov   word ptr fr.andptr2+2,ax       ; le segment du pointeur AND
!
!             :-- Prépare l'adressage du plan de bits ---------------
!
!       mov   ah,1                      ;Numéro du plan comme masque binaire
!
!cm1:   mov   al,SC_MAP_MASK            ;Charge en AL le numéro du registre
!       mov   dx,SC_INDEX                       ;Clôture la demande d'accès
!       out   dx,ax
!
!          :-- Routine de copie pour un plan de bits sans tenir compte -
!                                                       :-- du fond
!       mov   dx,word ptr fr.andbufptr                  ;Copie en var loc
!       mov   word ptr fr.andptr2,dx            ;l'offset du pointeur AND
!       mov   di,fr.stofs2                       ;DI sur offset de départ
!       mov   dh,bfr.hauteur2                                   ;DH = Lignes
!       mov   dl,bfr.largeur2                                   ;DL = octets
!
!cm2:   mov   cl,dl                                  ;Nombre d'octets en CL
!
!cm3:   lds   si,fr.fondbufptr                  ;Charge pointeur de  fond
!       lodsb                                   ;Lit un octet du buffer du fond
!       mov   word ptr fr.fondbufptr,si     ;Mémorise l'offset incrémenté
!
!       lds   si,fr.andptr2                     ;Charge pointeur sur buffer AND
!       and   al,[si]                   ;Combine le fond avec le masque AND
!       inc   si                        ;Incrémente l'offset dans le buffer AND
!       mov   word ptr fr.andptr2,si                    ;puis le mémorise
!
!       lds   si,fr.spribufptr;Charge le pointeur sur le buffer du sprite
!       or    al,[si]                   ;Combine un octet de ce buffer par OR
!       inc   si                ;Incrémente l'offset dans le buffer du sprite
!       mov   word ptr fr.spribufptr,si                 ;puis le mémorise
!
!       stosb                                   ;Dépose l'octet en mémoire d'écran
!       loop  cm3                                       ;Traite l'octet suivant
!
!       add   di,bx                                     ;DI sur ligne suivante
!       dec   dh                                        ;Reste-t-il une ligne ?
!       jne   cm2                                       ;Oui ---> on continue
!
!       shl   ah,1                                      ;Passe au plan suivant
!       test  ah,16                             ;Tous les plans ont-ils été traités ?
!       je    cm1                                       ;Non, on continue
!
!       mov   ax,(0FFh shl 8)+ SC_MAP_MASK              ;Autorise l'accès
!       mov   dx,SC_INDEX                               ;à tous les plans
!       out   dx,ax
!
!       pop   ds                                        ;Récupère DS et BP
!       pop   bp
!
!       add   sp,6                              ;Retire les variables locales
!       ret   22                                ;Retourne à l'appelant en enlevant
!                                                       ;les paramètres de la pile
!mergeandcopybuf2video endp
!
!;-----------------------------------------------------------------
!;-- CopyVideo2Buf: Copie dans un buffer une zone rectangulaire de
!;--                la mémoire d'écran
!;-- Appel depuis TP: CopyVideo2Buf( bufptr : pointer;
!;--                                 depage: byte;
!;--                                 dex,
!;--                                 dey    : integer;
!;--                                 largeur,
!;--                                 hauteur   : byte );
!;-- Info        : Dans cette version de la routine la zone à copier
!;--               doit commencer à une colonne de pixels divisible par
!;--               huit, et s'étendre sur un nombre de pixels multiple
!;--               de 8
!;--               LARGEUR est ici le nombre d'octets par ligne dans
!;--               un plan de bits !
!
!copyvideo2buf proc near
!
!sfr1    struc                                  ;Structure d'accès à la pile
!bp1     dw ?                                           ;Mémorise BP
!stofs1   dw ?                  ;Var loc: Offset de début en mémoire d'écran
!ret_adr1 dw ?                          ;Adresse de retour à l'appelant
!hauteur1 dw ?                                  ;Hauteur de la zone en pixels
!largeur1 dw ?                                  ;Largeur de la zone en pixels
!dey      dw ?                                  ;Ordonnée d'origine
!dex      dw ?                                  ;Abscisse d'origine
!depage   dw ?                                  ;Page d'origine
!bufptr1  dd ?                                  ;Pointeur sur buffer
!sfr1    ends                                   ;Fin de la structure
!
!fr      equ [ bp - bp1 ]                ;adr. les éléments de la structure
!bfr     equ byte ptr [ bp - bp1 ]   ;adr. un élément de pile comme octet
!
!       sub   sp,2                      ;De la place pour les variables locales
!
!       push  bp                                ;Prépare l'adressage des paramètres
!       mov   bp,sp                                     ; par le registre BP
!
!       push  ds
!       cld                                     ;Fixe le sens de parcours des chaînes
!
!             ;-- Calcule le segment d'accès à la mémoire d'écran ------
!
!       mov   ax,0A000h                 ;ES au début de la page d'origine
!       cmp   bfr.depage,0                              ;Est-ce la page 0 ?
!       je    cc0                                       ;Oui, AL est bon
!
!       mov   ax,0A6D6h                                 ;Non , page 1 en  A6D6h
!
!cc0:   mov   ds,ax
!
!          :-- Forme l'offset dans la page à lire  ----------------
```

```
;
        mov   ax,PIXX / 8                       ;AX sur position d'origine
        mul   fr.dey
        mov   bx,fr.dex
        shr   bx,1
        shr   bx,1
        shr   bx,1
        add   bx,ax
        mov   fr.stofs1,bx              ;mémorise le résultat dans var loc

        les   di,fr.bufptr1                     ;ES:DI pointe sur le buffer

              ;-- Initialise les compteurs pour la boucle de copie -----

        mov   dl,bfr.largeur1                                   ;DL = octets
        mov   bx,PIXX / 8               ;BX = Offset sur ligne suivante
        sub   bl,dl
        xor   ch,ch                             ;Octet fort du compteur tjs 0

                ;-- Prépare l'adressage du plan de bits -------------

        xor   ah,ah                                     ;Commence par le plan 0
        mov   al,GC_READ_MAP            ;Charge le numéro du registre en AL

ccl:    mov   dx,GC_INDEX ;Charge adresse d'index du contrôleur graphique
        out   dx,ax                             ;Charge le registre Read Map

          ;-- Routine de copie d'un plan de bits, ne tient pas compte--
                                                        ;-- du fond
;
        mov   dh,bfr.hauteur1                           ;DH = lignes
        mov   dl,bfr.largeur1                           ;DL = octets
        mov   si,fr.stofs1                      ;Offset de départ en SI

cc2:    mov   cl,dl                             ;Nombre d'octets en CL
        rep movsb                                       ;Copie une ligne
        add   si,bx                             ;SI sur ligne suivante
        dec   dh                                ;Reste-t-il une ligne ?
        jne   cc2                                       ;Oui on poursuit

        inc   ah                                ;Passe au plan suivant
        cmp   ah,4                              ;A-t-on traité ts les plans ?
        jne   cc1                               ;Non, c'est le tour du suivant

        pop   ds                                ;Récupère DS et BP
        pop   bp

        add   sp,2                              ;Retire les variables locales
        ret   14                                        ;Retourne à l'appelant
                                                ;et enlève les paramètres de la pile
copyvideo2buf endp
;
;-- Fin ---------------------------------------------------------------
;
CODE    ends                                    ;Fin du segment de code
        end                                             ;Fin du programme
```

Listing : V16COLPA.ASM (déjà décrit en 4.8.6)

4.8.10. Les registres des cartes EGA/VGA

Les cartes EGA/VGA reposent essentiellement sur quatre contrôleurs se répartissant les tâches liées à la génération du signal vidéo. Concrètement, il s'agit :

- ✓ du contrôleur CRT,
- ✓ du contrôleur d'attributs,
- ✓ du contrôleur graphique,
- ✓ du séquenceur et
- ✓ du convertisseur digital en analogique (DAC) existant uniquement sur une carte VGA.

Les cartes EGA et VGA disposent en outre de quelques registres généraux décrits également dans ce chapitre.

A l'origine, les divers supports de fonction pouvaient se distinguaient par rapport à la carte mère d'une telle carte car ils étaient intégrés dans les différentes structures. Par la suite, ils ont été touchés par la miniaturisation croissante et ont donné naissance à un ou deux contrôleurs étendus qui effectuent toutes les tâches nécessaires.

Ils ne se contentent pas d'implémenter d'innombrables fonctions et modes graphiques allant largement au-delà du véritable standard EGA/VGA, mais s'efforcent généralement de conserver telle quelle l'affectation initiale des registres IBM EGA/VGA pour des raisons de compatibilité. Les registres présentés ici ne divergent que très peu des

cartes EGA/VGA reconnues comme le modèle standard sur le marché. Nous insisterons davantage sur les registres dont le mode d'affectation varie en fonction des caPar la suite, ils ont été touchés par la miniaturisation croissante et ont donné naissance à un ou deux contrôleurs étendus qui effectuent toutes les tâches nécessaires.

Ils ne se contentent pas d'implémenter d'innombrables fonctions et modes graphiques allant largement au-delà du véritable standard EGA/VGA, mais s'efforcent généralement de conserver telle quelle l'affectation initiale des registres IBM EGA/VGA pour des raisons de compatibilité. Les registres présentés ici ne divergent que très peu des cartes EGA/VGA reconnues comme le modèle standard sur le marché. Nous insisterons davantage sur les registres dont le mode d'affectation varie en fonction des cartes EGA et VGA.

Les registres qui s'écartent du standard EGA/VGA et dont les caractéristiques nécessitent une explication plus approfondie ne sont pas cités dans ce chapitre. Mais en dehors de ces registres, il n'existe aucun standard pour les cartes Super EGA et Super VGA ni d'autres extensions. D'ailleurs, les cartes Super VGA des trois principaux constructeurs présentent des différences entre elles.

Malgré l'absence de ces registres, les lecteurs découvriront tout de même des points intéressants. En dehors d'un regard porté sur le mode de fonctionnement interne d'une carte EGA/VGA, les diverses descriptions de registres ouvrent la voie à de nombreuses techniques qui n'étaient même pas accessibles à travers le BIOS EGA/VGA étendu.

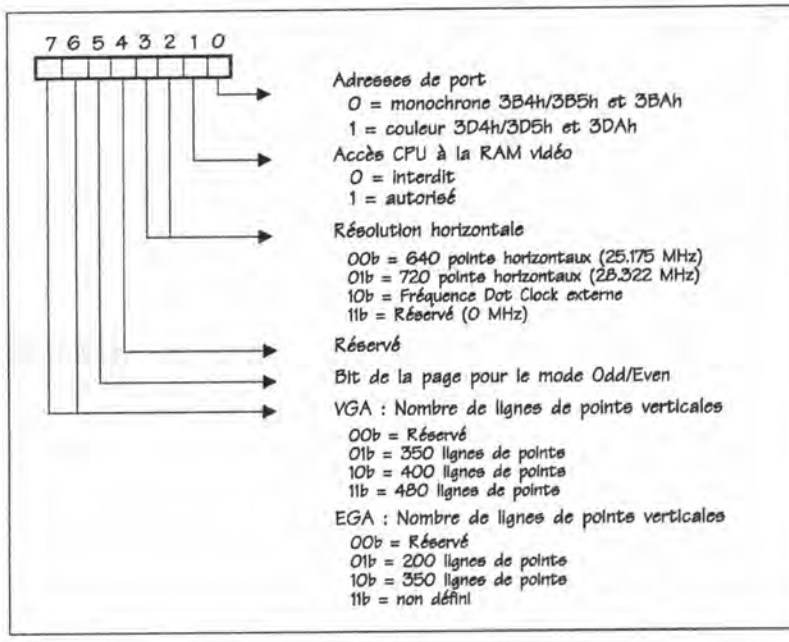
Il existe en outre toutes sortes de registres dont la manipulation est quelque peu délicate. Il s'agit en particulier des registres du contrôleur CRT pilotant la création des signaux vidéo et de synchronisation pour le retour horizontal et vertical du rayon électronique. Les combinaisons entre les divers registres sont complexes à établir et en raison de leur nature, ils risquent d'endommager le moniteur en cas de fausse programmation.

Il existe cependant des bits et registres que vous pouvez manipuler sans problème. Mais n'oubliez pas de faire attention à la différence d'affectation de certains registres et bits des cartes EGA et VGA. Des différences infimes sont en effet à prendre en considération mais nous attirerons toujours votre attention au moment voulu.

Registres généraux

Outre les registres du contrôleur qui sont très particuliers, une carte EGA ou VGA dispose d'autres registres informant sur le mode de fonctionnement de la carte et c'est pourquoi ils sont désignés par "registres généraux".

Miscellaneous Output — Registres généraux, Ecriture : Port 3CCh



0 Pour l'émulation des cartes MDA, ce bit peut servir à définir l'adresse de port des données CRT et du registre d'index ainsi que le registre Input Status 1. Alors que ces registres occupent normalement les ports 3D4h/3D5h et 3DAh, ils peuvent ainsi être commutés sur les adresses de port 3B4h/3B5h et 3BAh.

3+2 Ce champ de bit sélectionne le générateur de tâches actif. Il dessine ainsi une ligne de points pour la résolution horizontale puisqu'un lien direct se crée entre le Dot Clock Rate et le nombre de points affichables.

Le Dot Clock Rate de 0 MHz est réservé parce qu'il peut uniquement être utilisé pendant un reset de la carte VGA.

La modification de ce registre doit toujours s'accompagner d'un reset synchrone à travers le registre reset du séquenceur.

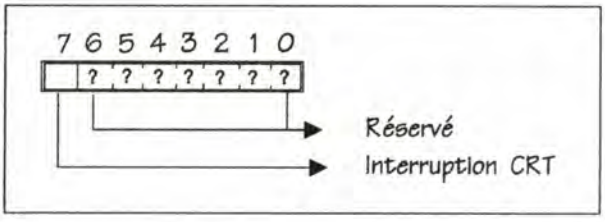
5 Dans les modes Odd/Even (mode vidéo 0, 1, 2, 3 et 7), ce bit agit comme un bit de poids faible de l'accès à la mémoire. Il décide par conséquent si seules les adresses paires ou impaires doivent être adressées dans les différents plans de bits. S'il contient la valeur 1 (défaut), tous les octets sont adressés sur les adresses d'offset paires. Inversement, la valeur 0 autorise l'accès aux adresses impaires.

Le bit perd sa valeur lorsque le mode Chain est défini à travers le bit 1 dans le registre 6 du contrôleur graphique ou le mode Chain4 à travers le bit 3 dans le registre 4 du séquenceur.

6+7 En fait, ces deux bits ne sélectionnent pas directement la résolution verticale mais la polarité des signaux Retrace horizontaux et verticaux. En pratique, on obtient ainsi la résolution verticale spécifiée.

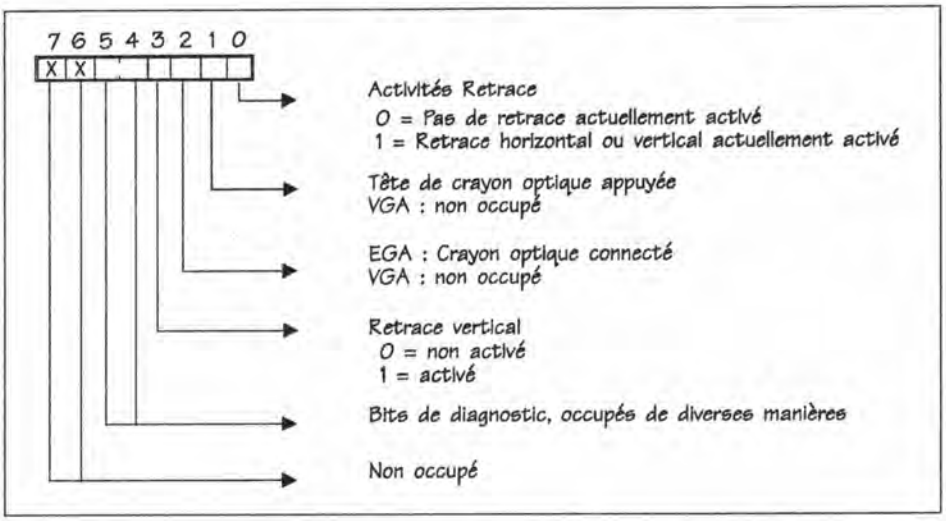
Attention ! Le mode d'émulation 200 points de la carte VGA ne peut pas être défini à travers ce champ de bits. Il s'agit en fait d'un mode 400 points où 200 lignes seulement sont affichées en double.

Input Statut 0 Registres généraux, Port 3C2h, Lecture seulement



7 Ce bit indique qu'un retour vertical du rayon électronique a eu lieu dans la mesure où cet événement doit déclencher une interruption. Après un tel retour, il reste sur 1 tant qu'il n'est pas rétabli à travers le bit 4 du registre Vertical Retrace End du contrôleur CRT.

Input Status 1 Registres généraux, Port 3BAh ou 3DAh, lecture seulement



1+2 Sur une carte EGA, ces bits représentent l'interface avec le crayon optique qui n'est plus soutenu par les cartes VGA. Outre le test de contrôle portant sur l'existence du crayon optique à travers le bit 2, le bit 1 peut servir à déterminer si la tête du crayon optique est actuellement enfoncée. La position du crayon optique peut alors être obtenue à travers les registres 10h et 11h du contrôleur CRT.

3 Pour de nombreux programmes, ce bit est surtout intéressant parce qu'il permet de lire l'état du retour vertical. Par conséquent, il donne le signal de départ permettant de modifier les registres. Sinon, l'écran se couvrirait de neige.

Contrôleur CRT

La tâche essentielle du contrôleur CRT consiste à configurer l'écran en créant des signaux pour le moniteur qui permettent au rayon électronique de piloter le tube cathodique. Ainsi, on rencontre ici de nombreux registres jouant surtout un rôle dans le timing du retour horizontal et vertical du rayon électronique. Ces registres présentent en général peu d'intérêt pour le programmeur car ils sont complexes à gérer. Pour les programmer, il suffit de s'en remettre au BIOS qui les programme en conséquence en changeant tout simplement le mode vidéo.

Mais le BIOS n'est pas capable de programmer des registres tels que les registres offset ou Line Compare utilisés pour des effets vidéo spéciaux. Voici la liste des 25 registres du contrôleur CRT.

Nº	Nom de registre
00h	Horizontal Total
01h	Horizontal Display End
02h	Start Horizontal Blanking
03h	End Horizontal Blanking
04h	Start Horizontal Retrace
05h	End Horizontal Retrace
06h	Vertical Total
07h	Overflow
08h	Vertical Pel Panning
09h	Maximum Scan Line
0Ah	Cursor Start
0Bh	Cursor End
0Ch	Start Address High
0Dh	Start Address Low
0Eh	Cursor Location High
0Fh	Cursor Location Low
10h	Start Vertical Retrace
11h	End Vertical Retrace
10h	Light-Pen Low
11h	Light-Pen High (EGA seulement)
12h	Vertical Display End
13h	Offset
14h	Underline Location
15h	Start Vertical Blank
16h	End Vertical Blank
17h	Mode Control
18h	Line Compare

L'adressage de tous les registres du contrôleur CRT s'effectue à travers un registre d'index et de données situés à l'adresse de port 3D4h ou 3D5h si la carte EGA ou VGA est utilisée en mode couleur. En mode monochrome, ces registres se trouvent en 3B4h et 3B5h.

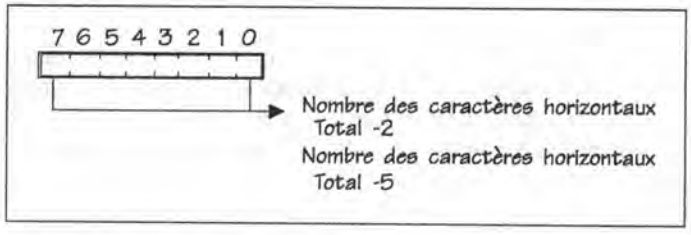
Selon le modèle classique, il convient d'écrire le numéro du registre dans le registre d'index avant d'accéder à un registre. Un accès en lecture permet alors de lire le contenu du registre adressé via le registre de données. Cela n'est exclusivement possible qu'avec les cartes VGA. Avec des cartes EGA, seuls les registres Light Pen sont lisibles puisque tous les autres sont destinés à l'écriture.

Pour un accès en écriture, vous ne pouvez écrire simultanément que dans les registres d'index et de données lors d'une opération 16 bits*. Il existe des exceptions représentées par certaines machines de marque Olivetti et AT&T. Dans ce cas, les registres d'index et de données doivent être écrits par deux accès 8 bits distincts et consécutifs.). La valeur du registre d'index doit être transmise dans l'octet de poids faible, celle du registre de données dans l'octet de poids fort de la valeur 16 bits.

Si l'adresse du registre souhaité a été écrite dans le registre d'index, elle n'est pas validée sur le registre de données par un accès en lecture ou écriture. Elle libère au contraire l'accès au registre souhaité. La conséquence est qu'il suffit de charger la première fois le numéro du registre dans le registre d'index en cas d'accès multiples et consécutifs à un registre CRT. Tous les accès ultérieurs peuvent alors s'effectuer directement à travers le registre de données.

N'oubliez pas qu'avec une carte VGA les huit premiers registres du contrôleur CRT peuvent uniquement être écrits lorsque le bit 7 du registre 11h est mis sur 0. Mais par le BIOS, il suffit de charger la valeur 1, ce qui interdit tout accès aux huit premiers registres CRT.

Horizontal Total — Contrôleur CRT, Registre 00h

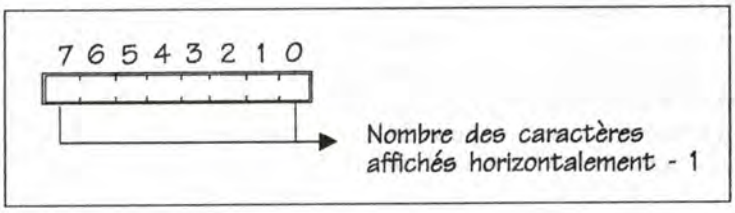


0-7 Le nombre total de caractères traités par ligne d'écran est stocké ici. En mode texe, le terme caractères désigne effectivement un caractère ASCII. En mode graphique, il correspond à un groupe de 8 points graphiques. Le nombre total de caractères est calculé à partir du quotient résultant entre la largeur de bande

et la fréquence de balayage horizontale divisé par le nombre de points par caractère.

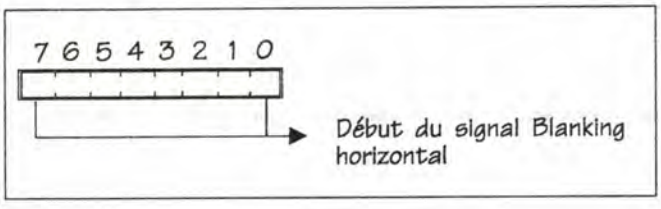
Avec une carte EGA, la valeur stockée dans ce registre doit être réduite de 2 par rapport au nombre total réel. Avec une carte VGA, elle doit être réduite de 5.

Horizontal Display — Contrôleur CRT, Registre 01h



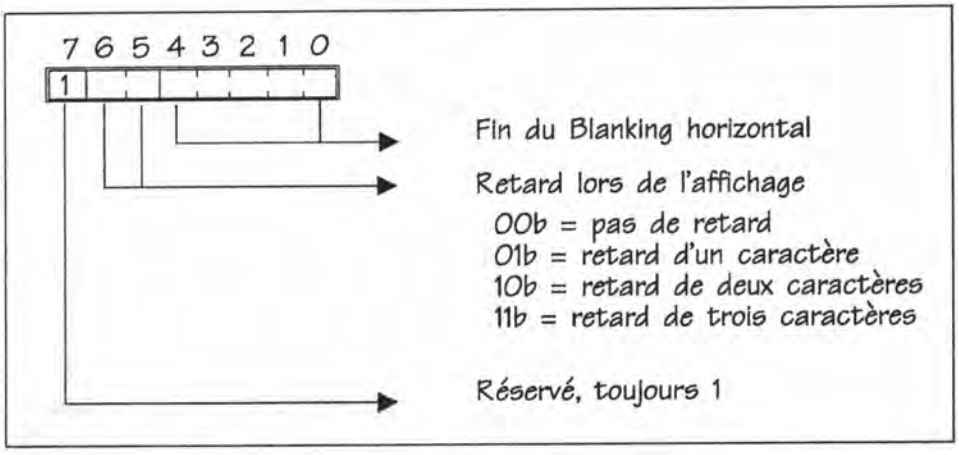
0-7 Ce registre contient le nombre effectif de caractères affichés. Que ce soit avec une carte EGA ou VGA, la valeur stockée ici doit être réduite de 1 par rapport à la valeur réelle.

Start Horizontal Blanking — Contrôleur CRT, Registre 02h



0-7 Le début du signal Blanking horizontal met fin à la sortie des caractères car le rayon électronique du tube cathodique est désactivé. L'unité choisie est à nouveau le caractère. Le premier caractère affiché dans la marge gauche de l'écran est doté du numéro 0.

End Horizontal Blanking Contrôleur CRT, Registre 03h

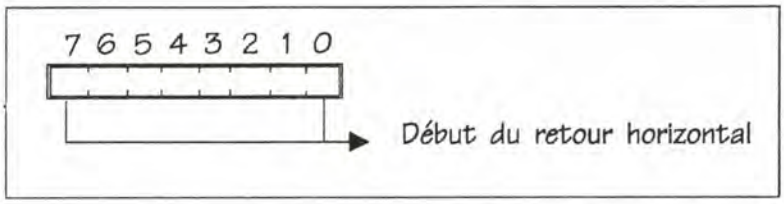


0-7 Pour la fin du Blanking horizontal, l'unité choisie est le caractère comme pour le début. Le premier caractère affiché dans la marge gauche de l'écran porte le numéro 0. Comme la fin du Blanking horizontal se situe toujours avant le début, on n'utilise pas 8 bits mais 6 bits pour coder la fin du Blanking. Le sixième bit se trouve alors dans le registre End Horizontal Retrace portant le numéro 05h dans le contrôleur CRT.

Dans ce registre, la valeur est calculée à partir de la somme des valeurs du registre Horizontal Blanking et la largeur du Blanking horizontal en caractères.

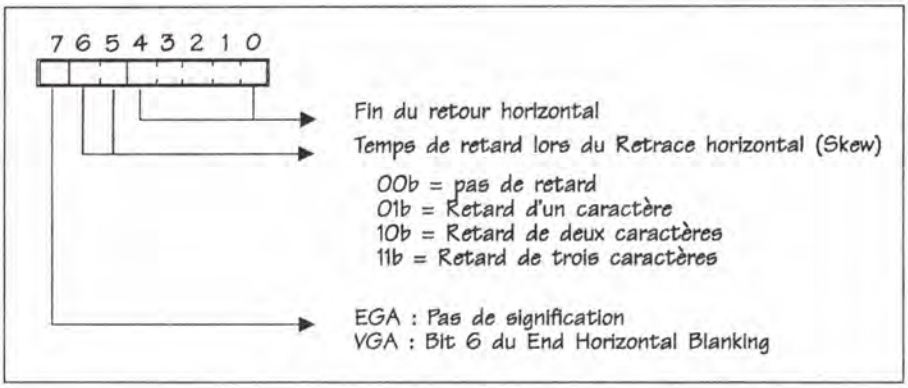
5+6 Dans certaines circonstances, il est utile de définir un délai d'affichage pour que le contrôleur CRT dispose d'un temps suffisant pour charger un caractère et son attribut depuis la RAM vidéo et lire ensuite le modèle de point via le générateur de caractères. Avec les cartes EGA, un caractère a besoin en outre d'un Skew alors qu'il n'est plus utile avec les cartes VGA.

Start Horizontal Retrace — Contrôleur CRT, Registre 04



0-7 Ce registre détermine le caractère après le parcours duquel commence le retour du rayon électronique. L'écran peut ensuite être centré horizontalement à l'aide de ce registre.

End Horizontal Retrace — Contrôleur CRT, Registre 05h

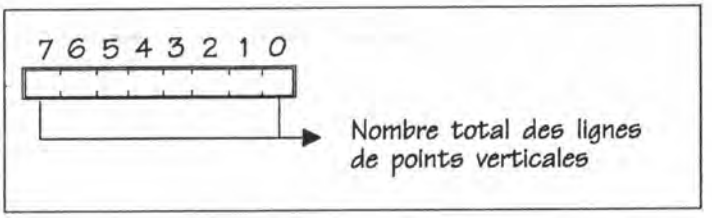


0-4 Détermine la fin du retour horizontal. Etant donné que la fin de ce retour se situe toujours avant le début, il faut uniquement 5 bits pour le coder. L'unité utilisée est à nouveau le caractère.

5+6 Comme pour la fin du Blanking horizontal, on peut définir un délai (Skew) pour la fin du retour horizontal. Il varie d'une carte à l'autre si bien que ces bits ne peuvent jamais être manipulés.

7 Ce bit sert d'extension aux bits 0 à 4 dans le registre End Horizontal Blanking et représente leur bit de poids fort.

Vertical Total Register — Contrôleur CRT, Registre 06h

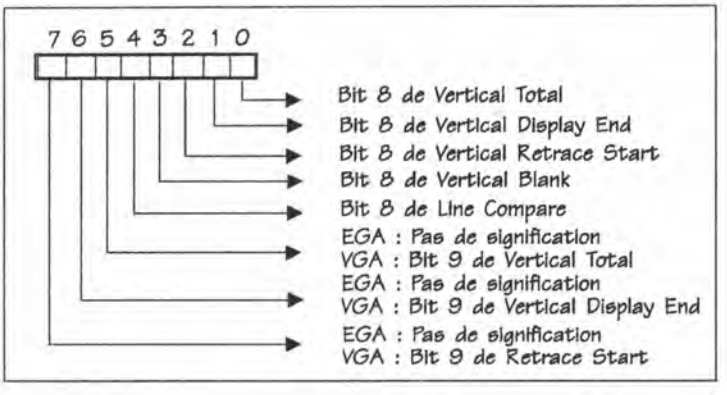


0-7 Ce registre contient le nombre total de lignes de points parcourues lors d'une configuration d'écran. Outre les lignes de points affichées, il faut y ajouter les lignes qui ont pu être parcourues au cours du retour vertical.

Que ce soit avec une carte EGA ou VGA, le nombre total de lignes de points franchissant largement la limite des 256, ce registre ne stocke seulement que les 8 bits de poids faible. Le huitième bit se trouve dans le registre Overflow portant le numéro 07h. Ce registre contient un dixième bit logé dans le bit 5 pour la carte VGA exclusivement.

Par rapport à la valeur réelle, le contenu de ce registre doit toujours être ramené à la valeur 2 que ce soit avec une carte EGA ou VGA.

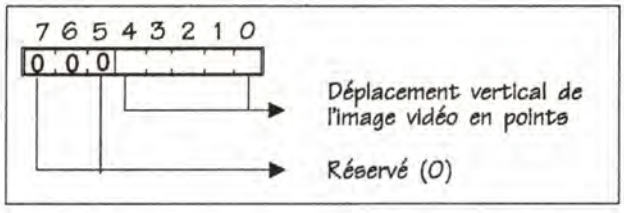
Registre Overflow — Contrôleur CRT, Registre 07h



Un registre Overflow est utile avec les cartes EGA et VGA parce que le nombre des lignes de points verticales dépasse 256. Par conséquent, il manque au moins un bit dans pratiquement tout registre lié au déplacement vertical du rayon électronique. Les bits de poids fort qui dont défaut sont donc regroupés

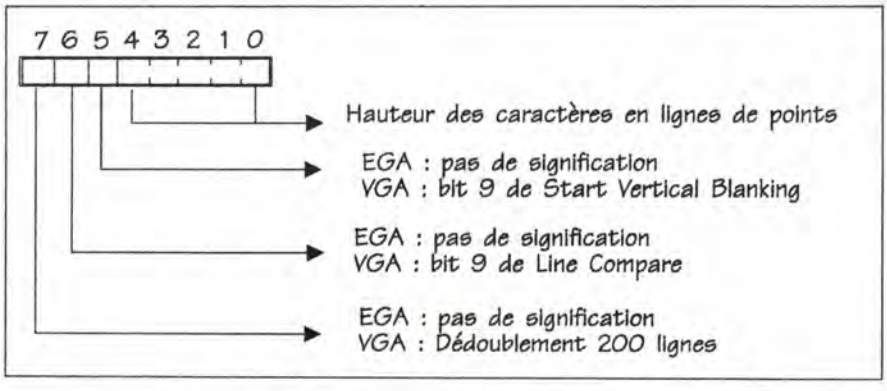
dans ce registre. Les autres bits Overflow sont intégrés dans le registre Maximum Scan Line avec une carte VGA. Il porte le numéro d'index 09h.

Vertical Pel Panning — Contrôleur CRT, Registre 08h



0-4 Ces bits permettent de réaliser un smooth scrolling sur le plan vertical. Dans ce cas, l'écran décale vers le haut ou le bas en fonction d'un nombre de points graphiques déterminés. La valeur 0 représente la position normale, toute valeur supérieure un décalage correspondant vers le haut, parce que lors de la configuration de l'écran, le contrôleur CRT ne commence pas à la ligne de points zéro mais à partir de celle indiquée dans ce registre.

Maximum Scan Line — Contrôleur CRT, Registre 09h



0-4 La hauteur des caractères en mode texte est contrôlée par ce bit en mesurant la hauteur en lignes de points. Par rapport à la hauteur de caractères réelle, la valeur stockée ici doit être réduite de 1.

En mode graphique, ce champ de bit contient normalement la valeur 0 -si un mode graphique VGA avec 200 lignes de points n'est pas actif- et pour

l'affichage duquel les lignes doivent être doublées. Ce registre contient alors la valeur 1.

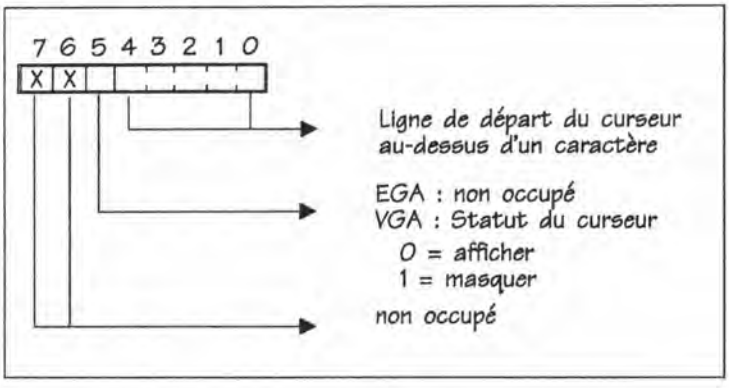
5 Alors que ce bit n'est pas occupé dans une carte EGA, il reçoit le bit 9 du registre Vertical Blank Start avec une carte VGA.

6 Egalement inoccupé sur une carte EGA, le neuvième bit du registre Line Compare existe sur une carte VGA.

7 Ce bit, inoccupé sur une carte EGA, permet de doubler les lignes de points sur une carte VGA dans les modes graphiques qui ne reposent uniquement que sur les 200 lignes de points. Les différentes lignes sont doublées de manière à pouvoir utiliser la résolution VGA normale de 400 lignes de points.

Faites attention à ce que la commutation de ce bit n'entraîne pas une déprogrammation des divers registres impliqués dans le timing vertical de la configuration écran.

Cursor Start Contrôleur CRT, Registre 0Ah

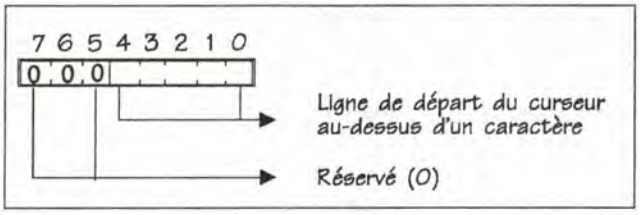


0-4 La ligne de départ du curseur sur le caractère est calculée à partir de ligne 0 et peut être déplacée dans un intervalle de 0 et 31. Si elle franchit la hauteur de caractères réelle en lignes de points, le curseur n'apparaît pas sur l'écran.

Si la ligne de départ du curseur est supérieure à la ligne de fin (registre CRT 0Bh), un curseur en deux parties apparaît avec une carte EGA et pas de curseur avec une carte VGA.

5 La création du curseur ne peut être désactivée par le bit 5 que sur une carte VGA. Ce bit est inoccupé sur une carte EGA.

Cursor End — Contrôleur CRT, Registre 0Bh



0-4 Dans ce champ de bit, la ligne de fin du curseur clignotant est définie par le caractère situé au-dessous. Cette valeur peut être comprise entre 0 et 31 mais ne doit pas dépasser la hauteur d'un caractère en lignes de points.

Si la ligne de fin est inférieure à la ligne de départ (registre CRT OAh), un curseur en deux parties apparaît avec une carte EGA et pas de curseur avec une carte VGA.

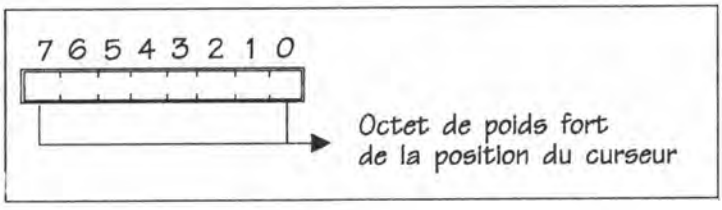
Start Address High — Contrôleur CRT, Registre 0Ch

0-7 En combinaison avec le registre 0Dh, ce registre détermine l'adresse d'offset à partir de laquelle le contrôleur CRT extrait les informations concernant le contenu d'écran depuis la RAM vidéo. Il définit ainsi le début de la page écran actuelle dans la RAM vidéo en divisant toujours par 2 la valeur stockée ici par rapport à l'offset véritable en mode Odd/Even et par 4 en mode Chain4.

Start Address Low — Contrôleur CRT, Registre 0Dh

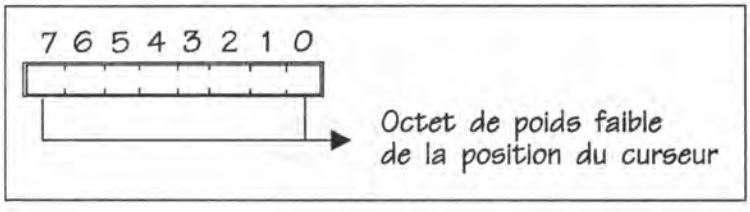
0-7 L'octet de poids faible de l'adresse de départ de la page d'écran actuelle de la RAM vidéo est stockée ici. Reportez-vous également au registre 0Ch.

Cursor Location High — Contrôleur CRT, Registre 0Dh



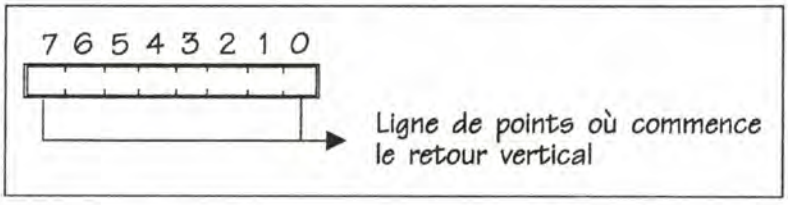
0-7 Ce registre définit la position actuelle du curseur en guise d'offset dans la page écran actuelle. L'adresse indiquée doit être divisée par 2 par rapport à l'adresse de caractères effective. Alors que l'octet de poids fort de cet offset se trouve dans ce registre, l'octet de poids faible est stocké dans le registre qui suit.

Cursor Location Low — Contrôleur CRT, Registre 0Eh



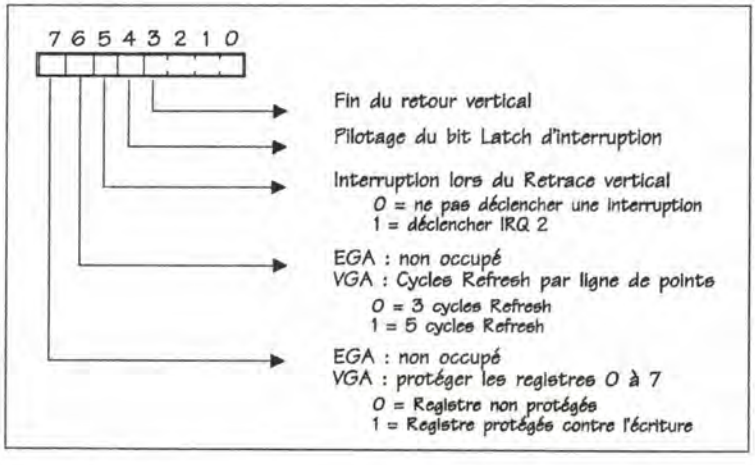
0-7 L'octet de poids faible de la position du curseur dans la page écran actuelle est stocké ici. Reportez-vous également au registre 0Dh.

Start Vertical Retrace — Contrôleur CRT, Registre 10h



0-7 Le contenu de ce registre permet de spécifier la ligne de points où commence le retour vertical du rayon électronique. Comme les cartes EGA et VGA fonctionnent avec plus de 256 lignes de points, les 8 bits prévus ici ne suffisent pas à recevoir cette information. Un neuvième bit est donc intégré dans le registre Overflow qui porte le numéro 07h. Les cartes VGA étant nécessaires pour coder cette ligne de point, le registre Overflow contient alors en plus un dixième bit.

Vertical Retrace End — Contrôleur CRT, Registre 11h



0-3 Stocke la ligne de points qui, une fois atteinte, désactive le signal de synchronisation et commence une nouvelle configuration de l'écran. Comme il n'existe que quatre bits pour recevoir cette information, cela s'effectue au plus tard dans la quinzième ligne de points de l'écran.

4 Le déclenchement d'une interruption verticale fixe sur 1 le bit 7 du registre Input Status pour afficher le début d'un retour vertical. Le bit d'interruption

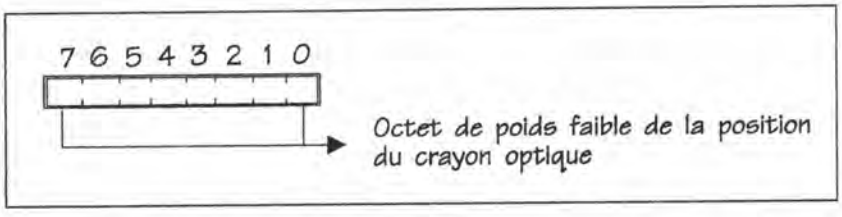
reste actif tant qu'il n'est pas rétabli par l'émission de la valeur 1 dans ce bit. Pendant ce laps de temps, il empêche le déclenchement d'une nouvelle interruption verticale.

5 Le déclenchement de l'interruption 2 au début de chaque retour vertical peut être activé à travers ce bit. Mais dans ce contexte, il ne faut pas oublier que de nombreuses cartes VGA ne peuvent pas provoquer une interruption verticale (comme les cartes IBM des modèles PS).

6 La possibilité de faire passer de 3 à 5 le nombre de cycles de rafraîchissement par ligne de points n'existe que dans les cartes VGA. Le temps nécessaire au rafraîchissement de la RAM vidéo étant plus élevé, VGA utilise ne fréquence de lignes moindre et peut fonctionner sur des moniteurs qui ne soutiennent pas la fréquence de lignes VGA normale.

7 L'accès aux huit premiers registres du contrôleur CRT peut être verrouillé par ce bit sur les cartes VGA. S'il est mis, ces registres peuvent être lus mais sont interdits à l'écriture.

Light Pen Low — EGA uniquement, lors des accès en lecture



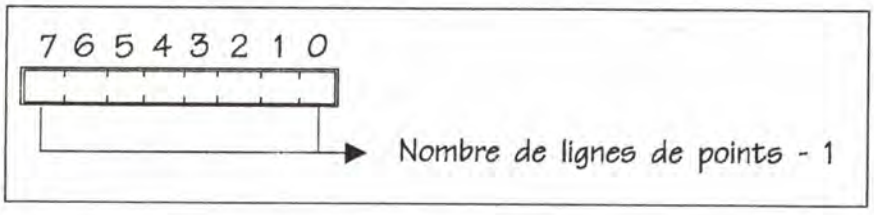
0-7 Le registre 10h déjà décrit ne peut être utilisé différemment pour les accès en lecture et écriture qu'avec des cartes VGA. Lors d'un accès en écriture, il reçoit le début du retour vertical mais la position actuelle du crayon optique lors d'un accès en lecture. Comme pour la position du curseur, cette valeur reproduit la position d'offset du caractère sur lequel se trouve le crayon optique, cette valeur étant divisée par le facteur deux.

Avec les cartes VGA qui ne soutiennent pas l'utilisation du crayon optique, ce registre retourne le contenu déjà décrit du registre Vertical Retrace Start lors d'un accès en lecture.

Light Pen High — EGA uniquement, lors des accès en lecture

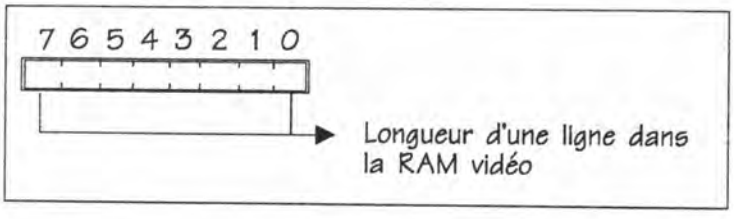
0-7 Avec des cartes EGA, ce registre retourne l'octet de poids fort de la position du crayon optique lors d'un accès en lecture. Reportez-vous également à la description du registre 10h.

End Vertical Display — Contrôleur CRT, Registre 12h



0-7 Ce registre conserve le numéro des lignes de points indiquant la fin de la configuration de l'écran. Leur codage nécessitant 9 bits sur une carte EGA et 10 bits sur une carte VGA, il n'est possible de stocker ici que 8 bits. Les bits supplémentaires se trouvent dans le registre Overflow portant le numéro 07h.

Registre d'offset — Contrôleur CRT, Registre 13h

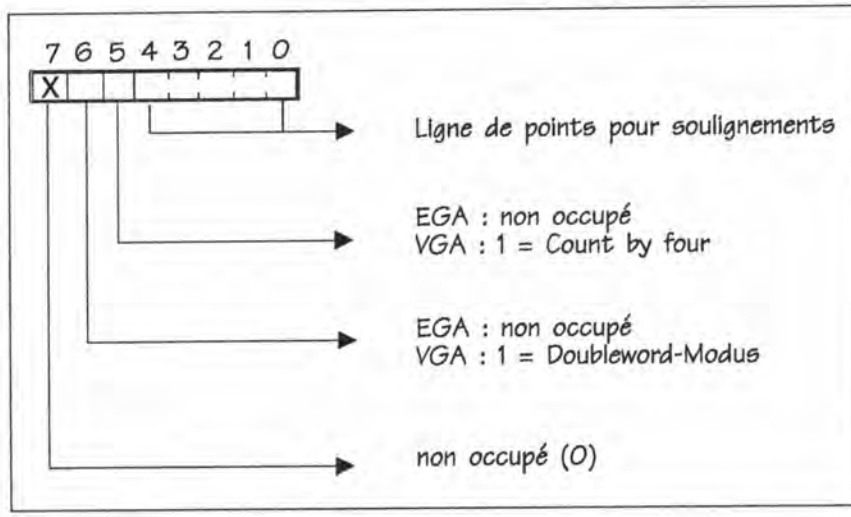


0-7 L'offset que le contrôleur CRT ajoute à l'adresse d'offset de la ligne précédente à chaque début de ligne est conservé dans ce registre. Par rapport à l'offset

réel, la valeur doit toutefois être divisée par un facteur déterminé compte tenu du mode d'adressage. En mode Odd/Even, il correspond à 2, en mode Chain4 à 4 et en mode octet normal à 1.

Normalement, la valeur de ce registre correspond à la longueur d'une ligne dans la RAM vidéo. En mode texte, considéré sur le plan interne comme le mode Odd/Even, il contient la valeur 80 parce qu'une ligne de texte reçoit 160 octets, soit 80 mots. La valeur peut être supérieure ou inférieure mais il faut en tenir compte au moment de compléter la RAM vidéo par des informations de points.

Underline Location — Contrôleur CRT, Registre 14h



0-4 Lorsque les cartes EGA et VGA fonctionnent en mode monochrome, elles peuvent souligner les caractères à l'écran. Ce champ de bits détermine la ligne de points où le soulignement doit apparaître.

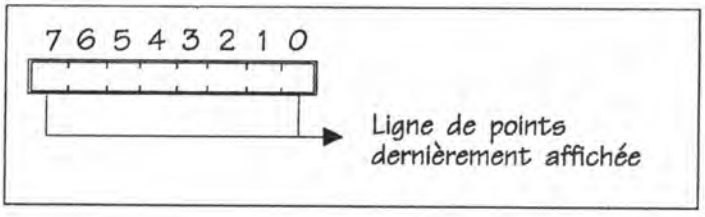
5 Exclusivement pour les cartes VGA, ce bit décide si le compteur d'adresse interne ne doit être incrémenté que tous les quatre coups du Character Clock. A cet effet, ces bits doivent afficher la valeur 1 et le bit 3 du registre Mode CRT (Count by two) la valeur 0. Si ce bit contient au contraire la valeur 1, le mode Doubleword est ignoré dans tous les cas.

6 Avec des cartes VGA, ce bit permet d'activer le mode Doubleword. Le contenu des bits Access Mode du registre Mode CRT est alors ignoré.

En mode Doubleword, l'adresse issue du compteur d'adresses interne pendant

la configuration de l'écran lors de l'accès à la RAM vidéo est décalée de 2 bits vers le haut alors que les bits 14 et 15 se promènent simultanément entre la position 0 et 1. Tant que le compteur d'adresses est inférieur à 4000h, toutes les cellules mémoire entre 0000h et FFFCh dont l'adresse modulo quatre donne la valeur zéro sont adressées. Lorsque le compteur d'adresses interne atteint la zone comprise entre 4000h et 7FFFh, toutes les cellules mémoire entre 0001h et FFFDh dont l'adresse module quatre donne la valeur un sont adressées. Le même procédé s'applique aux zones situées entre 8000h et BFFFh ou C000h et FFFFh. Mais les cellules mémoire adressées ici sont des cellules dont l'adresse modulo quatre donne la valeur deux ou trois.

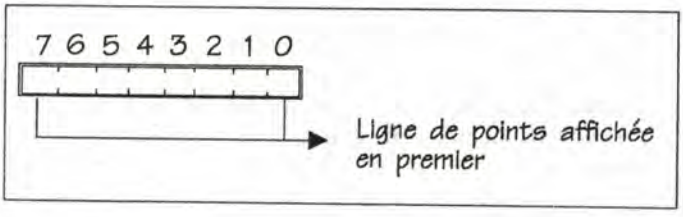
Start Vertical Blanking — Contrôleur CRT, Registre 15h



0-7 Ce registre conserve le numéro de la dernière ligne de points visible plus 1. Le neuvième bit nécessaire aux cartes EGA et VGA est géré dans le registre Overflow (registre 07).

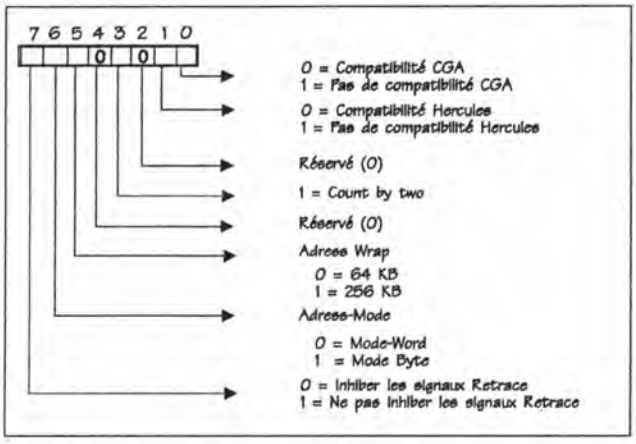
Un dixième bit s'avère en plus indispensable aux cartes VGA. Il se trouve dans le registre Maximum Scan Line portant le numéro 09h.

End Vertical Blanking — Contrôleur CRT, Registre 16h



0-7 Ce registre conserve la première ligne de points qui, une fois atteinte, désactive le signal Blanking vertical pour permettre une reconfiguration de l'écran.

Mode Control Contrôleur CRT, Registre 17h



0 Si un programme a mis ce bit sur 0, on peut simuler la structure de la RAM vidéo en CGA en utilisant d'autres registres dans une émulation du mode CGA 320*200 points 4 couleurs. Dans ce mode, CGA divise notamment la mémoire vidéo en deux blocs commençant à l'adresse d'offset 0000h et 2000h dans la RAM vidéo. Le premier bloc contient toutes les lignes paires, le second bloc les lignes impaires.

Pour émuler ce procédé, ce bit autorise le transfert du bit 0 depuis le compteur interne Row Scan vers le bit 13 du compteur d'adresses interne contenant les adresses qui permettent au CRT de lire les informations de points depuis la RAM vidéo.

Le bit 13 possédant la valeur 2000h, il en résulte que le CRT se promène en permanence entre les deux blocs aux adresses 0000h et 2000h parce que le bit 0 du compteur interne Row Scan alterne également en permanence entre 0 et 1. La condition préalable à cela est que la hauteur de caractères soit définie à deux à travers le registre Maximum Scan Line. A cet effet, il faut charger la valeur 1 dans ce registre.

1 Ce bit représente une extension du bit 0. Il doit être activé, c'es-à-dire mis sur 0, lorsqu'il s'agit de reconfigurer un mode vidéo étranger que la RAM vidéo divise en quatre blocs. C'est par exemple le cas de la carte graphique Hercules et d'autres aptateurs similaires CGA offrant une résolution de 320*400 points en 16 couleurs.

Dans un tel mode, la RAM vidéo est divisée en quatre blocs commençant aux adresses d'offset 0000h, 2000h, 4000h et 6000h. Le bloc contenant les différentes lignes est calculé par le modulo de son numéro de ligne par quatre.

Comme si la ligne 0 se trouvait dans le premier bloc, la ligne 2 dans le second, la trois dans le troisième et la quatre dans le quatrième.

Pour retracer cette organisation, le bit 1 du compteur interne Row Scan est copié dans le bit 14 du compteur d'adresses interne. Il faut néanmoins s'assurer que la hauteur de caractères est réglée sur quatre dans le registre Maximum Scan Line pour que le bit 1 du compteur Row Scan atteigne au moins une fois la valeur 1 et ainsi le bit 14 soit réglé sur 1 dans le compteur d'adresses. La valeur dans le compteur d'adresses est dans tous les cas supérieure à 4000h pour que les blocs deux et trois puissent être adressés.

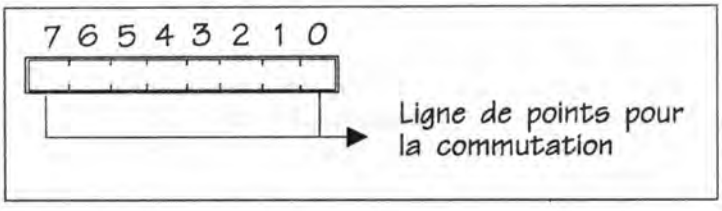
3 Si ce bit contient la valeur 0, le compteur d'adresses interne avance d'un octet à chaque coup frappé sur le Character Clock. Mais avec la valeur 1, ce bit veille à ce que ce compteur avance tous les deux coups.

5 Les cartes EGA équipées seulement de 64 Ko de RAM vidéo doivent régler ce bit sur 0 pour qu'un débordement n'ait pas lieu en mode Word (cf. bit 6). Le bit 13 est alors défini sur le bit 0 de la ligne d'adresse et non le bit 15 du compteur d'adresses interne.

6 Normalement, le mode Byte actif est celui dans lequel la valeur du compteur d'adresses interne est définie telle quelle aux 16 lignes d'adresses déterminant l'octet adressé dans la RAM vidéo.

Si ce bit est au contraire chargé avec la valeur 0, une commutation s'effectue vers le mode Word. Les différents bits d'adresse du compteur d'adresses interne sont alors décalés d'un bit vers la gauche et le bit de poids fort est placé dans la ligne d'adresse de poids faible A0. Tant que le compteur d'adresses est inférieur à 8000h, les octets adressés à cet effet sont les octets pairs situés entre 0000h et 0FFFEh. Toute valeur supérieure provoque l'adressage d'un octet impair compris entre 0001h et FFFFh.

Line Compare — Contrôleur CRT, Registre 18h



0-7 Deux blocs différents de la RAM vidéo peuvent se partager l'écran à travers ce registre. A cet effet, ce registre doit recevoir le numéro de la ligne de points

où doit s'effectuer le transfert entre le premier et le second bloc. Une fois cette ligne atteinte, le contrôleur CRT rétablit sur 0 l'adresse d'offset interne nécessaire pour demander les informations de points depuis la RAM vidéo. Ainsi, le contenu d'écran est extrait du début de la RAM vidéo.

Le démarrage d'un nouveau parcours de l'écran provoque l'affichage du bloc d'écran dont l'adresse figure dans les registres CRT correspondants (registres 0Ch et 0Dh).

Les cartes EGA et VGA affichant plus de 256 lignes de points à l'écran, les 8 bits prévus ici pour recevoir l'information nécessaire ne suffisent pas dans tous les cas. C'est pourquoi un neuvième bit se trouve dans le registre Overflow (registre 07), un dixième dans le registre Maximum Scan Line (registre 09) dans le cas de cartes VGA.

Le séquenceur

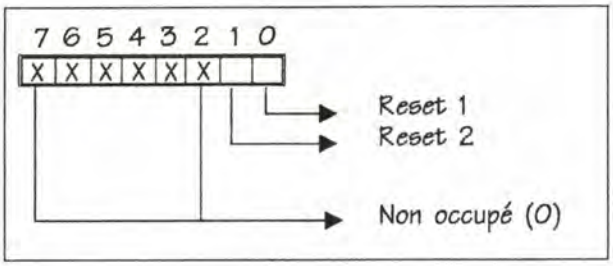
L'accès aux registres du séquenceur s'effectue selon le schéma habituel c'est-à-dire à travers un registre de données et un registre d'index. Le registre d'index se trouve à l'adresse de port 3C4h et précède indiscutablement le registre de données situé à l'adresse de port 3C5h.

Contrairement aux autres contrôleurs d'une carte EGA ou VGA, le séquenceur ne connaît pas beaucoup de limitations parce qu'il réalise des tâches à la fois diversifiées et totalement différentes. Par exemple, il convertit les accès à la RAM vidéo et les redirige vers les différents plans de bits ou sélectionne la table de caractères active. Il a en outre la charge de rafraîchir la RAM vidéo qui est une tâche liée au début du retour horizontal dans chaque ligne de points.

Avec les cartes EGA et VGA, le séquenceur dispose en tout de cinq registres différents comme le montre le tableau suivant. Attention ! Avec une carte VGA, les registres peuvent uniquement être lus ce qui n'est pas le cas avec les cartes EGA.

Nº	Nom de registre
00h	Reset
01h	Clocking Mode
02h	Map Mask
03h	Character Map Select
04h	Memory Mode

Reset (Séquenceur, Registre 00h)

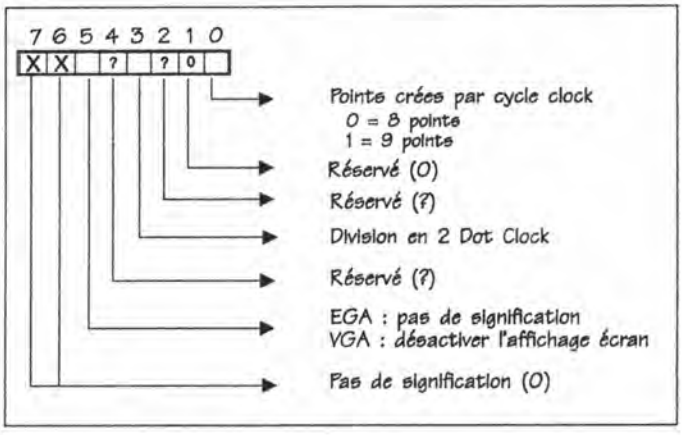


0 Ce bit contient normalement la valeur 1 et peut être mis sur 0 pour effectuer un reset du séquenceur autorisant la fin de son travail. Cela met fin à la création des signaux de synchronisation horizontaux et verticaux (l'écran devient noir) et le registre Character Map Select est chargé avec 0 tout en regénérant la RAM vidéo. Pour que son contenu reste conservé, il faut que ce bit soit remis sur 1 au plus tard au bout de 20 à 30 microsecondes. Le séquenceur est ainsi ravivé.

Un reset du séquenceur s'avère indispensable avant la déprogrammation des bits 0 et 3 du registre Clocking Mode du séquenceur ainsi que des bits 2 et 3 du registre Miscellaneous Output.

1 Ce bit fonctionne de manière similaire au bit 0 mais ne rétablit pas le registre Character Map Select en cas de reset. Alternativement, on peut faire appel au bit 0 pour un reset mais les deux bits doivent posséder la valeur 1 pour que le séquenceur puisse reprendre son travail.

Clocking Mode — Séquenceur, Registre 01h

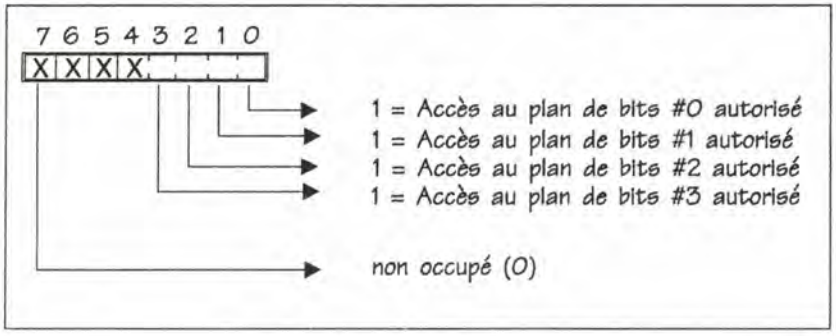


0 Ce bit permet de déterminer le nombre de points horizontaux créés par le contrôleur CRT par cycle Clock. Dans les modes graphiques et les modes texte en couleur de la carte EGA, ce nombre est toujours de huit. Mais dans les modes texte de la carte VGA et en cas de connexion d'un moniteur MDA à une carte EGA, ce bit est toujours réglé sur 1 pour créer neuf points par caractère.

3 Dans ce bit, la division par deux du Dot Clock Rate peut être obtenue par la valeur 1. Cette opération s'effectue automatiquement lors la commutation du mode vidéo à l'aide du BIOS en mode 320*200 points servant à émuler la carte graphique CGA. Ce bit contient la valeur 0 dans tous les autres modes (y compris le mode 320*200 points en 256 couleurs).

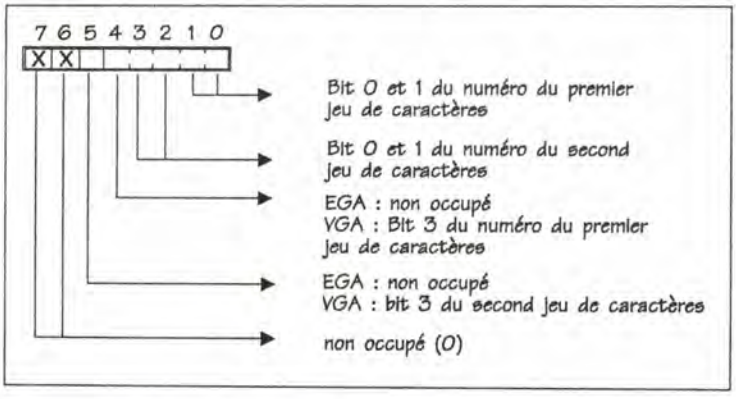
5 Avec une carte VGA, la création d'un signal vidéo peut être désactivée à travers ce bit s'il est chargé avec la valeur 1. La conséquence est que l'écran ne devient pas noir et la CPU autorise l'accès à la RAM vidéo.

Map Mask Séquenceur, Registre 02h



0-3 Chaque bit cité libère ou verrouille l'accès à un plan de bits. Cela est utile pour l'accès à la RAM vidéo en liaison avec les différents modes de lecture/écriture. Lors d'un accès en écriture, le bit concerné décide si un octet du plan de bits doit être complété ou non par le contenu du registre Latch correspondant. De même, lors d'un accès en lecture, chaque bit détermine si le contenu de l'octet adressé dans le plan de bits parvient dans le registre Latch adéquat ou si le contenu du registre Latch reste inchangé.

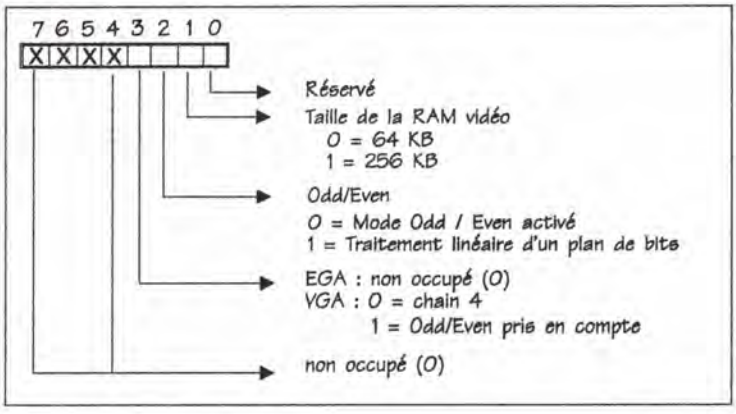
Character Map Select Séquenceur, Registre 03h



0+1+4 Ces bits déterminent le numéro de la table de caractères utilisée pour tous les caractères dont le bit 3 n'est pas fixé dans l'octet d'attribut. Avec une carte EGA, le bit 4 ne joue aucun rôle si bien qu'il n'est possible de sélectionner que les jeux 0 à 3. En revanche avec une carte VGA, le choix s'effectue parmi les jeux 0 à 7.

2+3+5 Ici est stocké le numéro de la table de caractères déterminant l'image de sortie de tous les caractères dont le bit 3 est réglé dans l'octet d'attribut. Dans ce cas également, le bit 5 de la carte EGA ne joue aucun rôle alors qu'il sert comme bit de poids fort du numéro du jeu avec une carte VGA.

Memory Mode Séquenceur, Registre 04h



1 Ce bit n'est intéressant que pour les cartes EGA parce que les cartes VGA sont généralement équipées de 256 Ko de RAM vidéo. Mais en ce qui concerne les cartes EGA, on peut considérer qu'il n'existe plus de cartes avec seulement 64 Ko.

2 Ce bit permet de définir la répartition des adresses de mémoire paires et impaires en divers plans de bits, le fameux mode Odd/Even. Dans ce mode, les accès aux adresses de mémoire paires sont redirigées automatiquement vers les plans de bits 0 et 2 et les accès aux adresses impaires vers les plans de bits 1 et 3. Dans les deux cas, le bit de poids faible de l'adresse d'offset est étendu au bit de page (bit 5 du registre Miscellaneous Output). Comme auparavant, l'accès aux divers plans de bits peut être interdit à travers le registre Map Select du séquenceur.

La répartition des adresses de mémoire ne s'effectue pas lorsque ce bit contient la valeur 1 ce qui permet de traiter linéairement les divers plans de bits.

Attention ! Le contenu de ce bit doit toujours correspondre au bit Odd/Even du registre 5 du contrôleur graphique même si ce registre suit une logique inverse.

3 En guise d'extension du mode Odd/Even, la carte VGA reconnaît le mode Chain4, une sorte de mode Odd/Even dédoublé. Il sert surtout dans les modes graphiques affichant 256 couleurs pour créer une RAM vidéo linéaire en A000h selon le point de vue du programme. En réalité, elle se répartit sur les quatre plans de bits. Cela ne concerne pas seulement l'accès à la RAM vidéo via l'unité centrale -cela ne produit aucun effet sur l'adressage de la RAM vidéo via le contrôleur CRT.

Tout comme dans le mode Odd/Even, les accès à la RAM vidéo sont redirigés vers l'un des quatre plans de bits en fonction de leur adresse où les deux bits inférieurs de l'adresse d'offset sont masqués ou mis sur 0. Dans les plans de bits, seules les cellules mémoire dont l'adresse d'offset est divisible par quatre sont adressées. Les deux bits masqués de l'adresse d'offset, soit le modulo de l'adresse d'offset modulo quatre, permettent de savoir quels sont les plans de bits adressés.

L'accès aux plans de bits ne s'effectue que si le plan de bits correspondant a été librement activé par le registre Map Mask du séquenceur.

Les bits assurant le contrôle du mode Odd/Even sont ignorés lorsque le mode Chain4 est actif. Ils ne redeviennent valides que lorsque le mode Chain4 est désactivé.

Le contrôleur d'attributs

Parmi les divers contrôleurs d'une carte EGA ou VGA, le contrôleur d'attributs a pour tâche de préparer les signaux de couleur pour l'écran. Il contrôle les registres de palette ainsi que des registres intervenant dans la génération du signal de couleur.

Les registres de ce contrôleur sont adressés à travers un registre d'adresses et de données combiné. Pour les accès en écriture aux divers registres, ce registre se trouve à l'adresse de port 3C0h, pour les accès en lecture -autorisés uniquement avec une carte VGA- à l'adresse de port 3C1h.

Pour accéder à l'un des registres du contrôleur, il faut obligatoirement inscrire le numéro du registre dans le port 3C0h ou 3C1h. Lors d'un accès en lecture, le contenu du registre peut être extrait immédiatement à partir du port 3C1h. Mais lors d'un accès en écriture, le nouveau contenu du registre doit être envoyé au port 3C0h pour parvenir dans le registre souhaité.

Dans le registre d'index et de données combiné situé à l'adresse de port 3C1h, il suffit d'entrer le numéro du registre concerné lors d'un accès en lecture. Mais son homologue contient une information supplémentaire décidant du statut du contrôleur d'attributs. Il suffit que ce bit soit mis sur 0 pour que le lien entre le contrôleur d'attributs et le contrôleur CRT soit annulé. L'écran devient alors noir ou affiche la couleur spécifiée dans le registre

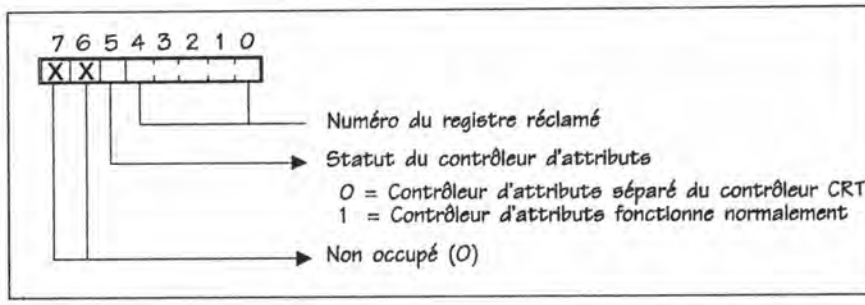
Overscan du contrôleur d'attributs. Un signal de couleur individuel pour les caractères ou points n'est plus créé sur l'écran.

Avec une carte EGA, le contrôleur d'attributs dispose en tout de 20 registres auquel s'y ajoute un registre dans le cas d'une carte VGA. L'affectation de certains registres présente des différences selon qu'il s'agit d'une carte EGA ou VGA. Pour créer un signal de couleur, une carte VGA doit tenir compte de la table de couleurs DAC alors que cette dernière n'existe pas dans une carte EGA.

Comme nous l'avons déjà signalé pour le séquenceur, les registres peuvent uniquement être lus avec des cartes VGA contrairement aux cartes EGA avec lesquelles ils sont "write only".

Nº	Nom de registre
00h-0Fh	Registres de palettes
10h	Mode Control
11h	Overscan Color
12h	Color Plan Enable
13h	Horizontal Pel Panning
14h	Color Select (uniquement VGA)

Address (Contrôleur d'attributs, Registre d'adresses)

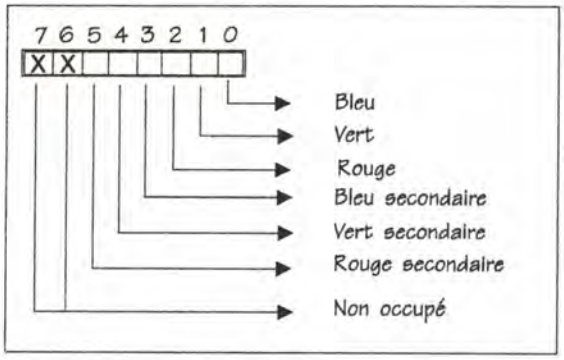


0-4 Ces bits servent à adresser les registres du contrôleur d'attributs comme il est d'usage avec les autres contrôleurs d'une carte EGA/VGA.

5 Le contrôleur d'attributs fonctionne normalement lorsque ce bit est actif. S'il est annulé, le contrôleur d'attributs est séparé du contrôleur CRT ce qui fait noircir momentanément l'écran à moins qu'il n'affiche la couleur conservée dans le registre Overscan.

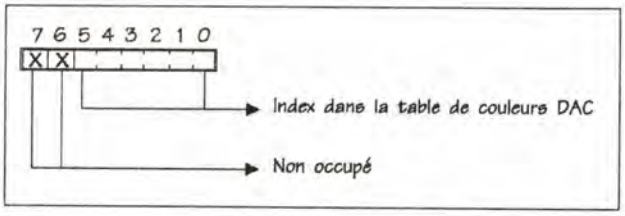
C'est le seul et unique moment où les registres de palettes peuvent être adressés à partir de l'unité centrale pour les charger ou écrire dans ces derniers.

Palette	Contrôleur d'attributs, Registres 00h à 0Fh, EGA



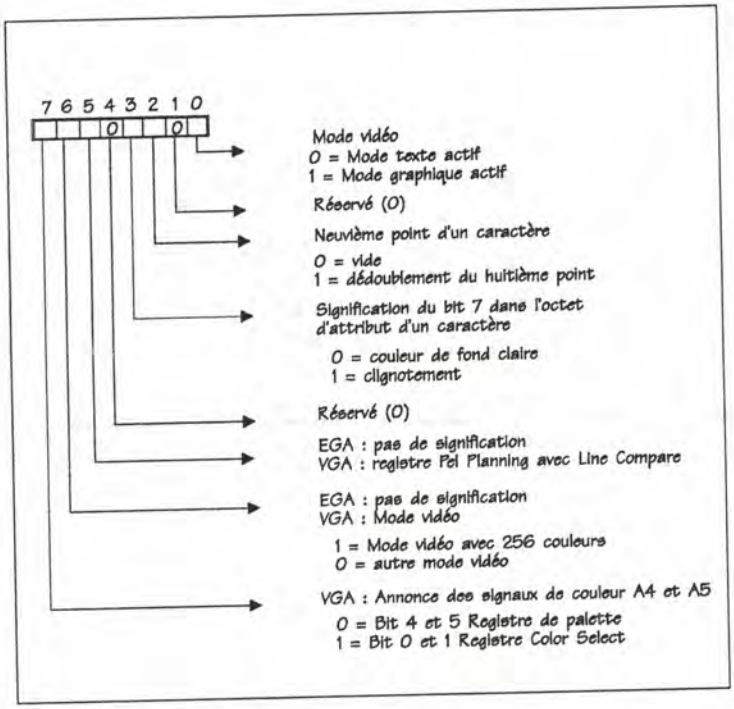
0-5 Avec une carte EGA, les 16 registres de palettes reçoivent un code de couleur tout à fait concret construit à partir de 6 bits et autorisant ainsi 64 types de combinaisons.

Palette	Contrôleur d'attributs, Registres 00h à OFh, VGA



0-5 Contrairement aux cartes EGA, les registres de palettes ne reçoivent pas un code de couleurs avec des cartes VGA mais tout simplement un indice dans la table de couleurs DAC. Cet indice permet au contrôleur d'attributs de charger la couleur d'un caractère depuis la table de couleurs DAC et envoyer les signaux de couleur adéquats au moniteur.

Mode Control — Contrôleur d'attributs, Registre 10h



0 Ce bit informe le contrôleur d'attributs quel est le mode actuellement actif (mode texte ou graphique).

2 Ce bit n'est intéressant que pour les modes texte où les caractères sont représentés avec une largeur de neuf points. Il permet d'établir l'origine du neuvième point parce que les tables de caractères en ROM ne contiennent que huit points par ligne.

Le neuvième point reste vide si ce bit contient la valeur 0. Ainsi se crée un espace d'un pixel entre deux caractères consécutifs. Si ce bit est actif, la procédure dépend du caractère concerné. S'il s'agit d'un caractère de cadre (codes ASCII entre C0h et DFh), le neuvième point est copié à partir du huitième. Le neuvième point reste toutefois vide pour les autres caractères.

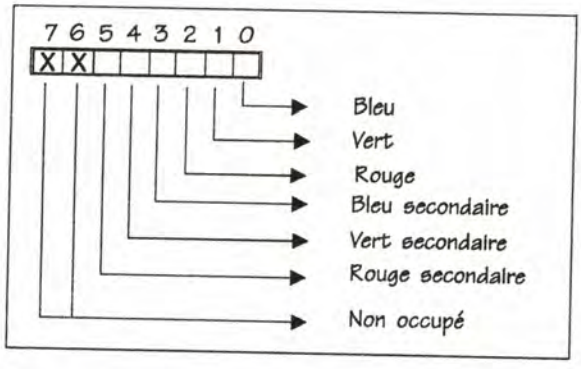
3 Le réglage de ce bit active la création d'un curseur en mode texte. Le curseur apparaît à l'écran pendant 16 répétitions d'images consécutives puis reste invisible pendant les 16 répétitions suivantes. Selon la fréquence de configuration de l'écran, cette durée se situe entre 1/3 et 1/4 de seconde.

5 Si l'écran est divisé en deux portions à l'aide du registre Line Compare, ce bit décide si un Panning horizontal et vertical doit s'effectuer sur la première portion de l'écran ou sur les deux parties dans le cas d'une carte VGA. Pour que l'action s'applique aux deux parties de l'écran, il faut entrer la valeur 0 dans ce bit.

6 La mise en service de ce bit règle le contrôleur d'attributs sur un mode graphique 256 couleurs où les informations de couleur ne sont pas lues à travers les registres de palettes mais directement depuis les registres DAC.

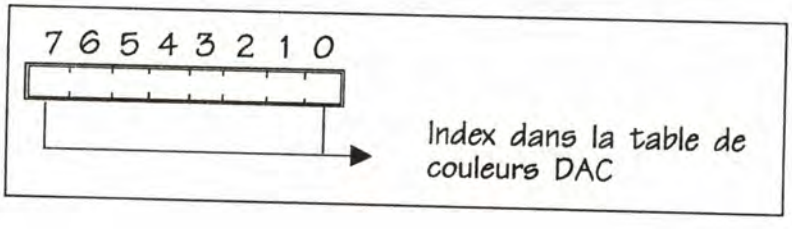
7 Dans un mode avec 16 couleurs ou moins, ce bit décide si les signaux de couleurs A4 et A5 pour l'adressage des registres DAC doivent être extraits du registre de palettes correspondant en même temps que les signaux A0 à A3 ou à partir des bits 0 et 1 du registre Color Select.

Overscan Contrôleur d'attributs, Registre 11h, EGA



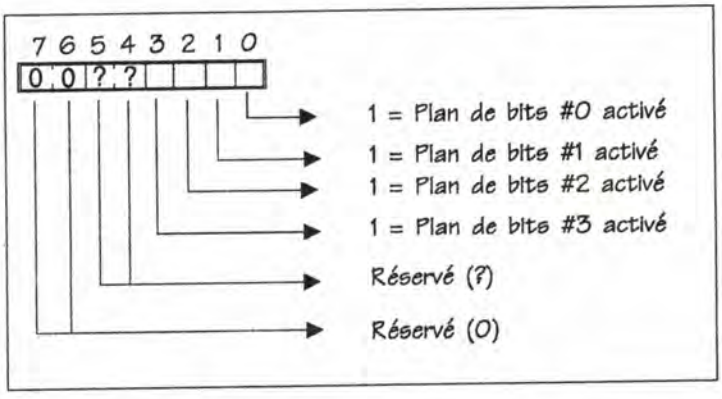
0-5 Avec une carte EGA, ces bits conservent la couleur du cadre de l'écran en gardant le même format que celui utilisé dans les registres de palettes.

Overscan Contrôleur d'attributs, Registre 11h, VGA



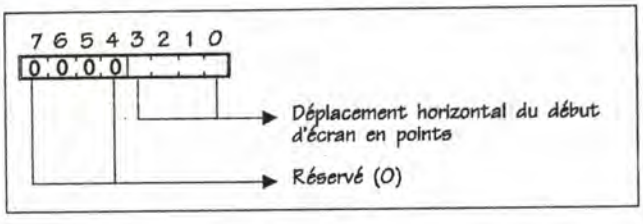
0-7 Avec une carte VGA, la couleur du cadre est sélectionnée parmi les couleurs de la table DAC. Les 256 entrées peuvent être adressées à travers ce registre.

Color Plane Enable — Contrôleur d'attributs, Registre 12h



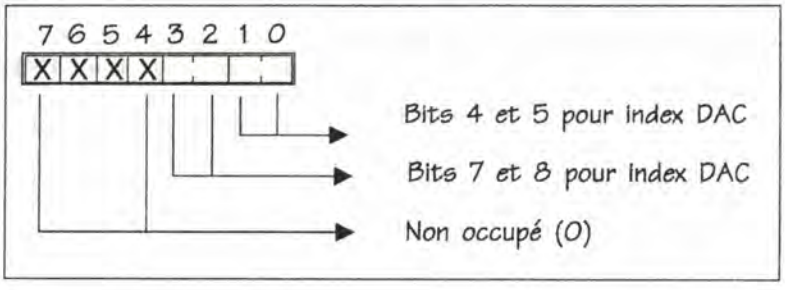
0-3 Les quatre bits de ce champ permettent d'activer ou de désactiver individuellement les différents plans de bits lors de la transmission des informations de couleurs depuis la RAM vidéo vers le contrôleur d'attributs. Grâce à une programmation des registres de palettes ou DAC, il est ainsi possible d'exclure des points précis de l'affichage.

Horizontal Pel Panning — Contrôleur d'attributs, Registre 13h



0-3 Le compteur nécessaire au Pel Panning horizontal est stocké ici. Il détermine le nombre de bits à prendre en compte pour décaler la portion visible de l'écran vers la gauche. Reportez-vous également en 4.8.3.

Color Select	Contrôleur d'attributs, Registre 14h, VGA uniquement



0+1 Ces deux bits interviennent lorsque le bit 7 est actif dans le registre Mode Control du contrôleur d'attributs. Ils remplacent alors les bits 4 et 5 du registre de palettes concerné.

2+3 Dans tous les modes graphiques utilisant moins de 256 couleurs, ces deux bits déterminent les bits 6 et 7 pour l'indice de la table de couleurs DAC indépendamment d'un autre registre ou champ de bits.

Contrôleur graphique

Le contrôleur graphique est utilisé lors de tout accès en lecture et écriture à la RAM vidéo. Il contrôle le transfert effectué depuis l'unité centrale vers la RAM vidéo en passant par les registres Latch. Son cahier de registres contient les registres qui déterminent le mode Read/Write actuel ainsi que tous les registres qui reçoivent des informations sur les paramètres des divers modes.

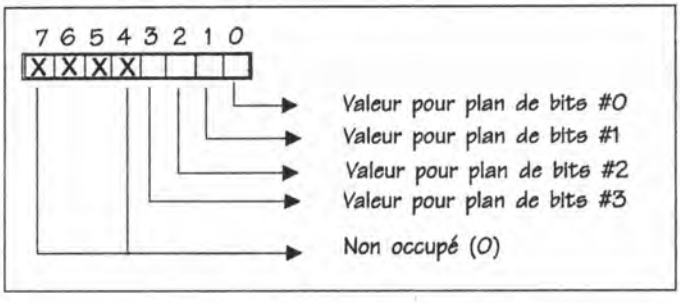
Un registre de données et d'index situé à l'adresse de port 3CFh ou 3CEh permet d'adresser les neuf registres du contrôleur graphique. Comme à l'accoutumée, il faut transmettre d'abord le numéro du registre à adresser dans le registre d'index (3CEh). L'accès au registre souhaité est ainsi ouvert qui doit ensuite déboucher sur un accès en lecture ou écriture au registre de données (3CFh).

Lors de l'accès aux registres de ce contrôleur, n'oubliez pas que vous avez seulement le droit de les lire avec des cartes VGA et que les cartes EGA refusent également la lecture de ces registres.

Voici la liste des registres du contrôleur graphique :

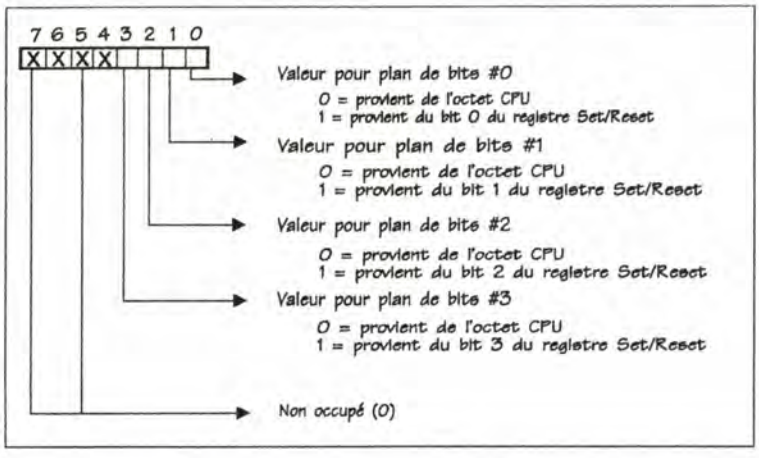
Nº	Nom de registre
00h	Set/Reset
01h	Enable Set/Reset
02h	Color Compare
03h	Function Select
04h	Read Map Select
05h	Graphics Mode
06h	Miscellaneous
07h	Color Don't Care
08h	Bit Mask

Set/Reset (Contrôleur d'attributs, Registre 00h)



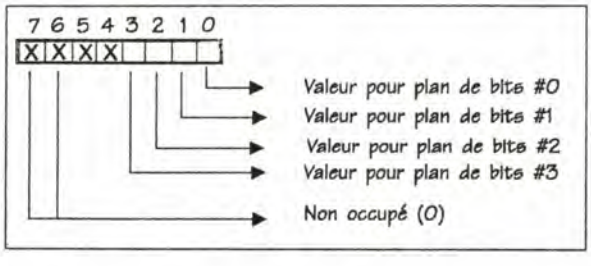
0-3 Ces bits interviennent pour l'accès à la RAM vidéo en mode Write 0. Ils déterminent individuellement la valeur pour les différents plans de bits. Cette valeur doit être inscrite dans les 8 bits du plan concerné lorsque l'origine des données transmises est définie à travers le registre Enable Set/Reset (registre 02h).

Enable Set/Reset — Contrôleur graphique, Registre 01h



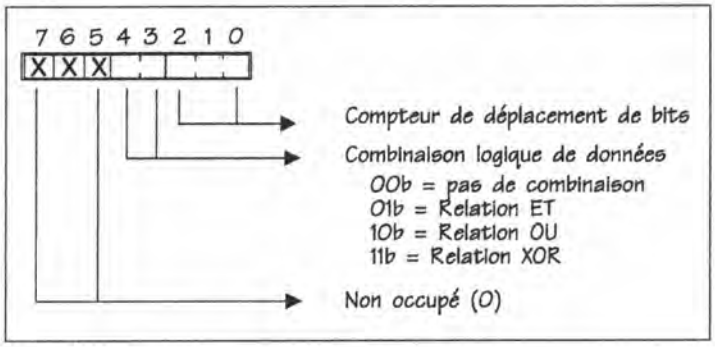
0-3 Pour l'utilisation du mode Write 0, ces bits déterminent si la valeur inscrite dans les 8 bits de l'octet d'un plan de bits adressé doit être extraite de l'octet CPU ou du registre Set/Reset (registre 00h).

Color Compare — Contrôleur graphique, Registre 02h



0-3 Le contenu de ces bits est utile pour les accès en lecture à la RAM vidéo en mode Read 1 des cartes EGA et VGA. Dans ce mode, les quatre octets lus dans les quatre plans de bits de la RAM vidéo sont regroupés en huit groupes de quatre bits représentant la couleur d'un point. Les huit codes de couleur ainsi obtenus sont comparés individuellement avec le contenu de ce registre. Le résultat de la comparaison est transmis à l'unité centrale.

Function Select — Contrôleur graphique, Registre 03h

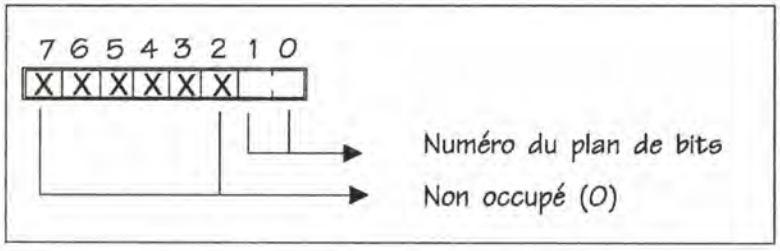


0-2 Pour les accès en écriture à la RAM vidéo en modes Write 0 et 3, ces bits déterminent le nombre de positions de bits à prendre en compte pour faire subir une rotation vers la gauche à l'octet CPU avant qu'il ne soit combiné avec le contenu du registre Latch.

3+4 En modes Write 0 et 2, ce champ de bits définit une combinaison logique à établir entre un octet CPU et les données des quatre registres Latch avant qu'elles ne soient inscrites dans les quatre plans de bits.

Les quatre bits du registre Bit Mask correspondant chacun à un registre Latch décident si une telle combinaison doit s'effectuer ou non. Une combinaison ne se crée en fait que si le bit correspondant reçoit la valeur 1. Ces bits décident alors de la nature de cette combinaison.

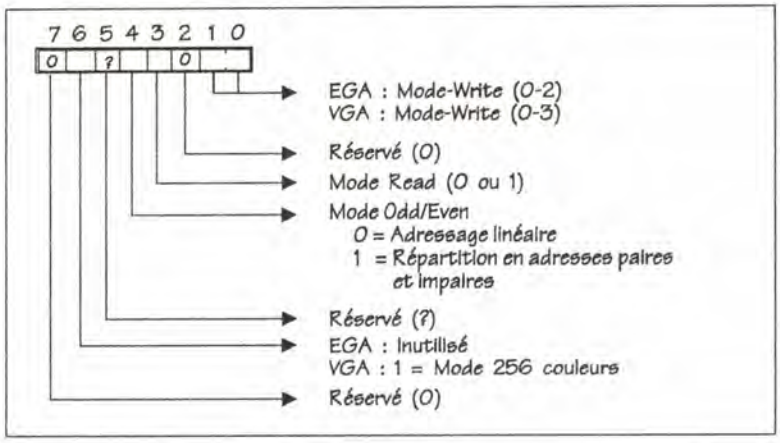
Read Map Select — Contrôleur graphique, Registre 04h



0+1 Pour l'accès en lecture en mode Read 0, le numéro du plan de bits et donc le numéro du registre Latch est conservé dans ces deux bits. Son contenu doit être transmis à l'unité centrale. Les accès en lecture en mode Read 1 subissent

une influence par voie de conséquence tout comme les accès en lecture en mode Chain4 qui peut être défini à travers le bit 3 du registre Memory Mode du séquenceur.

Graphics Mode — Contrôleur graphique, Registre 05h



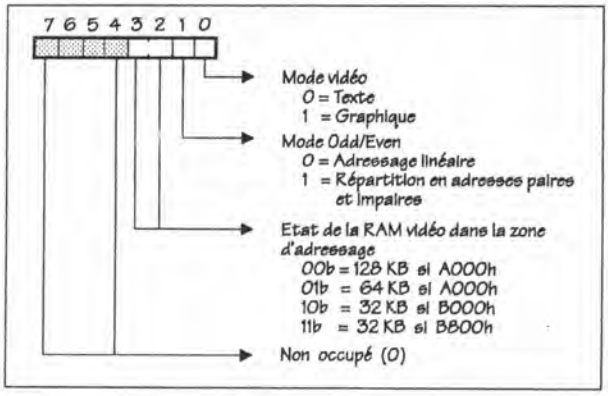
0+1 Conservent le numéro du mode Write utilisé lors des accès en écriture à la RAM vidéo quel que soit le mode vidéo actuel.

2 Conserve le numéro du mode vidéo actif.

3 Il faut communiquer au contrôleur graphique si un mode Odd/Even est actif ou si les plans de bits doivent être adressés linéairement. A cet effet, ce bit doit être chargé tel qu'il a été stocké dans le bit 2 du registre Memory Mode du séquenceur. Il faut en outre faire attention à la polarité inverse des deux bits.

5 Spécialement pour une carte VGA, ce bit conserve le paramètre indiquant si un mode 256 couleurs est actif ou non parce que le contrôleur graphique doit adapter en conséquence le mode de transfert des informations de couleur au contrôleur d'attributs.

Miscellaneous — Contrôleur graphique, Registre 06h

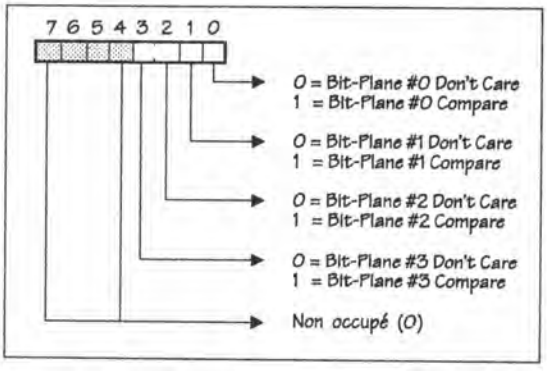


0 La valeur conservée ici permet de savoir si le mode actuellement actif est un mode texte ou graphique. Ainsi, le contrôleur graphique n'accède plus aux registres Latch servant à convertir les codes ASCII en modèles de points en mode texte.

1 A nouveau, ce bit indique si un mode Odd/Even est actuellement actif ou si les plans de bits peuvent être adressés linéairement.

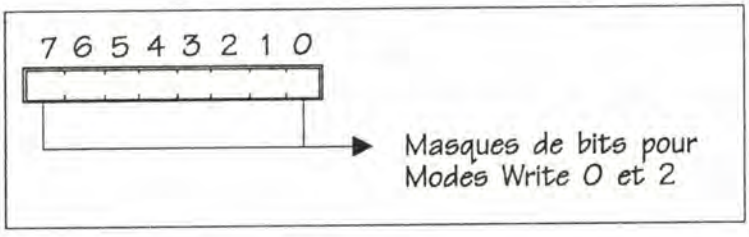
2+3 Ce champ de bits indique à quel endroit de la zone d'adresse du processeur la RAM vidéo doit-elle être intégrée. Ce champ de bits est également réglé par le BIOS lors de la commutation à un mode vidéo précis compte tenu du moniteur connecté.

Color Don't Care — Contrôleur graphique, Registre 07h



0-3 Ces bits influent sur le fonctionnement de la carte vidéo lors des accès en lecture à la RAM vidéo en mode Read 1. Ces bits décident notamment sur les plans de bits à inclure et à exclure de la comparaison des couleurs.

Bit Mask	Contrôleur graphique, Registre 08h



0-7 Lors d'un accès en écriture à la RAM vidéo en modes Write 0 et 2, les divers bits décident des bits à copier tels quels dans le plan de bits concerné parmi ceux des quatre registres Latch et ceux à combiner avec d'autres données (de l'unité centrale ou du registre Set/Reset) par une relation logique à travers le registre Function Select. Dans les quatre registres Latch, cela concerne tous les bits dont le bit correspondant est réglé dans ce registre.

Convertisseur digital en analogique (DAC)

Un DAC n'est disponible que sur une carte VGA. Il convertit les codes de couleur digitaux en signaux analogiques. Il représente la dernière phase de la conversion de couleur. Sur une carte VGA, il fait suite aux registres de palettes qui désignent l'un des 256 registres DAC ainsi que d'autres informations.

Un tel registre de couleur se compose de 18 bits différents dont chaque groupe de six bits reçoit le code de couleur pour l'une des trois couleurs fondamentales rouge, vert et bleu (RVB). Il résulte ainsi un total de 262 144 couleurs (256 Ko).

L'accès aux registres de couleur DAC s'effectue à travers les registres cités dans le tableau suivant. Contrairement aux autres contrôleurs d'une carte EGA ou VGA, ils ne sont pas adressés via un registre de données et d'index mais une adresse de port individuelle.

Port	Nom de registre
3C8h	Pel Write Address
3C7h	Pel Read Address
3C7h	DAC State

Port	Nom de registre
3C9h	Pel Data
3C6h	Pel Mask

Pour décrire un registre de couleur DAC, il faut avant tout inscrire son numéro dans l'intervalle 0-255 du registre Pel Write Address. Il faut ensuite transmettre la nouvelle couleur du registre adressé au registre Pel Data. La largeur de ce registre n'étant seulement que de 8 bits et comme il n'est pas en mesure de recevoir simultanément le code couleur 18 bits, il faut le communiquer par petites doses au registre Pel Data. Il convient donc d'envoyer d'abord les 6 bits composant le rouge au registre Pel Data puis les 6 bits composant le vert et ensuite les 6 bits composant le vert.

Sachant qu'en règle général on ne charge pas un seul registre DAC mais la palette intégrale des 256 registres ou du moins une grande partie, la valeur du registre Pel Write Address est augmentée automatiquement après la sortie des trois composants de couleur dans le registre Pel Data. Le résultat est que les trois composants de couleur sont directement chargés dans le registre Pel Data pour le registre DAC suivant sans renouveler l'accès au registre Pel Write Address.

Ce cycle ne se termine que si un nouveau numéro de registre est chargé dans le registre Pel Address Write ce qui met en route un nouveau processus de chargement. Un accès en lecture au registre Pel Address permet à tout moment de connaître le numéro du registre DAC qui est en train d'être traité.

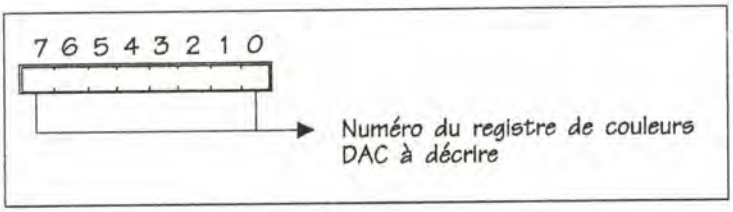
Les accès en écriture ainsi qu'en lecture aux registres de couleur DAC s'effectuent selon le schéma décrit. Il convient à cet effet de transmettre au préalable le numéro du premier registre à lire au registre Pel Read Address. Les proportions de rouge, vert et bleu dans le registre souhaité peuvent ensuite être lues successivement à travers le registre Pel Data. Mais il faut instaurer une pause entre les trois accès en lecture au registre Pel Data. Dans un programme assembleur, cette pause s'obtient à l'aide d'un saut vers la commande qui suit obligatoirement.

Ici aussi, le contenu du registre Pel Address augmente une fois la lecture achevée pour que le registre DAC suivant puisse être lu immédiatement.

L'accès à ces registres étant interdit au contrôleur d'attributs lors des accès en lecture/écriture aux registres DAC, il faut désactiver l'écran pendant ce laps de temps pour empêcher l'apparition de neige.

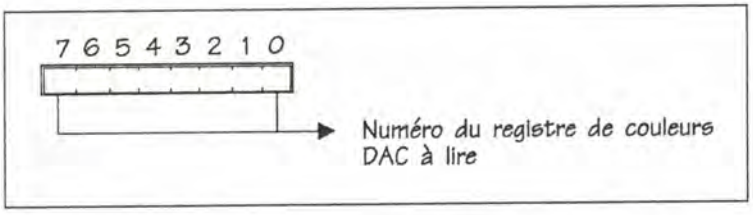
Voici maintenant la description des différents registres DAC.

Pel Write Address DAC, Port 3C8h, VGA uniquement



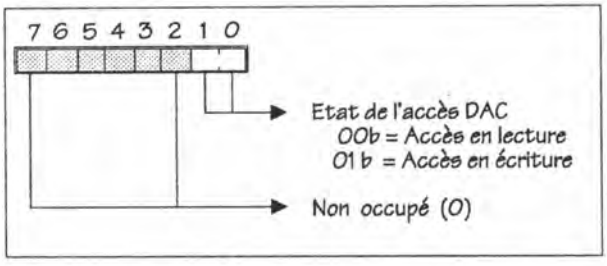
0-7 Ce registre conserve le numéro du registre DAC dans lequel il s'agit d'écrire à la suite d'un accès au registre Pel Data. Après chaque opération d'écriture, cette valeur augmente automatiquement pour qu'il soit possible d'écrire dans le registre DAC suivant.

Pel Read Address DAC, Port 3C7h, VGA uniquement



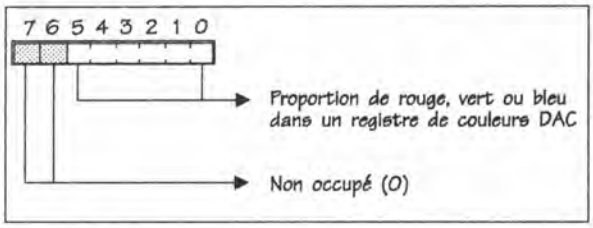
0-7 Le numéro du registre DAC dont le contenu doit être lu lors d'un accès en lecture ultérieur au registre Pel Data est stocké ici. Après chaque opération de lecture, cette valeur augmente automatiquement pour qu'il soit possible de lire le registre DAC suivant.

DAC State DAC, Port 3C7h, VGA uniquement



1+2 Indique si le registre DAC se trouve déjà en mode lecture ou écriture.

Pel Data	DAC, Port 3C9h, VGA uniquement



Lors d'une opération d'écriture dans l'un des registres DAC, il faut d'abord reporter la proportion de rouge, puis de vert et ensuite de bleu de la couleur dans ce registre. Seuls les six bits inférieurs parviennent effectivement dans le registre DAC.

Lors d'un accès en lecture aux registres DAC, ce registre fournit d'abord les proportions de rouge, puis de vert et ensuite de bleu de la couleur.

Pel Mask	DAC, Port 3C6h, VGA uniquement

0-7 Si le contrôleur d'attributs lit le contenu d'un registre DAC en générant une couleur pendant la configuration de l'écran, le contenu de ce registre est alors combiné avec le numéro du registre adressé par un ET logique. Pour les animations par exemple, cela permet de définir des groupes de couleurs sur le niveau inférieur de registres DAC déterminés.

Normalement, ce registre contient la valeur FFh pour que tous les accès aux registres DAC s'effectuent sans aucune modification.

4.9. Les cartes Super VGA

Avant que les fabricants de cartes ne parviennent à élucider le secret des nouveaux chips VLSI d'IBM et à les imiter, ces fabricants sont parvenus en quelques mois à tenir tête aux modèles compatibles VGA. Comme on a pu l'observer avec les cartes EGA, une

rude bataille s'engagea entre les fabricants pour s'octroyer une part maximale du marché tout en jouant sur les prix et en continuant d'innover les caractéristiques performantes de leurs cartes. De cette bataille surgit une nouvelle classe de cartes, à savoir les cartes Super VGA. Entièrement compatibles avec le standard VGA d'IBM, elles disposaient d'innombrables registres électroniques qui n'étaient même pas définis dans le standard IBM. Ces registres leur ont permis de surpasser le modèle original d'IBM grâce aux caractéristiques suivantes :

- ✓ Davantage de couleurs dans les modes graphiques VGA classiques,
- ✓ Modes graphiques jusque-là inconnus avec hautes résolutions,
- ✓ Modes texte avec plus de colonnes et lignes,
- ✓ Curseur électronique en mode graphique également,
- ✓ Zoom Hard de zones d'écran en mode graphique,

etc.

Mais les divers fabricants ont dû se résigner à établir un modèle standard pour cette nouvelle classe de cartes vidéo bien qu'ils se soient efforcés de régler les performances, les adapter électroniquement et assurer leur adressage à travers le BIOS en ROM. L'utilisateur qui conçoit actuellement un programme pour sa carte Super VGA doit être sûr de ne plus travailler en collaboration avec les toutes premières cartes Super VGA. A l'ère de Windows 3, cela ne porte toutefois pas préjudice puisque les fabricants de cartes doivent fournir un driver Windows à l'utilisateur pour la carte installée. Toujours est-il que les applications DOS supportent mal les cartes Super VGA parce qu'il faut développer des routines appropriées à chaque carte existante qui sont spécialement liées à l'électronique de cette carte.

Ce chapitre tente d'apporter une lumière dans l'obscurité du marché Super VGA sans décrire les différents aspects ni la programmation de chaque carte. A la fin de ce chapitre, nous allons en revanche présenter le standard VESA dans lequel les principaux fabricants de cartes Super VGA ont vu la lueur de leur politique de standardisation vouée à l'échec. Le standard VESA permet à vos programmes de fonctionner avec des cartes Super VGA de modèles différents sans être obligé de se concentrer sur un modèle précis.

Les principaux créateurs

Bien qu'il existe des centaines de constructeurs de cartes Super VGA de par le monde, les éléments fondamentaux sont pourtant le fruit d'une bonne douzaine de fabricants capables de créer des chips pour la fabrication de cartes Super VGA. Les principaux sont :

- ✓ ATI dont le VGAWONDER se rencontre dans de nombreux ordinateurs. Sa particularité est d'émuler dans le plus petit détail tous les modèles graphiques antérieurs (y compris Hercules).

- ✓ Chips & Technologies qui est une société reconnue surtout pour la fabrication des chips NEAT.
- ✓ Genoa dont la carte n'appartient pas tout simplement à la classe des cartes Super VGA mais porte carrément le nom SuperVGA.
- ✓ Headland mieux connue sous le nom VideoSeven.
- ✓ Tseng, incontestablement le numéro un du marché avec les chips VGA ET3000 et ET4000 qui se rencontrent dans la majorité des cartes VGA.
- ✓ Paradise régnant depuis longtemps sur l'empire de Western Digital, le constructeur de disques durs et mémoires.

Bien que les principaux fabricants de chips des cartes VGA et Super VGA soient au nombre de six, il ne faut pas en conclure que les différences entre les cartes Super VGA se réduisent à ces six modèles de chips. Bien au contraire, pratiquement chacun des chips cités offre une grande liberté de choix au constructeur de cartes Super VGA quant aux modes soutenus et leur adressage par le BIOS.

Modes texte Super VGA

Si on fait le tri parmi les performances offertes par les diverses cartes Super VGA, il en ressort un mode texte avec 25 lignes sur 132 colonnes qui est soutenu par la plupart de ces cartes. Il existe aussi de nombreux modes texte qui affichent certes 132 colonnes mais un nombre de lignes particulièrement élevé.

Les modes classiques sont toutefois les modes 80 colonnes où le nombre de lignes écran est augmenté ainsi que les modes 100 colonnes avec un nombre de lignes tout aussi important. Le top niveau de ce groupe est sans doute le mode texte en 50 lignes sur 160 colonnes. Mais pour ce qui est de savoir si les différents caractères sont nettement identifiables à l'écran est une autre affaire.

Voici une description des modes textes fournis par les fabricants de cartes mais cette liste ne prétend nullement être complète.

Modes textes étendus sur des cartes Super VGA

Colonnes	Lignes
80	30
80	34
80	43
80	60
100	37
100	43
100	60
100	75
132	25
132	28
132	30
132	43
132	44
132	50
160	50

En règle générale, tous ces modes peuvent être initialisés par la fonction 00h de l'interruption vidéo du BIOS. Mais les fabricants n'ont pas pu se mettre d'accord pour uniformiser les numéros de code si bien qu'ils proposent leur propre mixture. On ne peut même pas profiter du fait que dans ces modes la RAM vidéo soit configurée exactement comme dans les modes texte classiques. Cela est fort dommage car aurait pu utiliser ces modes sans gros effort en définissant tout simplement les routines existantes avec une largeur de lignes plus élevée dans la RAM vidéo.

Eu égard à la résolution de l'écran, si vous souhaitez malgré tout assurer la souplesse de vos programmes DOS en mode texte, nous vous conseillons d'avoir recours aux fonctions du BIOS VESA ou de collaborer avec l'utilisateur. Il existe notamment des programmes de configuration permettant de définir la résolution écran au niveau du DOS. Dans ce cas, il suffit de transmettre au programme la résolution écran spécifiée lors du démarrage du programme en guise de paramètre dans la ligne de commande ou de le préciser par exemple dans un fichier de configuration que le programme lit après le démarrage.

Modes graphiques Super VGA

L'éventail des nouveaux modes graphiques reconnus par les cartes Super VGA n'est pas de moindre taille. Il s'agit d'une part des modes graphiques classiques du simple standard VGA capables désormais d'afficher par exemple 256 couleurs au lieu des 16 actuelles. D'autre part, des modes graphiques inconnus viennent s'y ajouter offrant des résolutions nettement plus élevées atteignant 1024*768 points.

Malgré la jouissance procurée par ces modes et le nombre de couleurs accru, il ne faut pas oublier que l'augmentation de la résolution écran va de pair avec la vitesse. Il existe en effet une grande différence entre l'affichage de 300 000 points en mode 640*480 ou 500 000 points en mode 800*600. En fait de compte, c'est toujours l'unité centrale qui est la pièce maîtresse devant définir chaque point séparément et ne pouvant se consacrer actuellement à aucune autre tâche. Ce n'est pas sans raison que la plupart des utilisateurs de Windows 3 préfèrent le bon vieux mode standard VGA 640*480 au lieu d'un mode Super VGA au risque de perdre un temps énorme pour la configuration de l'écran.

Bon nombre de modes exotiques avec des résolutions de 720*396 ou 960*720 points tentent de se frayer un chemin parmi les trois maîtres absolus dans ce domaine reconnus par la plupart des cartes Super VGA, à savoir :

- ✓ le mode 640*400 points avec 256 couleurs,
- ✓ le mode 800*600 points avec 16 ou 256 couleurs et
- ✓ le mode 1024*768 points avec 16 couleurs.

Le tableau suivant montre que les modes cités représentent une petite portion parmi la multitude des modes graphiques offerts sous des variantes plus ou moins élaborées par les cartes Super VGA.

Modes graphiques étendus des cartes Super VGA			
Résolution	**Couleur**	**Points**	**Mémoire**
512*480	256	245 760	256 Ko
640*400	256	256 000	256 Ko
640*480	256	307 200	512 Ko
720*396	16	285 120	256 Ko
720*512	16	368 640	256 Ko
720*512	256	368 640	512 Ko
720*540	16	388 800	256 Ko
720*540	256	388 800	512 Ko
752*410	16	308 320	256 Ko

Modes graphiques étendus des cartes Super VGA			
Résolution	**Couleur**	**Points**	**Mémoire**
800*600	16	480 000	256 Ko
800*600	256	480 000	512 Ko
960*720	16	691 200	512 Ko
1024*768	16	786 432	512 Ko
1024*768	256	786 432	1 Mo

Quel que soit le mode graphique soutenant une carte Super VGA, ces modes ne peuvent être utilisés que si un moniteur adéquat et une RAM vidéo suffisante sont disponibles. Pour les modes avec une résolution de 640*480 points, un moniteur VGA à fréquence fixe est par exemple largement suffisant alors que les modes haute résolution nécessitent un moniteur multisync. Avec une résolution de 1024*768 points, on entre dans une zone technique réservée aux moniteurs XL. Il s'agit en fait de moniteurs multisync particuliers que l'on rencontre généralement dans le domaine CAO/PAO où des résolutions de 1024*768 points représentent tout simplement le bas de gamme.

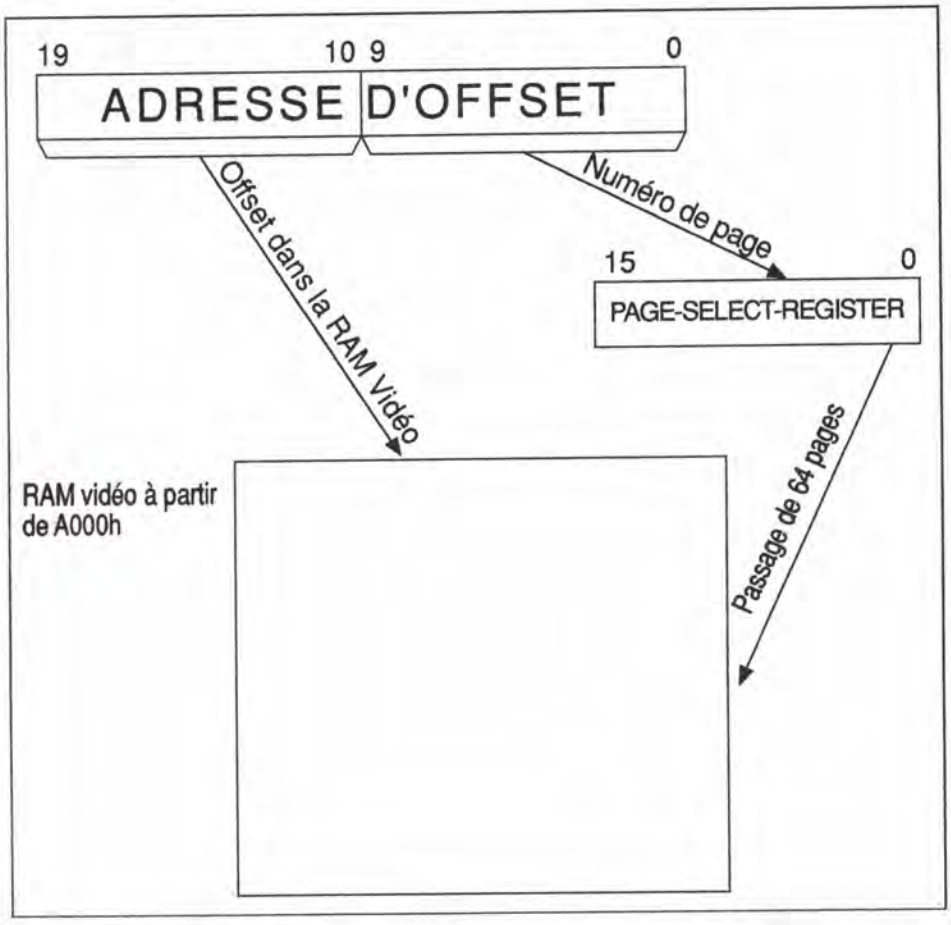
Organisation de la RAM vidéo dans les modes Super VGA

En observant les modes graphiques Super VGA, on aperçoit une différence entre les modes selon qu'ils affichent 16 ou 256 couleurs comme on a pu le constater avec les modes VGA classiques. Dans les deux cas, l'organisation de la RAM vidéo suit le même schéma que celui des modes VGA standard. Pour les modes 16 couleurs, cela signifie que 4 bits issus des quatre plans de bits consécutifs reçoivent la couleur d'un point. Dans les modes 256 couleurs, c'est un octet entier qui reçoit la couleur d'un point. Pour calculer son adresse, il suffit de multiplier le numéro de ligne par la longueur de ligne et d'additionner ensuite les numéros de colonnes. Seule la longueur des lignes dans la RAM vidéo et la longueur des pages changent par rapport aux modes VGA classiques en raison de l'accroissement de la résolution.

"Tout est pour le mieux" serait-on tenté de dire s'il n'y avait un petit hic. Les mécanismes classiques d'accès à la RAM vidéo ne fonctionnent plus avec les modes nécessitant plus de 256 Ko par page écran. En modes 16 couleurs, on peut adresser au maximum 256 Ko de la RAM vidéo à travers le segment de mémoire 64 Ko à partir de l'adresse A000:0000 avec l'aide des quatre plans de bits. La situation est pire en modes 256 couleurs où on ne dispose que de 64 Ko pour adresser les 256 Ko (cf. 4.8.7).

En tous cas, la limitation à 64 Ko à partir de A000:0000 représente une barrière insurmontable pour tous les modes graphiques dans lesquels il faut adresser plus de 256 Ko. Toutes les cartes Super VGA sont dotées d'un mécanisme permettant d'adresser la totalité de la RAM vidéo d'une carte Super VGA via le segment de mémoire à partir de A000. La contrainte est que l'adressage ne s'effectue pas simultanément mais

toujours par un bloc de 64 Ko en modes 256 couleurs ou 256 Ko en modes 16 couleurs. Toutes les cartes disposent à cet effet de divers registres électroniques inexistants sur des cartes VGA classiques. Ils servent à intégrer une portion déterminée de la RAM vidéo dans le segment de mémoire A000. Cette méthode est d'ailleurs similaire à celle utilisée pour gérer la mémoire paginée (cf. 12.1).



Intégration de parties déterminées de la RAM vidéo dans le segment de mémoire depuis A000:0000 en divisant l'adresse d'offset en une portion Page et une portion Offset

Cette technique ne facilite certainement pas les routines de définition ou lecture d'un point. Il faut non seulement calculer l'offset dans la RAM vidéo mais définir la fenêtre de mémoire de manière à atteindre l'octet souhaité dans la RAM vidéo. Il faut en outre faire attention au fait que la fenêtre de mémoire ne peut être décalée qu'avec une granularité précise représentant en général une puissance de deux de 1 Ko (1 Ko, 2 Ko, 4 Ko, 8 Ko, etc.). Dans ce contexte, on parle également de pages dans lesquelles la RAM vidéo est répartie. La taille d'une page correspond à la granularité. Avec une granularité de 1 Ko, le segment de mémoire A000 peut contenir par exemple 64 pages

consécutives. Le numéro de la première page est déterminé par un registre électronique baptisé Page Select.

Avec cette méthode, la cellule de mémoire visible dans le segment A000 ne peut pas commencer à n'importe quelle adresse d'offset mais uniquement à un multiple de la granularité. Par exemple, si on veut accéder à la cellule 65539 (c'est-à-dire 64 Ko + 3) avec une granularité de 1 Ko, on doit d'abord charger 64 dans le registre Page Select pour pouvoir accéder à l'octet souhaité dans la RAM vidéo à travers la cellule de mémoire A000:0003. Mais dans ce cas, il existe 63 combinaisons supplémentaires pour la valeur du registre Page Select et l'adresse d'offset pour l'accès au segment de mémoire A000. Ainsi, on peut également charger la valeur 63 dans le registre Page Select pour accéder à l'octet souhaité à travers l'adresse d'offset 1024+3 par rapport au segment de mémoire A000.

En raison de cette forme d'organisation, les routines graphiques sont obligées de calculer l'adresse d'offset dans la RAM vidéo dans un numéro de page et un offset pour avoir accès au segment de mémoire A000. Cette tâche n'est pas difficile si on se réfère à la représentation binaire des éléments nécessaires.

Le point de départ est l'adresse d'offset pour l'accès à la RAM vidéo. Sa largeur doit être de 20 bits si on souhaite adresser une RAM vidéo de 1 Mo. 19 bits suffisent s'il n'y a que 512 Ko mais nous allons partir de 20 bits. Avec une granularité de 1 Ko, on doit diviser cette adresse d'offset par 1024 pour obtenir la valeur du registre Page Select. Sur le plan binaire, "diviser" signifie décaler le dividende vers autant de positions vers la droite que nécessaires pour afficher le diviseur en binaire. Sachant qu'1 Ko correspond à la valeur 2^{10}, on doit décaler l'adresse d'offset 20 bits de 10 bits vers la droite pour obtenir le nombre souhaité. Les positions que l'on décale à partir de la virgule constituent l'offset dans le segment de mémoire A000. Cette répartition est illustrée dans la figure précédente.

Outre le calcul du numéro de page et l'adresse d'offset, les routines graphiques sont obligées d'assumer d'autres tâches liées au mécanisme du Paging. L'affaire se complique lorsqu'il s'agit de copier des zones distinctes de plus de 64 Ko dans la RAM vidéo. Dans ce cas, il n'est plus possible de visualiser simultanément les zones source et cible dans le segment de mémoire A000. Il faut alors subdiviser la procédure de copie en trois étapes interminables. Cela consiste à rendre visible la zone source dans le segment de mémoire A000 pour copier ensuite le programme dans un buffer représentant une partie de la mémoire principale de la RAM. C'est ensuite au tour de la zone cible d'apparaître dans le segment de mémoire A000 ce qui permet de copier le contenu du buffer dans la zone cible. Cette procédure ressemble à un échange de deux variables par une troisième.

De telles opérations étant généralement liées au contexte des animations, certaines cartes Super VGA divisent le segment de mémoire A000 en deux grandes parties de 32 Ko chacunes disposant de leur propre registre Page Select. Cela permet d'atteindre simultanément deux zones différentes de la RAM vidéo à travers le segment de mémoire

A000 et de copier directement deux zones à travers la RAM vidéo sans avoir à activer un buffer intermédiaire.

Mais à nouveau, il convient de prendre en compte les différences imposées par les fabricants de cartes Super VGA puisqu'il n'existe aucune uniformité quant à la procédure de sélection de la page. Toutes sortes de registres impliqués dans ce phénomène continuent de faire leur apparition en intégrant les pages dans le segment de mémoire A000 avec des granularités tout à fait variables. On ne peut pas toujours partir du principe qu'il existe deux zones paging différentes.

Si vous souhaitez adapter vos programmes graphiques aux modes Super VGA nécessitant plus de 256 Ko, vous aurez certainement fort à faire pour régler les routines. Cela explique pourquoi les fabricants de cartes fournissent des drivers logiciels avec des programmes comme AutoCAD ou Windows.

Le standard VESA apporte toutefois une lueur d'espoir puisqu'il met une interface logicielle non tributaire de l'électronique à la disposition des programmes en modes Super VGA. Reportez-vous à la section 4.9.1 à ce sujet.

Expériences personnelles

Si vous vous entêtez à programmer en modes Super VGA malgré tous les désagréments, nous vous conseillons le mode 800*600 points en 16 couleurs pour plusieurs raisons. Ce mode n'est pas seulement disponible sur la plupart des cartes Super VGA, mais nécessite seulement 256 Ko de RAM. Par conséquent, la programmation des registres Paging ne s'avère nullement indispensable et vous avez la certitude que toute carte Super VGA fera des miracles sur l'écran quelle que soit la capacité mémoire. En travaillant dans ce mode, vous pouvez accéder effectivement aux routines liées aux modes graphiques standard VGA pouvant afficher 16 couleurs.

Les deux programmes suivants en Pascal (V8060P.PAS) et en C (V8060C.C) illustrent clairement ce phénomène. Ils reposent essentiellement sur les routines des programmes de la section 4.8.6. L'initialisation du mode graphique 800*600 points ainsi que la définition des points graphiques s'effectuent à l'aide des routines Assembleur intégrées dans les modules V8060CA.ASM et V8060PA.ASM.

La tâche la plus ardue est confiée à la routine d'initialisation INIT800600 car elle doit découvrir le numéro de code du mode graphique 800*600 points compte tenu de la carte vidéo installée. A cet effet, elle utilise une procédure toute simple : elle teste tout simplement plusieurs numéros de code jusqu'à ce que le BIOS accepte un des numéros de code. Six numéros de code sont stockés dans une sorte de tableau à l'intérieur du programme Assembleur. Généralement, ils servent à désigner le mode graphique 800*600 points. Ce tableau porte le nom MODENR. Il est parcouru de haut en bas par la routine INIT800600.

Les numéros de code les plus largement diffusés et donc procurant le plus de vraisemblance occupent le début du tableau. Il s'agit des codes 6Ah, 58h et 29h. Les numéros de code 54h, 16h et 79h sont moins probables mais sont suffisamment représentés pour être cités dans ce tableau. Lors de nos tests, ces six numéros de code sont sortis avec divers types de cartes Super VGA et concordent certainement avec plus de 80 % des cartes Super VGA vendues sur le marché. Si votre carte Super VGA utilise un autre code pour le mode graphique 800*600 points et si vous connaissez ce code, vous pouvez naturellement l'inclure dans le tableau MODENR.

Après avoir initialisé un mode du tableau à l'aide de la fonction 00h du BIOS vidéo, la routine INIT800600 reconnaît le numéro de code qui correspond au mode graphique 800*600 points. Puis elle appelle la fonction 0Fh du BIOS vidéo qui retourne le numéro de code du mode vidéo actuel. Si le numéro de code spécifié était inconnu lors de l'appel de la fonction 00h, ce mode ne peut évidemment pas être installé par le BIOS. La routine retourne alors un autre numéro de code reproduisant le mode actuel comme auparavant.

Tant que le numéro de code préalablement spécifié n'est pas retourné par la fonction 0Fh, la routine INIT800600 continue à parcourir le tableau code par code jusqu'à ce que le BIOS accepte un numéro de code ou la fin du tableau est atteinte. Dans ce cas, cette routine retourne la valeur 0 dans la version C et FALSE dans la version Pascal. Mais si le mode peut être défini, la valeur retournée en C est 1 et TRUE en Pascal.

Grâce à cette méthode, nous avons pu lire le mode adéquat en testant diverses cartes Super VGA. Mais signalons quand même un petit défaut. L'opération ne fonctionne pas lorsqu'on la manoeuvre avec une carte Super VGA qui utilise un des codes du tableau MODENR avec un mode graphique différent du mode 800*600 points en 16 couleurs. Dans ce cas, INIT800600 retourne TRUE mais le reste du programme ne fonctionne pas correctement parce qu'on se trouve dans un mode graphique complètement différent de celui qui était attendu. Quant à nous, nous n'avons pas rencontré ce problème lors de nos tests.

Listing : V8060C.C

```
/*****************************************************************
*                        V 8 0 6 0 C . C                         *
*----------------------------------------------------------------*
* Fonction          : Montre comment exploiter le mode 800*600 16 *
*                     couleurs de la carte Super VGA. Ce programme *
*                     accède aux routines en assembleur du module  *
*                     V8060CA.ASM                                  *
*----------------------------------------------------------------*
* Auteur        : MICHAEL TISCHER                                  *
* Développé le  : 14.01.1991                                       *
* Dernière MAJ  : 14.02.1992                                       *
*----------------------------------------------------------------*
* Modèle mémoire    : SMALL                                        *
*----------------------------------------------------------------*
* (MICROSOFT C)                                                    *
* Compilation       : CL /AS v8060c.c v8060ca                      *
*----------------------------------------------------------------*
* (BORLAND TURBO C)                                                *
* Compilation       : Utiliser un projet avec le contenu suivant   *
*                       v8060c.c                                   *
*                       v8060ca.obj                                *
*----------------------------------------------------------------*
*  Appel            : v8060c                                      *
*****************************************************************/

#include <dos.h>
#include <stdarg.h>
#include <stdlib.h>
#include <io.h>
#include <stdio.h>
#include <conio.h>

/*-- Déclarations de types ---------------------------------------*/

typedef unsigned char BYTE;

/*-- Références externes aux routines en assembleur -------------*/

extern int  init800600( void );
extern void setpix( int x, int y, unsigned char couleur);
extern BYTE getpix( int x, int y );
extern void far * getfontptr( void );
```

```
/*-- Déclarations dépendantes du compilateur ----------------------*/

#ifndef __TURBOC__
  #define random(x) ( rand() % (x+1) )          /* Fonction aléatoire */
#endif

/*-- Constantes ---------------------------------------------------*/

#define MAXX        799                         /* Coordonnées maximales */
#define MAXY        599
#define NBLINE      2500                        /* Nombre de lignes */
#define XDISTANCE   40      /* Distance entre le rectangle et le bord */
#define YDISTANCE   30
#define X1          ( 2 * XDISTANCE )       /* Coordonnées du rectangle */
#define Y1          ( 2 * YDISTANCE )
#define X2          ( MAXX-XDISTANCE )
#define Y2          ( MAXY-YDISTANCE )
#define XRAND       random( X2 - X1 - 1 ) + X1 + 1      /* Coordonnées */
#define YRAND       random( Y2 - Y1 - 1 ) + Y1 + 1       /* aléatoires */

/*********************************************************************
*  IsVga: Teste la présence d'une carte VGA.                         *
**-------------------------------------------------------------------**
*  Entrée : néant                                                    *
*  Sortie : 0 si pas de carte VGA, sinon -1                          *
*********************************************************************/

BYTE IsVga( void )
{
 union REGS  Regs;              /* Registres pour gérer l'interruption */

 Regs.x.ax = 0x1a00;            /* La fonction 1AH n'existe qu'en VGA */
 int86( 0x10, &Regs, &Regs );
 return ( Regs.h.al == 0x1a );              /* Est-elle disponible ? */
}
/*********************************************************************
*  PrintChar : Ecrit un caractère en dehors de la zone visible       *
*              de la mémoire d'écran                                 *
**-------------------------------------------------------------------**
*  Entrée  :   caractere = caractère à afficher                      *
*              x, y      = Coordonnées du coin supérieur gauche      *
*              cc        = Couleur du caractère                      *
*              cf        = Couleur du fond                           *
*  Info    : Le caractère est dessiné dans une matrice de 8*8 pixels *
*            sur la base du jeu de caractères 8*8 en ROM             *
*********************************************************************/

void PrintChar( char caractere, int x, int y, BYTE cc, BYTE cf )
{
 typedef BYTE   CARDEF[256][8];     /* Structure du jeu de caractères */
 typedef CARDEF far *CARPTR;        /* Pointe sur un jeu de caractères */

 BYTE     i, k,                             /* Compteur d'itérations */
          masque;       /* Masque binaire pour dessiner le caractère */

 static   CARPTR fptr = (CARPTR) 0;      /* Jeu de caractères en ROM */

 if( fptr == (CARPTR) 0 )       /* A-t-on déjà déterminé ce pointeur ? */
   fptr = getfontptr();       /* Non, détermine avec fonct. assembleur */

  /*- Dessine le caractère pixel par pixel ---------------------------*/

 if( cf == 255 )                            /* Caractère transparent ? */
   for( i = 0; i < 8; ++i )     /* Oui, dessine que pixels du 1er plan */
   {
     masque = (*fptr)[caractere][i];        /* Motif bin. pour ligne */
     for( k = 0; k < 8; ++k, masque <<= 1 )  /* Parcourt les colonnes */
       if( masque & 128 )                     /* Pixel à dessiner ? */
         setpix( x+k, y+i, cc );                          /* Oui */
   }
 else                                       /* Non dessine chaque pixel */
   for ( i = 0; i < 8; ++i )                    /* Parcourt les lignes */
   {
     masque = (*fptr)[caractere][i];        /* Motif bin. pour ligne */
     for( k = 0; k < 8; ++k, masque <<= 1 )  /* Parcourt les colonnes */
       setpix( x+k, y+i, (BYTE) (( masque & 128 ) ? cc : cf) );
   }
}

/*********************************************************************
*  Line: Trace un segment dans la fenêtre graphique en appliquant    *
*        l'algorithme de Bresenham                                   *
**-------------------------------------------------------------------**
*  Entrées : X1, Y1 = Coordonnées de l'origine                       *
*            X2, Y2 = Coordonnées de l'extrémité terminale           *
*            COULEUR = couleur du segment                            *
*********************************************************************/

/*-- Fonction accessoire pour échanger deux variables entières -------*/

void SwapInt( int *i1, int *i2 )
{
 int dummy;

 dummy = *i2;  *i2  = *i1;  *i1  = dummy;
}

/*-- Procédure principale -----------------------------------------*/

void Line( int x1, int y1, int x2, int y2, BYTE couleur )
{
 int d, dx, dy,
     aincr, bincr,
     xincr, yincr,
     x, y;

 if( abs(x2-x1) < abs(y2-y1) )      /* Sens du parcours : axe X ou Y ? */
 {                                                          /* Par Y */
   if( y1 > y2 )                          /* y1 plus grand que y2? */
   {
     SwapInt( &x1, &x2 );                 /* Oui échange X1 et X2, */
     SwapInt( &y1, &y2 );                                /* Y1 et Y2 */
   }

   xincr = ( x2 > x1 ) ? 1 : -1;          /* Fixe le pas horizontal */

   dy    = y2 - y1;
   dx    = abs( x2-x1 );
   d     = 2 * dx - dy;
   aincr = 2 * (dx - dy);
   bincr = 2 * dx;
   x     = x1;
   y     = y1;

   setpix( x, y, couleur );             /* dessine le premier pixel */
   for(y=y1+1; y<= y2; ++y )              /* Parcourt l'axe des Y */
   {
     if( d >= 0 )
     {
       x += xincr;
       d += aincr;
     }
     else
       d += bincr;
     setpix( x, y, couleur );
   }
 }
 else                                                       /* par X */
 {
   if( x1 > x2 )                          /* x1 plus grand que x2? */
   {
     SwapInt( &x1, &x2 );                 /* Oui, échange X1 et X2 */
     SwapInt( &y1, &y2 );                                /* Y1 et Y2 */
   }

   yincr = ( y2 > y1 ) ? 1 : -1;           /* Fixe le pas vertical */

   dx    = x2 - x1;
   dy    = abs( y2-y1 );
   d     = 2 * dy - dx;
   aincr = 2 * (dy - dx);
   bincr = 2 * dy;
   x     = x1;
   y     = y1;

   setpix( x, y, couleur );             /* Dessine le premier pixel */
   for( x=x1+1; x<=x2; ++x )              /* Parcourt l'axe des X */
   {
     if( d >= 0 )
     {
       y += yincr;
       d += aincr;
     }
     else
       d += bincr;
     setpix( x, y, couleur );
   }
 }
}

/*********************************************************************
*  GrafPrintf: Affiche une chaine formatée sur l'écran graphique     *
**-------------------------------------------------------------------**
*  Entrées : X, Y   = Coordonnées de départ (0 - ...)                *
*            CC     = Couleur des caractères                         *
*            CF     = Couleur du fond  (255 = transparent)           *
*            STRING = Chaîne avec indications de formatage           *
*            ...    = Expressions comme pour printf                  *
*********************************************************************/

void GrafPrintf( int x, int y, BYTE cc, BYTE cf, char * string, ... )
{
 va_list  parameter;     /* Liste de paramètres pour les macros VA_...*/
 char     affichage[255],          /* Buffer pour la chaîne formatée */
          *cp;
```

```
va_start( parameter, string );          /* Convertit les paramètres */
vsprintf( affichage, string, parameter );              /* Formate */
for( cp = affichage; *cp; ++cp, x+= 8 )        /* Affiche la chaîne */
  PrintChar( *cp, x, y, cc, cf );          /* formatée par PrintChar */
}

/*****************************************************************
*  DrawAxis: Draws axes from the left and top borders on the screen.  *
**---------------------------------------------------------------**
*  Input   : STEPX = Increment for X-axis                         *
*            STEPY = Increment for Y-axis                         *
*            FG    = Foreground color                             *
*            BK    = Background color (255 = transparent)         *
*****************************************************************/

void DrawAxis( int stepx, int stepy, BYTE fg, BYTE bk )
{
 int  x, y;                                    /* boucle coordonnées */

 Line( 0, 0, MAXX, 0, fg );                        /* Dessine axe X */
 Line( 0, 0, 0, MAXY, fg );                        /* Dessine axe Y */

 for( x = stepx; x < MAXX; x += stepx )            /* Echelle axe X */
 {
   Line( x, 0, x, 5, fg );
   GrafPrintf( x < 100 ? x - 8 : x - 12, 8, fg, bk, "%d", x );
 }

 for( y = stepy; y < MAXY; y += stepy )            /* Echelle axe Y */
 {
   Line( 0, y, 5, y, fg );
   GrafPrintf( 8, y-4, fg, bk, "%3d", y );
 }
}


/*****************************************************************
*  Demo: Démontre l'usage des différentes fonctions de ce module  *
*****************************************************************/

void Demo( void )
{
 int  i;                                   /* Compteur d'itérations */

 DrawAxis( 30, 20, 15, 255 );                    /* Dessine des axes */

 GrafPrintf( X1, MAXY-10, 15, 255,
             "V8060C.C  -  (c) by MICHAEL TISCHER" );

 Line( X1, Y1, X1, Y2, 15 );                     /* Encadre le rectangle */
 Line( X1, Y2, X2, Y2, 15 );
 Line( X2, Y2, X2, Y1, 15 );
 Line( X2, Y1, X1, Y1, 15 );

   /*-- Dessine des segments aléatoires dans le rectangle ----------*/

 for( i = NBLINE; i > 0 ; --i )
   Line( XRAND, YRAND, XRAND, YRAND, (BYTE) (i % 16) );
}

/*-------------------------------------------------------------------*/
/*--                    PROGRAMME PRINCIPAL                        --*/
/*-------------------------------------------------------------------*/

void main( void )
{
 union REGS  regs;

 printf( "V8060C.C  - (c) 1991, 92 by MICHAEL TISCHER\n\n" );
 if( IsVga() )                          /* Dispose-t-on d'une carte VGA ? */
 {                          /* Oui, mais le mode 800*600 est-il accessible ? */
   if( Init800600() )
   {                                                    /* Ça marche ! */
     Demo();                                      /* Exécute la démo */
     getch();                               /* Attend une frappe de touche */
     regs.x.ax = 0x0003;                      /* Rétablit le mode texte */
     int86( 0x10, &regs, &regs );
   }
   else
   {
     printf( "Attention! Le mode 800*600 n'a pas pu être initialisé\n");
     exit( 1 );
   }
 }
 else
 {                                                  /* Non, pas de carte VGA */
   printf( "Attention! Ce programme nécessite une carte VGA !");
   exit(1);                                         /* Termine le programme */
 }
}
```

Listing : V8060CA.ASM

```
;*****************************************************************;
;*                      V 8 0 6 0 C A . A S M                    *;
;*---------------------------------------------------------------*;
;*   Fonction        : contient diverses routines pour travailler *;
;*                     dans le mode 800*600 des cartes Super-VGA- *;
;*                     avec 16 couleurs                           *;
;*---------------------------------------------------------------*;
;*   Auteur          : MICHAEL TISCHER                            *;
;*   Développé le    : 14.01.1991                                 *;
;*   Dernière MAJ    : 14.02.1992                                 *;
;*---------------------------------------------------------------*;
;*   Modèle mémoire  : SMALL                                      *;
;*---------------------------------------------------------------*;
;*   Assemblage      : MASM /mx V8060CA      ou   TASM -mx V8060CA*;
;*****************************************************************;

IGROUP group _text                  ;Regroupe les segments de programme
DGROUP group _bss, _data            ;Regroupe les segments de données
       assume CS:IGROUP, DS:DGROUP, ES:DGROUP, SS:DGROUP

_BSS   segment word public 'BSS'        ;Segment des variables statiques
_BSS   ends                                     ; non initialisées

_DATA  segment word public 'DATA'       ;Segment des variables globales
                                          ; et statiques initialisées


_DATA  ends

;=== Constantes ================================================

GC_INDEX      = 3ceh          ;Registre d'index du contrôleur graphique
GC_READ_MAP   = 4                       ;Numéro du registre Read Map
GC_BIT_MASK   = 8             ;Numéro du registre de masquage binaire
GC_GRAPH_MODE = 5               ;Numéro du registre de mode graphique

;=== Données ===================================================

_DATA  segment word public 'DATA'

       ;-- Numéro de code associé au mode 800*600 de différentes ------
       ;-- cartes Super-VGA --------------------------------------------

modenr     db 6Ah, 58h, 29h, 54h, 16h, 79h
modenrend  equ this byte

_DATA  ends

;=== Programme =================================================

_TEXT  segment byte public 'CODE'                  ;Segment de programme

;-- Déclarations publiques ---------------------------------------

public   _init800600                        ;Initialise le mode 800*600
public   _setpix                                    ;Dessine un pixel
public   _getpix                          ;Lit la couleur d'un pixel
public   _getfontptr  ;Renvoie un pointeur sur le jeu de caractères 8*8

;-----------------------------------------------------------------
;-- INIT800600: Initialise le mode 800*600 Super-VGA en 16 couleurs

;-- Déclaration : int init800600( void );
;-- Valeur de retour : 1 = mode initialisé , 0 = Erreur

_init800600 proc near

       ;-- Essaye les modes du tableau MODENR jusqu'à ce que l'un d'eux-
       ;--   soit accepté par le BIOS ------------------------------

       mov  si,offset modenr    ;Commence par le premier mode du tableau
it1:   xor  ah,ah                        ;Fonction 00h: Initialise le mode
       mov  al,[si]                  ;Charge le numéro de code du tableau
       int  10h                                        ;Initialise le mode
```

```
        mov  ah,0fh                         ;Fonction 0Fh: teste le mode
        int  10h
        cmp  al,[si]                        ;Le mode s'est-il déclenché ?
        je   it2                                        ;Oui --> OK

        ;-- Mauvais numéro de code, choisit le suivant dans le tableau --

        inc  si                                     ;SI sur le code suivant
        cmp  si,offset modenrend            ;A-t-on parcouru tt le tableau ?
        jne  it1                                        ;Non, on recommence

        xor  ax,ax                  ;Oui, clôture la fonction par une erreur
        ret                                             ;Retourne à l'appelant

it2:    ;-- Le mode a pu être initialisé ------------------------------

        mov  ax,1                                           ;tout est bon
        ret                                         ;Retour à l'appelant

_init800600 endp                                    ;Fin de la procédure

;-- SETPIX: Dessine un pixel dans une couleur donnée --------------
;-- Déclaration : void setpix( int x, int y, unsigned char couleur );

_setpix    proc near

sframe    struc                             ;Structure d'accès à la pile
bp0       dw ?                                          ;Mémorise BP
ret_adr0  dw ?                          ;Adresse de retour à l'appelant
x0        dw ?                                          ;Abscisse X
y0        dw ?                                          ;Ordonnée Y
couleur   dw ?                                            ;Couleur
sframe    ends                                  ;Fin de la structure

frame  equ [ bp - bp0 ]             ;adresse les éléments de la structure

        push bp                           ;Prépare l'adressage des paramètres
        mov  bp,sp                                  ; par le registre BP

        ;-- Calcule l'offset dans la mémoire d'écran et le décalage -----

        mov  ax,frame.y0                            ;Charge l'ordonnée  Y
        mov  dx,800/8                 ;Multiplie par la largeur d'une ligne
        mul  dx
        mov  bx,frame.x0                            ;Charge l'abscisse X
        mov  cl,bl                    ;Mémorise l'octet faible pour le calcul
                                                    ; du décalage

        shr  bx,1                           ;Divise l'abscisse X par huit
        shr  bx,1
        shr  bx,1
        add  bx,ax              ;Y ajoute l'offset issu de la multiplication

        and  cl,7                   ;Calcule le masque binaire à partir de X
        xor  cl,7
        mov  ah,1
        shl  ah,cl

        mov  dx,GC_INDEX                    ;Accède au contrôleur graphique
        mov  al,GC_BIT_MASK               ;Masque bin. dans registre masquage
        out  dx,ax                                  ; Effectue la sortie

        mov  ax,(02h shl 8) + GC_GRAPH_MODE ;Déclenche le mode Write 2 &
        out  dx,ax                                              ;Read 0

        mov  ax,0A000h   ;Charge en RES le segment de la mémoire d'écran
        mov  es,ax

        mov  al,es:[bx]                         ;Charge le registre latch
        mov  al,byte ptr frame.couleur                    ;Fixe la couleur
        mov  es:[bx],al                      ;Réécrit le registre latch

        ;-- Remet les valeurs par défaut dans les différents registres --
        ;-- du contrôleur graphique , qui ont été modifiés --------------

        mov  ax,(0FFh shl 8 ) + GC_BIT_MASK
        out  dx,ax

        mov  ax,(00h shl 8) + GC_GRAPH_MODE
        out  dx,ax

        pop  bp
        ret                                             ;Retourne à l'appelant

_setpix    endp                                     ;Fin de la procédure

;-- GETPIX: Détermine la couleur d'un pixel -----------------------------
;-- Déclaration : unsigned char getpix( int x, int y );

_getpix    proc near

sframe1   struc                             ;Structure d'accès à la pile
bp1       dw ?                                          ;Mémorise BP
ret_adr1  dw ?                          ;Adresse de retour à l'appelant
x1        dw ?                                          ;Abscisse X
y1        dw ?                                          ;Ordonnée Y
sframe1   ends                                  ;Fin de la structure

frame  equ [ bp - bp1 ]             ;adresse les éléments de la structure

        push bp                           ;Prépare l'adressage des paramètres
        mov  bp,sp                                  ; par le registre BP

        push si

        ;-- Calcule d'abord l'offset en mémoire d'écran et le décalage --

        mov  ax,frame.y1                            ;Charge l'ordonnée Y
        mov  dx,800 / 8               ;Multiplie par la largeur d'une ligne
        mul  dx
        mov  si,frame.x1                            ;Charge l'abscisse X
        mov  cx,si                          ;La mémorise pour le décalage

        shr  si,1                           ;Divise l'abscisse par huit
        shr  si,1
        shr  si,1
        add  si,ax              ;Y ajoute l'offset issu de la multiplication

        and  cl,7                   ;Calcule le masque binaire à partir de X
        xor  cl,7
        mov  ch,1
        shl  ch,cl

        mov  ax,0A000h                      ;Charge en ES le segment
        mov  es,ax                           ; de la mémoire d'écran

        mov  dx,GC_INDEX                ;Accède au contrôleur graphique
        mov  ax,(3 shl 8)+ GC_READ_MAP                  ;Lit d'abord
        xor  bl,bl                                      ; le plan 3

gp1:    out  dx,ax                          ;Indique le plan de bit à lire
        mov  bh,es:[si]                 ;Charge la valeur du registre latch
        and  bh,ch                          ;Ne garde que le pixel souhaité
        neg  bh                           ;Fixe le bit 7 en fonction du pixel
        rol  bx,1                   ;Rotation bit 7 de BH vers bit 1 dans BL.

        dec  ah                             ;Traite le plan de bits suivant
        jge  gp1                        ;sup ou égal à 0 ? ---> on continue

        mov  al,bl                          ;Résultat de la fonction en AL

        pop  si
        pop  bp
        ret                                             ;Retourne à l'appelant

_getpix    endp                                     ;Fin de la procédure

;-- GETFONTPTR: Renvoie un pointeur FAR sur le jeu de caractères 8*8 ---
;-- Déclaration : void far * getfontptr( void )

_getfontptr proc near

        push bp                                                 ;Empile BP

        mov  ax,1130h               ;Charge les registres pour l'interruption
        mov  bh,3
        int  10h                        ;Déclenche l'interruption vidéo du BIOS

        mov  dx,es                      ;Retourne le pointeur ES:BP en DX:AX
        mov  ax,bp

        pop  bp                                         ;Reprend BP sur la pile
        ret                                              ;Retourne à l'appelant

_getfontptr endp                                    ;Fin de la procédure

;-- Fin ----------------------------------------------------------------

_text   ends                                    ;Fin du segment de programme
        end                                 ;Fin de la source en assembleur
```

Listing : V8060P.PAS

```
{*********************************************************************}
{*                          V8060P.PAS                               *}
{**-----------------------------------------------------------------**}
{* Fonction : Montre comment exploiter le mode 800*600 16 couleurs de *}
{*            la carte Super VGA.                                    *}
{*            Ce programme utilise les routines en assembleur du     *}
{*            module V8060PA.ASM                                     *}
{**-----------------------------------------------------------------**}
{* Auteur       : MICHAEL TISCHER                                    *}
{* Développé le : 14.01.1991                                         *}
{* Dernière MAJ : 14.01.1991                                         *}
{*********************************************************************}

program V8060P;

uses dos, crt;

{-- Déclarations de types ------------------------------------------}

type BPTR = ^byte;

{-- Références externes aux routines en assembleur ----------------}

{$L v8060pa}                          { Intègre le module en assembleur }

function  initB00600 : boolean; external;
procedure setpix( x, y : integer; couleur : byte ); external;
function  getpix( x, y: integer ) : byte ; external;

{-- Constantes ----------------------------------------------------}

const MAXX     = 799;                              { Coordonnées maximales }
      MAXY     = 599;
      NBLINE   = 2500;                             { Nombre de lignes }
      XDISTANCE = 40;          { Distance entre le rectangle et le bord }
      YDISTANCE = 30;
      X1       = ( 2 * XDISTANCE );          { Coordonnées du rectangle }
      Y1       = ( 2 * YDISTANCE );
      X2       = ( MAXX-XDISTANCE );
      Y2       = ( MAXY-YDISTANCE );

{*********************************************************************}
{* IsVga : Teste la présence d'une carte VGA.                        *}
{**-----------------------------------------------------------------**}
{* Entrée  : néant                                                   *}
{* Sortie  : TRUE ou FALSE                                           *}
{*********************************************************************}

function IsVga : boolean;

var Regs : Registers;                    { Registres pour l'interruption}

begin
  Regs.AX := $1a00;                 { La fonction 1Ah n'existe qu'en VGA }
  Intr( $10, Regs );
  IsVga := ( Regs.AL = $1a );
end;

{*********************************************************************}
{* PrintChar : Affiche un caractère en mode graphique                *}
{**-----------------------------------------------------------------**}
{* Entrée : caractere = le caractère à afficher                      *}
{*          x, y      = Coordonnées du coin sup gauche               *}
{*          cc        = Couleur du caractère                         *}
{*          cf        = Couleur du fond                              *}
{* Info   : Le caractère est dessiné dans une matrice de 8*8 pixels  *}
{*          sur la base du jeu de caractères 8*8 en ROM              *}
{*********************************************************************}

procedure PrintChar( caractere : char; x, y : integer; cc, cf : byte );

type CARADEF = array[0..255,0..7] of byte; {Structure jeu de caractères}
     CARAPTR = ^CARADEF;              { Pointe sur le jeu de caractères }

var  Regs  : Registers;       { Registres pour gérer les interruptions }
     ch    : char;                                  { Pixel du caractère }
     i, k,                                      { Compteur d'itérations }
     Masque : byte;          { Masque binaire pour dessiner le caractère}

const fptr : CARAPTR = NIL;   { Pointe sur le jeu de caractères en ROM }

begin
  if fptr = NIL then              { A-t-on déjà déterminé ce pointeur ? }
    begin                                                        { Non}
      Regs.AH := $11;                    { Appelle l'option $1130 de la }
      Regs.AL := $30;                          { fonction vidéo du BIOS }
      Regs.BH := 3;           { pour obtenir un pointeur sur le jeu 8*8 }
      intr( $10, Regs );
      fptr := ptr( Regs.ES, Regs.BP );           { Compose le pointeur }
    end;

  if ( cf = 255 ) then                          { Caractère transparent ? }
    for i := 0 to 7 do { Oui ne dessine que les pixels du premier plan }
      begin
        Masque := fptr^[ord(caractere),i];{Motif binaire pour une ligne}
        for k := 0 to 7 do
          begin
            if ( Masque and 128 <> 0 ) then       { Pixel à dessiner ? }
              setpix( x+k, y+i, cc );                          { Oui }
            Masque := Masque shl 1;
          end;
      end
  else                                        { Non, tient compte du fond }
    for i := 0 to 7 do                              { Parcourt les lignes }
      begin
        Masque := fptr^[ord(caractere),i];{Motif binaire pour une ligne}
        for k := 0 to 7 do
          begin
            if ( Masque and 128 <> 0 ) then          { Premier plan ? }
              setpix( x+k, y+i, cc )                             { Oui }
            else
              setpix( x+k, y+i, cf );                        { Non, fond}
            Masque := Masque shl 1;
          end;
      end;
end;

{*********************************************************************}
{* Line : Trace un segment dans la fenêtre graphique en appliquant   *}
{*        l'algorithme de Bresenham                                  *}
{**-----------------------------------------------------------------**}
{* Entrée : X1, Y1  = Coordonnées de l'origine (0 - ...)             *}
{*          X2, Y2  = Coordonnées de l'extrémité terminale           *}
{*          COULEUR = couleur du segment                             *}
{*********************************************************************}

procedure Line( x1, y1, x2, y2 : integer; couleur : byte );

var d, dx, dy,
    aincr, bincr,
    xincr, yincr,
    x, y                  : integer;

{-- Procédure accessoire pour échanger deux variables entières --------}

procedure SwapInt( var i1, i2: integer );

var dummy : integer;

begin
  dummy := i2;
  i2    := i1;
  i1    := dummy;
end;

{-- Procédure principale ------------------------------------------}

begin
  if ( abs(x2-x1) < abs(y2-y1) ) then  { Parcours : par l'axe X ou Y ? }
    begin                                                 { par l'axe des Y }
      if ( y1 > y2 ) then                          { y1 supérieur à y2 ? }
        begin
      SwapInt( x1, x2 );                          { Oui, échange X1 et X2, }
      SwapInt( y1, y2 );                                    { Y1 et Y2 }
        end;

      if ( x2 > x1 ) then xincr := 1          { Fixe le pas horizontal }
                     else xincr := -1;

      dy := y2 - y1;
      dx := abs( x2-x1 );
      d  := 2 * dx - dy;
      aincr := 2 * (dx - dy);
      bincr := 2 * dx;
      x := x1;
      y := y1;

      Setpix( x, y, couleur );                { Dessine le premier point }
      for y:=y1+1 to y2 do                          { Parcourt l'axe des Y }
        begin
          if ( d >= 0 ) then
            begin
              inc( x, xincr );
              inc( d, aincr );
```

```
          end
        else
          inc( d, bincr );
        Setpix( x, y, couleur );
      end;
  end
else                                                    ( par l'axe des X )
  begin
    if ( x1 > x2 ) then                                 ( x1 plus grand que x2 ? )
      begin
    SwapInt( x1, x2 );                                  ( Oui, échange X1 et X2 )
    SwapInt( y1, y2 );                                          ( Y1 et Y2 )
      end;

    if ( y2 > y1 ) then yincr := 1                      ( Fixe le pas vertical )
                   else yincr := -1;

    dx := x2 - x1;
    dy := abs( y2-y1 );
    d  := 2 * dy - dx;
    aincr := 2 * (dy - dx);
    bincr := 2 * dy;
    x := x1;
    y := y1;

    Setpix( x, y, couleur );                            ( Dessine le premier point )
    for x:=x1+1 to x2 do                                  ( Parcourt l'axe des X )
      begin
        if ( d >= 0 ) then
          begin
            inc( y, yincr );
            inc( d, aincr );
          end
        else
          inc( d, bincr );
        Setpix( x, y, couleur );
      end;
  end;
end;

{**********************************************************************
*  GrafPrint: Affiche une chaîne formatée sur l'écran graphique       *
***********************************************************************
*  Entrées : X, Y   = Coordonnées de départ (0-...)                   *
*            CC     = Couleur des caractères                          *
*            CF     = Couleur du fond (255 = transparent)             *
*            STRING = Chaîne avec indications de formatage            *
**********************************************************************}

procedure GrafPrint( x, y : integer; cc, cf : byte; strt : string );

var i : integer;                                        ( Compteur d'itérations )

begin
 for i:=1 to length( strt ) do
   begin
     printchar( strt[i], x, y, cc, cf );      ( Affiche par printchar   )
     inc( x, 8 );                          ( x à la position du caractère suivant )
   end;
end;

{**********************************************************************
*  DrawAxis: Dessine des axes en haut et à gauche de l'écran          *
***********************************************************************
*  Entrées : XSTEP = Pas selon l'axe X                                *
*            YSTEP = Pas selon l'axe Y                                *
*            CC    = Couleur des caractères                           *
*            CF    = Couleur de fond (255 = transparent)              *
**********************************************************************}

procedure DrawAxis( stepx, stepy : integer; cc, cf : byte );

var x, y     : integer;                                 ( Variables d'itération )
    grad : string[3];

begin
 Line( 0, 0, MAXX, 0, cc );                                    ( Trace l'axe X  )
 Line( 0, 0, 0, MAXY, cc );                                    ( Trace l'axe Y  )
 x := stepx;                                                   ( Gradue l'axe X    )
 while ( x < MAXX ) do
   begin
     Line( x, 0, x, 5, cc );
     str( x, grad );
     if ( x < 100 ) then
       GrafPrint( x - 8 , 8, cc, cf, grad )
     else
       GrafPrint( x - 12, 8, cc, cf, grad );
     inc( x, stepx );
 end;

 y := stepy;                                                   ( Gradue l'axe Y    )
 while ( y < MAXY ) do
   begin
     Line( 0, y, 5, y, cc );
     str( y:3, grad );
     GrafPrint( 8, y-4, cc, cf, grad );
     inc( y, stepy );
 end;
end;


{**********************************************************************
*  Demo : Démontre l'usage des différentes fonctions de ce module     *
***********************************************************************
*  Entrée : néant                                                     *
**********************************************************************}

procedure Demo;

var i : integer;                                        ( Compteur d'itérations )

begin
 Randomize;          ( Initialise le générateur de nombres aléatoires )
 DrawAxis( 30, 20, 15, 255 );                                ( Trace des axes )
 GrafPrint( X1, MAXY-10, 15, 255,
            'VB060P.PAS  -  (c) by MICHAEL TISCHER' );

 Line( X1, Y1, X1, Y2, 15 );                          ( Dessine le rectangle )
 Line( X1, Y2, X2, Y2, 15 );
 Line( X2, Y2, X2, Y1, 15 );
 Line( X2, Y1, X1, Y1, 15 );

 {-- TRace des segments aléatoires à l'intérieur du rectangle ------}

 for i := 1 to NBLINE do
  Line( random( X2 - X1 - 1 ) + X1 + 1,
        random( Y2 - Y1 - 1 ) + Y1 + 1,
        random( X2 - X1 - 1 ) + X1 + 1,
        random( Y2 - Y1 - 1 ) + Y1 + 1,
        i mod 16 );
end;

{----------------------------------------------------------------------}
{--                        PROGRAMME PRINCIPAL                       ---}
{----------------------------------------------------------------------}

begin
 writeln( 'VB060P.PAS  - (c) 1991 by MICHAEL TISCHER'#13#10 );
 if IsVga then                                  ( Dispose-t-on d'une carte VGA ? )
   begin                                  ( Oui mais permet-elle le mode 800*600  ? )
     if Init800600 then
                                                                  ( Mode ok )
     begin
       Demo;                                                ( Effectue la démo )
       repeat until keypressed;                  ( Attend une frappe de touche )
       Textmode( CO80 );                                ( Revient en mode texte )
         end
       else
     writeln( 'Attention! Le mode 800*600 '+
                                    'n''a pas pu être initialisé');
     end
 else
   writeln( 'Attention! Ce programme exige une carte VGA');
end.
```

Listing : V8060PA.ASM

```
;*****************************************************************;
;*                      V 8 0 6 0 P A . A S M                    *;
;*---------------------------------------------------------------*;
;*  Fonction : contient différentes routines pour travailler dans *;
;*             le mode graphique 800*600 de la carte Super-VGA   *;
;*             avec 16 couleurs                                  *;
;*---------------------------------------------------------------*;
;*  Auteur       : MICHAEL TISCHER                               *;
;*  Développé le : 14.01.1991                                    *;
;*  Dernière MAJ : 14.01.1991                                    *;
;*---------------------------------------------------------------*;
;*  Assemblage   : MASM /mx V8060PA;      ou  TASM -mx V8060PA    *;
;*                 ... puis lier à V8060P.PAS                    *;
;*****************************************************************;

;-- Constantes ---------------------------------------------------

GC_INDEX      = 3ceh          ;Registre d'index du contrôleur graphique
GC_READ_MAP   = 4                     ;Numéro du registre Read-Map
GC_BIT_MASK   = 8             ;Numéro du registre de masquage binaire
GC_GRAPH_MODE = 5               ;Numéro du registre de mode graphique

;-- Segment de données -------------------------------------------

DATA  segment word public       ;A initialiser au moment de l'exécution
DATA  ends

;-- Programme ----------------------------------------------------

CODE  segment byte public                           ;Segment de programme

      assume cs:code, ds:data

;-- Déclarations publiques ---------------------------------------

public   init800600                       ;Initialise le mode 800 *600
public   setpix                                   ;Dessine un pixel
public   getpix                       ;Détermine la couleur d'un pixel

;-- Données dans le segment de code ------------------------------

            ;-- Numéros de code donnés par différents cartes Super VGA
                                                ;--  au mode 800*600

modenr    db 6Ah, 58h, 29h, 54h, 16h, 79h
modenrend equ this byte

;---------------------------------------------------------------------
;-- INIT800600 : initialise le mode 800*600 en 16 couleurs de la
;--              de la Super-VGA

;-- Appel depuis TP: function init800600 : boolean;
;-- Valeur de retour : TRUE = mode initialisé , FALSE = Erreur

init800600 proc near

                        ;-- Essaye tous les modes du tableau MODENR
                ;--  jusqu'à ce que l'un d'eux soit accepté par le BIOS

      mov  si,offset modenr    ;Commence par le premier mode du tableau
it1:  xor  ah,ah                    ;Fonction 00h: Initialiser le mode
      mov  al,cs:[si]                       ;Charge le code du tableau
      int  10h                                    ;Initialise le mode
      mov  ah,0fh                         ;Fonction 0Fh: Lire le mode
      int  10h
      cmp  al,cs:[si]                          ;A-t-il été enclenché ?
      je   it2                                              ;Oui--> OK

            ;-- Mauvais code, lit le suivant dans le tableau----------

      inc  si                                        ;SI sur code suivant
      cmp  si,offset modenrend          ;A-t-on parcouru tt le tableau ?
      jne  it1                                        ;Non, on recommence

      mov  al,0                        ;Oui, clôture la fonction avec erreur
      ret                                          ;Retourne à l'appelant

it2:                                  ;-- Le mode a pu être initialisé

      mov  al,1                                             ;Tout va bien
      ret                                         ;Retourne à l'appelant

init800600 endp                                    ;Fin de la procédure

;---------------------------------------------------------------------
;-- SETPIX: Dessine un pixel dans une certaine couleur
;-- Appel depuis TP: setpix( x , y : integer; couleur : byte );

setpix    proc near

sframe    struc                          ;Structure d'accès à la pile
bp0       dw ?                                          ;Mémorise BP
ret_adr0  dw ?                      ;Adresse de retour à l'appelant
couleur   dw ?                                            ;Couleur
y0        dw ?                                        ;Ordonnée Y
x0        dw ?                                        ;Abscisse X
sframe    ends                                  ;Fin de la structure

frame  equ [ bp - bp0 ]           ;Adresse les éléments de la structure

       push bp                          ;Prépare l'adressage des paramètres
       mov  bp,sp                                    ; par le registre BP

            ;-- Calcule l'offset en mémoire d'écran et le décalage ---

       mov  ax,frame.y0                                ;Charge l'ordonnée Y
       mov  dx,800 / 8                ;Multiplie par la largeur de la ligne
       mul  dx
       mov  bx,frame.x0                                ;Charge l'abscisse X
       mov  cl,bl                          ;Mémorise l'octet faible pour le
                                                     ; calcul du décalage
       shr  bx,1                                ;Divise l'abscisse X par huit
       shr  bx,1
       shr  bx,1
       add  bx,ax                ;Y ajoute l'offset issu de la multiplication

       and  cl,7                      ;Calcule le masque binaire à partir de X
       xor  cl,7
       mov  ah,1
       shl  ah,cl

       mov  dx,GC_INDEX                      ;Accède au contrôleur graphique
       mov  al,GC_BIT_MASK                 ;Charge le masque binaire dans le
       out  dx,ax             ; registre de masquage et effectue la sortie

       mov  ax,(02h shl 8) + GC_GRAPH_MODE  ;Installe le mode Write 2 &
       out  dx,ax                                              ; Read 0

       mov  ax,0A000h                                  ;Charge en ES le segment
       mov  es,ax                                        ;de la mémoire d'écran

       mov  al,es:[bx]                           ;Charge le registre latch
       mov  al,byte ptr frame.couleur            ;Fixe la couleur du pixel
       mov  es:[bx],al                       ;Réécrit dans le registre latch

       ;-- Remet les valeurs par défaut dans les registres du contrôleur
                                                ;-- graphique qui ont été modifiés

       mov  ax,(0FFh shl 8 ) + GC_BIT_MASK
       out  dx,ax

       mov  ax,(00h shl 8) + GC_GRAPH_MODE
       out  dx,ax

       pop  bp
       ret  6                                ;Retourne à l'appelant en retirant
                                                     ; les arguments de la pile
setpix endp                                            ;Fin de la procédure

;---------------------------------------------------------------------
;-- GETPIX: Détermine la couleur d'un pixel
;-- Appel depuis TP: x := getpix( x , y : integer );

getpix    proc near

sframe1   struc                          ;Structure d'accès à la pile
bp1       dw ?                                          ;Mémorise BP
ret_adr1  dw ?                      ;Adresse de retour à l'appelant
y1        dw ?                                        ;Ordonnée Y
x1        dw ?                                        ;Abscisse X
sframe1   ends                                  ;Fin de la structure

frame  equ [ bp - bp1 ]           ;Adresse les éléments de la structure

       push bp                          ;Prépare l'adressage des paramètres
       mov  bp,sp                                    ; par le registre BP

            ;-- Calcule l'offset en mémoire d'écran et le décalage-

       mov  ax,frame.y1                                ;Charge l'ordonnée Y
       mov  dx,800 / 8                ;Multiplie par la largeur de la ligne
       mul  dx
```

```
        mov   si,frame.x1                   ;Charge l'abscisse X
        mov   cx,si                    ;La mémorise pour le décalage

        shr   si,1                     ;Divise l'abscisse X par huit
        shr   si,1
        shr   si,1
        add   si,ax          ;Y ajoute l'offset issu de la multiplication

        and   cl,7             ;Calcule le masque binaire à partir de X
        xor   cl,7
        mov   ch,1
        shl   ch,cl

        mov   ax,0A000h                     ;Charge en ES le segment
        mov   es,ax                          ;de la mémoire d'écran

        mov   dx,GC_INDEX              ;Accède au contrôleur graphique
        mov   ax,(3 shl 8)+ GC_READ_MAP                  ;Lit d'abord
        xor   bl,bl                                    ;le plan 3

gpl:    out   dx,ax                    ;Indique le plan de bits à lire
        mov   bh,es:[si]          ;Charge la valeur du registre latch

        and   bh,ch                    ;Ne garde que le pixel souhaité
        neg   bh                  ;Fixe le bit 7 en fonction du pixel
        rol   bx,1                   ;Effectue une rotation du bit 7
                                        ; de BH vers le bit 1 en BL

        dec   ah                        ;Traite le plan de bits suivant
        jge   gpl                ;sup ou égal à 0 ?---> on continue

        mov   al,bl                        ;Renvoie le résultat par AL

        pop   bp
        ret   4                  ;Retourne à l'appelant en retirant
                                        ; les arguments de la pile

getpix endp                                    ;Fin de la procédure

;— Fin ————————————————————————————————————————————————

CODE    ends                                   ;Fin du segment de code
        end                                        ;Fin du programme
```

Le standard VESA

Au cours de l'histoire du PC, les constructeurs du matériel électronique ont souvent eu l'occasion de constater que l'absence de standard porte préjudice au marché. En contrepartie, l'absence de standard leur donne toujours la liberté de surpasser les produits de leurs concurrents en augmentant sans cesse les performances de leurs propres produits. Tout cela rend néanmoins la vie dure au programmeur. Cette tendance touche particulièrement le domaine des cartes Super VGA où de nouveaux fabricants continuent d'inonder en permanence le marché avec des modes texte et graphiques toujours nouveaux. Et là où des modes similaires sont proposés, les fabricants imposent quand même leurs différences en variant la numérotation de ces modes, l'organisation de la RAM vidéo et leur adressage. Conséquence : rares sont les programmes qui soutiennent les modes vidéo étendus des cartes Super VGA. Il s'agit en fait des constructeurs qui fournissent des drivers logiciels pour leurs cartes pour que les programmes puissent au moins utiliser une partie de ces modes.

Pour mettre fin à ce dilemme, les principaux fabricants de cartes VGA se sont réunis autour d'une table en 1989 pour donner naissance au comité VESA ou mieux Video Electronic Standards Association. Les partenaires éminents de cette association sont les sociétés ATI, Chips & Technologies, Everex, Genoa, Intel, Phoenix Technologies, Orchid, Paradise, Video Seven et bien d'autres encore.

Le but de cette association est de développer une extension BIOS permettant au programmeur d'accéder à des cartes Super VGA de modèles variés indépendamment de l'électronique. Cette extension BIOS, baptisée standard VESA, fut présentée publiquement en 1990. La version actulle porte le numéro 1.0. Les constructeurs ont intégré directement les extensions BIOS qui y étaient décrites dans le BIOS VGA étendu que leur cartes conservent sur une structure en ROM. Pour les anciennes cartes, ils ont décidé de fournir des drivers logiciels chargés en mémoire comme des programmes TSR et rajoutant ainsi les nouvelles fonctions VESA dans le BIOS VGA étendu. Pour la première fois, le programmeur peut adapter ses programmes de manière à les utiliser avec des cartes Super VGA variées sans être obligé de se soucier de l'électronique.

Ce chapitre décrit le standard VESA en présentant :

- ✓ les modes graphiques soutenus par ce standard,
- ✓ les différentes fonctions et la manière d'appeler ce standard,
- ✓ les tâches exécutées par les fonctions de ce standard.

Les modes graphiques du standard VESA

Le premier souci des membres du comité VESA était de se mettre d'accord sur les modes graphiques soutenus par le futur standard VESA. Chaque constructeur voulait naturellement imposer son mode graphique. Après maintes discussions, ils sont parvenus à établir une liste de 9 modes qui sont cités dans le tableau suivant.

Pour identifier les différents modes à l'aide du BIOS VESA, on lui communique les numéros de code tout comme les modes graphiques classiques reconnus par les BIOS VGA. Mais ici, le choix s'est porté sur les codes situés au-delà de 100h (256) parce que de nombreux constructeurs ont déjà affecté les codes au-dessous de 100h à leurs propres modes. Contrairement aux modes vidéo classiques représentés par un numéro de code 8 bits, les numéros de code des modes VESA ont une largeur 16 bits. La seule exception à cette règle est le mode graphique 800*600 points en 16 couleurs auquel les constructeurs ont déjà attribué le numéro de code 6Ah pour des raisons de compatibilité. Ce numéro de code est conservé dans le standard VESA. Toujours est-il que ce mode est soutenu comme un mode VESA étendu.

Les modes graphiques du BIOS VESA

Code	Résolution	Couleurs	Mémoire*
100h	640*400	256	256 Ko
101h	640*480	256	512 Ko
102h	800*600	16	256 Ko
103h	800*600	256	512 Ko
104h	1024*768	16	512 Ko
105h	1024*768	256	1 Mo
106h	1280*1024	16	1 Mo
107h	1280*1024	256	1,25 Mo
6Ah	800*600	16	256 Ko

*** Les indications concernent les capacités possibles de 256, 512 ou 1024 Ko et non le besoin en mémoire réel d'une page écran.**

Appel des fonctions du BIOS VESA

Tout comme les fonctions du BIOS VGA, celles du BIOS VESA sont accessibles par l'interruption vidéo 10h du BIOS. 6 sous-fonctions implémentées au sein de la fonction 4Fh peuvent être appelées via cette fonction. Avant l'appel de la fonction, il faut charger le numéro de fonction 4Fh dans le registre AH et le numéro de sous-fonction compris entre 0 et 5 dans le registre AL.

Tâche des fonctions du BIOS VESA

Nº	Tâche
00h	Déterminer les performances de la carte Super VGA
01h	Déterminer les donnés-clé d'un mode déterminé
02h	Définir le mode VESA
03h	Lire le mode vidéo actuel
04h	Sauvegarder/Restaurer le statut de la carte Super VGA
05h	Définir/Demander la fenêtre d'accès à la RAM vidéo

Les registres AH et AL sont également ceux dans lesquels on obtient le résultat de l'appel de fonction. Si les fonctions du BIOS VESA sont soutenues par la carte vidéo installée, on rencontre alors la valeur 4Fh dans le registre AL. Elle a été transmise auparavant à la fonction dans le registre AH. La valeur 0 signale que la fonction appelée s'est exécutée correctement. Quant à la valeur 1, elle indique que l'opération a échoué. Les autres valeurs de retour (02h à 0FFh) sont considérées comme réservées. Un message d'erreur est retourné à leur rencontre. Le contenu des autres registres du processeur, hormis AH et AL, reste inchangé par l'appel de ces fonctions tant qu'il ne sert pas à retourner des informations.

Lors de l'utilisation des fonctions VESA, il n'est pas toujours indispensable de demander le contenu des registres Ah et AL après chaque appel de fonction au risque de faire face à un échec de l'opération. Dans tous les cas, le premier appel d'une fonction VESA doit servir à confirmer si les fonctions VESA sont soutenues par la carte vidéo installée. Il faut utiliser à cet effet le résultat retourné dans les registres AH et AL.

Déterminer les performances d'une carte Super VGA

Bien que le standard VESA offre une interface aux principaux modes vidéo des cartes modernes Super VGA, il ne garantit toutefois pas que tous les modes VESA soient reconnus par une carte précise. Le premier travail effectué avec les fonctions doit donc consister à appeler la sous-fonction 00h pour obtenir les informations nécessaires concernant les caractéristiques de la carte Super VGA installée. Outre les numéros de

fonction et sous-fonction en AH et AL, la sous-fonction attend un pointeur FAR sur un buffer de 256 octets dans la paire ES:DI. La sous-fonction 00h place de nombreuses informations sur la carte Super VGA et ses caractéristiques dans ce buffer.

Structure du bloc de données de la sous-fonction 00h		
Offset	**Contenu**	**Type**
00h	Signature VESA ("VESA")	4 BYTE
04h	Version VESA, numéro de version principal	1 BYTE
05h	Version VESA, numéro de version secondaire	1 BYTE
06h	Pointeur FAR sur chaîne ASCIIZ avec le nom du fabricant de cartes	1 DWORD
0Ah	Flag décrivant la caractéristique de la carte. Inutilisé actuellement, par conséquent 0000h.	1 DWORD
0Eh	Pointeur FAR sur la liste des numéros de code des modes vidéo soutenus	1DWORD

A travers le dernier pointeur de la table, la carte installée fournit la liste des numéros de code des modes vidéo soutenus. Cette information est sans doute la plus importante pour le programmeur. Cette liste peut être située dans le BIOS en ROM de la carte ou dans la mémoire principale. Elle se constitue de différents mots. Chaque mot reçoit le numéro d'un mode soutenu où les modes VESA décrits en entrée tout comme les modes spécifiques au fabricant sont fournis par cette liste. Les derniers se distinguent facilement des modes VESA parce que leurs numéros de code sont inférieurs à 100h. L'octet de poids fort contient alors 00h dans l'entrée de liste correspondante.

Comme la longueur de la liste peut varier en fonction des caractéristiques individuelles de la carte installée et de sa capacité RAM, sa fin est marquée par un mot doté de la valeur FFFFh. Ce dernier ne représente pas un mode vidéo précis.

Lire un mode vidéo déterminé

Il ne suffit pas de connaître la disponibilité d'un mode précis pour travailler sous ce mode. C'est pourquoi, la sous-fonction 01h permet d'obtenir toutes les informations nécessaires pour programmer dans un mode précis. Outre les numéros de fonction et sous-fonction dans les registres AH et AL, il faut transmettre également le numéro du mode souhaité à la fonction. Il s'agit d'un des modes de la liste retournée par la sous-fonction 00h. Dans la paire de registres ES:DI, la fonction attend un pointeur sur un bloc de mémoire devant recevoir les informations. Ce buffer doit offrir de la place à 29 octets.

La liste des modes de la sous-fonction 00h contenant à la fois les modes VESA et les modes individuels de la carte installée, il est possible de lire des informations sur les modes VESA à l'aide de la sous-fonction 01h. Cette sous-fonction retourne alors des informations sur les modes texte étendus des différentes cartes et non seulement sur les modes graphiques.

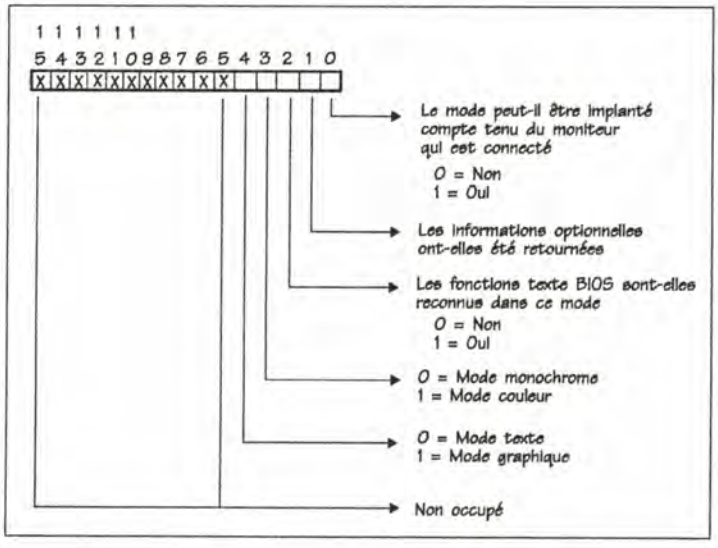
La figure suivante montre la structure des informations telles qu'elles sont entrées dans le buffer spécifié par la sous-fonction 01h.

Structure du bloc de données de la sous-fonction 01h		
Offset	**Contenu**	**Type**
00h	Flag de mode, voir plus loin	1 WORD
02h	Flags pour la première fenêtre d'accès, voir plus bas	1 BYTE
03h	Flags pour la seconde fenêtre d'accès, voir plus bas	1 BYTE
04h	Granularité en Ko utilisée pour décaler les deux fenêtres d'accès	1 WORD
06h	Taille des deux fenêtres d'accès en Ko	1 WORD
08h	Adresse de segment de la première fenêtre d'accès	1 WORD
0Ah	Adresse de segment de la seconde fenêtre d'accès	1 WORD
0Ch	Pointeur FAR sur routine de définition de la zone visible dans les deux fenêtres d'accès	1 DWORD
10h	Nombre d'octets occupés par chaque ligne de points dans la RAM vidéo (Informations optionnelles, voir Flag de mode)	1 WORD
12h	Résolution X en points/caractères	1 WORD
14h	Résolution Y en points/caractères	1 WORD
16h	Largeur de la matrice de caractères en points	1 BYTE
17h	Hauteur de la matrice de caractères en points	1 BYTE
18h	Nombre de plans de bits	1 BYTE
19h	Nombre de bits par point écran	1 BYTE
1Ah	Nombre de blocs mémoire	1 BYTE
1Bh	Modèle de mémoire	1 BYTE
1Ch	Taille des blocs de mémoire en Ko	1 BYTE

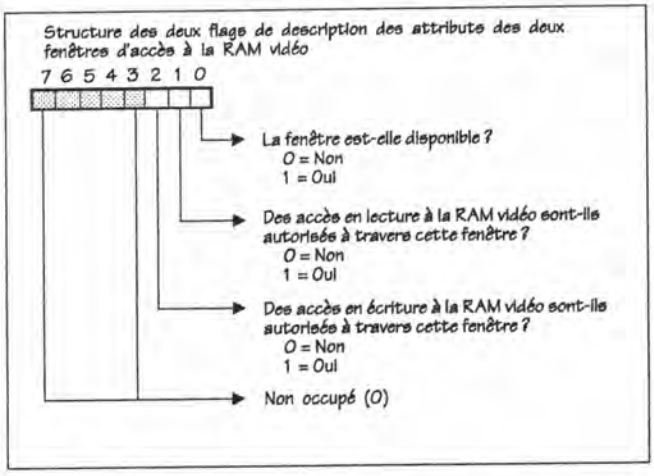
Le Flag de mode, introduisant le bloc d'informations et représentant un champ de bits, donne des informations importantes sur le mode demandé. Comme le montre la figure suivante, ce flag renseigne s'il s'agit d'un mode texte ou graphique, si la couleur est soutenue ou si le texte peut être affiché à l'aide des fonctions texte du BIOS. Il indique

également si ce mode peut effectivement être activé sur la carte vidéo compte tenu du moniteur installé et de la mémoire disponible.

Mais il est aussi important de savoir si les champs optionnels sont complétés dans le bloc de données car ils contiennent de nombreuses informations utiles dans ce mode. En règle générale, on peut supposer que ces informations sont déjà disponibles.



Les deux flags représentent également des champs de bits décrivant les fenêtres d'accès à la RAM vidéo. Toutes les cartes Super VGA ne disposant pas de deux fenêtres d'accès, elles doivent permettre de contrôler s'il existe réellement une seconde fenêtre d'accès. On apprend également ici dans quelle fenêtre il faut lire les informations et dans laquelle il faut les copier dans la mesure où il n'est pas possible de lire la RAM vidéo ou d'y écrire à travers une fenêtre.



En dehors des flags décrivant les attributs des deux fenêtres d'accès à la RAM vidéo, le bloc de données de la sous-fonction 01h reproduit d'autres informations qui sont importantes pour exploiter ces fenêtres d'accès. Il s'agit d'abord des adresses de segment des deux fenêtres d'accès. Dans ce cas, l'entrée pour la seconde fenêtre d'accès n'est valide que si une telle fenêtre existe. La taille des fenêtres d'accès ainsi que la granularité utilisée pour les décaler "à travers" la RAM vidéo sont également des informations importantes.

Pour permettre ce déplacement, le BIOS VESA fournit une routine qui assure le programmeur contre tout risque d'incompatibilité régnant entre les diverses cartes Super VGA, dans ce contexte en particulier. Pour plus d'informations sur cette routine, reportez-vous à la sous-fonction 05h à la fin de cette section.

En ce qui concerne les informations optionnelles, on cite d'abord la résolution X et Y du mode vidéo. S'il s'agit d'un mode texte, ces informations se rapportent aux lignes et colonnes sinon elles font référence à la résolution de l'écran en points. La taille de la matrice de caractères est entrée en points dans les deux champs qui suivent et en mode texte seulement.

A la suite de ces champs, vous obtenez le nombre de plans de bits intervenant dans le mode concerné et le nombre de bits utilisés pour coder un point écran. Ces deux informations ne sont fournies qu'en mode graphique. Cela vaut également pour le champ suivant qui reproduit le nombre des blocs de mémoire. Cette information n'est intéressante que dans les modes graphiques de la carte CGA et Hercules car ces deux cartes reportent les lignes graphiques de l'écran dans des blocs différents de la RAM vidéo. La taille de ces blocs de mémoire est stockée dans le dernier champ de la structure de données.

Il est cependant précédé d'un champ qui est important pour l'accès à la RAM vidéo. Il désigne le modèle de mémoire selon lequel la RAM vidéo est configurée dans le mode

vidéo concerné. Les divers formes de code sont représentées par des codes énumérés dans le tableau suivant.

Codes admis pour la description d'un modèle de mémoire	
Nº	**Fonction**
00h	Mode texte
01h	Format CGA, soit 2 ou 4 blocs mémoire
02h	Format Hercules avec 4 blocs mémoire
03h	Format EGA/VGA normal pour modes graphiques 16 couleurs
04h	Format compacté avec deux points à 4 bits par octet
05h	Format EGA/VGA normal pour modes graphiques 256 couleurs
06h-0Fh	Réservé
10h-FFh	Codes spécifiques au fabricant, inutilisés actuellement

Activer un mode

Si vous avez appelé la liste des modes vidéo soutenus à l'aide de la sous-fonction 00h, demandé les divers modes avec la sous-fonction 01h et décidé du mode répondant au mieux aux besoins de votre programme, vous pouvez activer ce mode avec la sous-fonction 02h.

Avant d'appeler la fonction, vous devez d'abord charger le numéro de code du mode souhaité dans le registre BX en dehors des numéros de fonction et sous-fonction. Si le contenu actuel de la RAM vidéo reste conservé une fois le mode activé, il faut régler le bit 15 sur 1 dans le registre BX.

Après appel de la fonction, il faut obligatoirement s'assurer que le mode souhaité peut être activé en consultant le contenu des registres AH et AL. Si vous rencontrez la valeur 00h dans le registre AH et la valeur 4Fh dans le registre AL, la mise en service du mode est verrouillée.

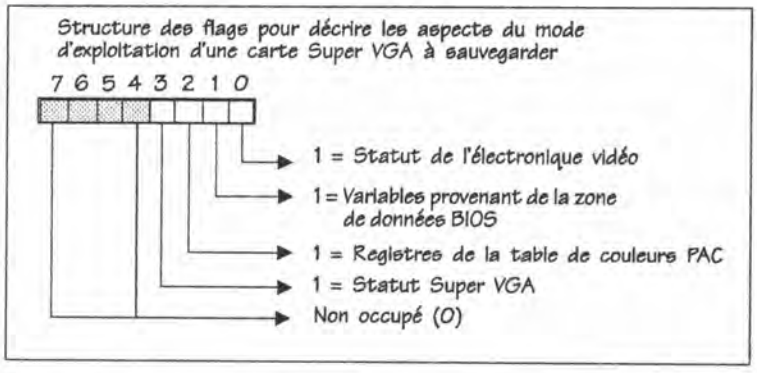
La sous-fonction 03h fait également partie du contexte des modes graphiques. Elle ne sert pas à activer un mode mais à lire le mode actuel. Pour son appel, elle attend seulement les numéros de fonction et sous-fonction. Dans le registre BX, elle retourne alors le numéro du mode vidéo actuel. Comme à l'accoutumée, les valeurs au-delà de 100h représentent les modes VESA alors que les numéros de code inférieurs correspondent à un mode VGA standard ou un mode spécifique au fabricant.

Sauvegarder et restaurer le mode d'exploitation

Lors de leur activation, des programmes TSR tels que SideKick rencontrent un problème qui les oblige à sauvegarder le mode d'exploitation actuel d'une carte avant de commuter vers un mode texte par exemple. Lorsqu'ils sont désactivés, l'écran doit être rétabli au même mode que précédemment. Et cela ne fonctionne que si on spécifie le mode actuel avant l'activation. Cette manipulation devient quelque peu complexe avec des cartes Super VGA et c'est pourquoi le BIOS VESA fournit trois sous-fonctions avec la sous-fonction 04h qui déchargent un programme de cette lourde tâche.

Avant de sauvegarder le mode actuel, il faut appeler la sous-fonction 00h. Elle communique au programmeur la quantité de mémoire nécessaire pour sauvegarder le mode d'exploitation. Outre les numéros de fonction et sous-fonction, la fonction attend la valeur 00h dans le registre DL et un champ de bits dans le registre CX. Ce dernier décrit les éléments du mode d'exploitation à sauvegarder. La figure suivante montre la structure de ce champ de bits.

Seul le contenu de la RAM vidéo que le programme doit sauvegarder puis restaurer reste exclu de la mémorisation. Mais les parties qui doivent effectivement être écrasées sont à sauvegarder. Dans un programme TSR qui commute par exemple du mode graphique vers le mode texte, il s'agit des premiers 4 Ko de la RAM vidéo utilisés pour l'affichage du texte et non la totalité de l'écran graphique qui peut avoir besoin de 1 Mo en fonction du mode graphique.



En guise de réponse, la sous-sous-fonction 00h retourne le nombre de blocs de 64 Ko dans la registre BX à réserver pour recevoir les informations d'état spécifiées. Pour sauvegarder le mode d'exploitation, le programme d'appel doit réserver un buffer dont la taille correspond à :

```
64 * BX
```

Si ce buffer est réservé dans le programme, le mode d'exploitaton peut être sauvegardé à l'aide de la sous-sous-fonction 01h. En dehors des numéros de fonction et sous-fonc-

tion, il faut indiquer trois paramètres supplémentaires dans les registres du processeur. Le registre DL attend le numéro de la sous-sous-fonction 01h et le registre CX un champ de bits décrivant les éléments du mode à sauvegarder comme pour la sous-sous-fonction 00h. Naturellement, il est important que ce champ de bits ne contienne plus d'éléments par rapport au précédent appel de la sous-sous-fonction 00h. Sinon, le buffer risque de déborder parce que la place serait insuffisante pour contenir toutes les informations.

L'adresse du buffer doit être transmise à la sous-sous-fonction 01h sous forme de pointeur FAR à travers la paire de registres ES:BX.

Les mêmes registres entrent en action lorsqu'il s'agit de restaurer le mode sauvegardé. C'est la sous-sous-fonction 02h qui est compétente dans ce domaine. Outre les numéros de fonction et sous-fonction en DL, elle attend cette valeur. A nouveau, le registre CX indiquant les éléments du mode d'exploitation à restaurer et la paire de registres ES:BX en tant que pointeur sur le buffer de sauvegarde sont réclamés.

Déplacer la fenêtre d'accès

Il est tout à fait aisé de travailler sous les modes Super VGA où chaque page écran nécessite moins de 256 Ko de mémoire. Ainsi, on peut accéder à la totalité de la RAM vidéo à travers le segment de mémoire 64 Ko à partir de A000:0000. Mais l'affaire se corse lorsqu'il faut adresser plus de 256 Ko et c'est le cas de la plupart de ces modes. On ne s'en sort alors plus avec les fenêtres d'accès qui autorisent l'accès à une fenêtre mémoire de 64 Ko à travers le segment 64 Ko en A000:0000. Grâce aux quatre plans de bits, il devient possible d'adresser simultanément 256 Ko de la RAM vidéo. Dans la section 4.9, nous avons déjà souligné les principales différences existant entre les cartes Super VGA notamment dans le mode de définition de cette fenêtre et la programmation des registres qui y sont impliqués.

Grâce à la sous-fonction 05h du BIOS VESA, ce problème appartient déjà au passé. Cette fonction se charge en effet de définir la fenêtre d'accès en tenant compte de l'électronique installée. Outre les numéros de fonction et sous-fonction ainsi que la valeur 00h dans le registre BH, elle attend le numéro de la fenêtre d'accès demandée en BL sous la forme d'une valeur 0 ou 1. Notez toutefois que la seconde fenêtre ne peut être déplacée que si la sous-fonction 01h a signalé son existence lors de son appel.

Le dernier paramètre attendu dans le registre DX est un facteur lié à la granularité de la fenêtre d'accès. Il indique le début de la zone adressable dans la RAM vidéo. Si la granularité se situe par exemple autour de 1 Ko, alors la valeur 256 rend les second 256 Ko de la RAM vidéo disponibles pour une carte Super VGA à travers le segment 64 Ko en A000:0000.

Dans les modes graphiques haute résolution d'une carte Super VGA, cette fonction étant fréquemment utilisée et son appel en tant que fonction d'interruption nécessitant beaucoup de temps, la BIOS VESA a prévu un appel direct au moyen d'un appel FAR.

Dans le bloc de données spécifié, la sous-fonction 01h retourne l'adresse de cette routine sous forme d'un pointeur FAR. Notez que la méthode FAR Call n'est pas soutenue obligatoirement par tous les BIOS VESA. Dans ce cas, l'indication est donnée comme une adresse de la routine 0000:0000.

Mais si la fonction 05h est disponible comme une routine FAR Call, vous ne devez pas hésiter à vous en servir si vous attachez quelque importance à la vitesse de votre programme. Contrairement à la fonction d'interruption, elle ne retourne pas un code d'état dans le registre AX mais elle génère le changement du registre AX et DX pendant l'appel de la fonction.

La sous-fonction 05h n'est pas seulement responsable de la définition de la fenêtre d'accès. Elle retourne également son état par rapport à la RAM vidéo. Outre les numéros de fonction et sous-fonction, il faut lui transmettre la valeur 1 dans le registre BH. Comme résultat, elle retourne aussitôt l'état de la fenêtre compte tenu de sa granularité dans le registre DX.

4.10. Programmation des cartes TIGA

De nouveaux standards graphiques se sont toujours octroyés une place dans le monde PC grâce à des résolutions élevées, à plus de couleurs ou tout simplement une netteté de l'image. Avec la reconnaissance des cartes Super VGA avec leur résolution standard de 800*600 points, on est arrivé à un tournant de l'histoire où un nouvel élément constitue la préoccupation majeure : la vitesse insatisfaisante de la sortie graphique. Voici que surgit le standard TIGA qui soulage considérablement le processeur grâce à un processeur pour la sortie graphique.

Le standard TIGA est une interface logicielle assurant le lien entre une application et une carte graphique équipée d'un processeur graphique TI34010 ou TI34020. Tout comme le processeur proprement dit, elle dispose d'un langage Assembleur et machine s'intéressant moins à l'arithmétique ou à la gestion du programme mais davantage au dessin de cercles, lignes ou polygônes. En clair, la sortie graphique s'en trouve extrêmement accélérée. Désormais, le processeur n'a plus besoin de calculer les points d'une ligne par ses propres soins et les entrer dans la RAM vidéo. Il délègue au contraire cette tâche au processeur graphique qui fonctionne parallèlement à la véritable exécution du programme.

Le développement du standard TIGA tend à atteindre quatre buts principaux décrits comme suit par Texas Instruments :

- ✓ accélérer les applications
- ✓ faciliter l'usage
- ✓ rendre extensible
- ✓ ne pas dépendre de l'électronique

La mise en place du standard TIGA et une carte graphique appropriée accélèrent considérablement les applications parce que deux processeurs assurent désormais parallèlement le traitement des tâches. L'interface TIGA est approuvée en conséquence comme une moyen de communication particulièrement souple et rapide entre le processeur principal et le processeur graphique.

Afin d'exploiter aussi confortablement que possible les facultés du standard TIGA, des fonctions graphiques fondamentales sont prédéfinies et conçues pour une interface C. Cette interface est compatible avec les divers outils Microsoft C tels que QuickC comme environnement de développement pour les applications TIGA.

Si une application a besoin des services des fonctions graphiques qui ne font pas encore partie du standard TIGA, le développeur peut élaborer des extensions TIGA en C, Assembleur ou en combinant les deux. Lors de l'exécution d'une application, ces extensions peuvent être chargées dans une carte graphique compatible TIGA et être ainsi mises en service.

Les fonctions TIGA ne sont pas liées à des considérations graphiques précises telles que la résolution de l'écran ou le nombre de couleurs affichables. Grâce au 34010, il est en effet possible de développer des cartes graphiques de toutes sortes. De ce point de vue, les fonctions TIGA ressemblent aux fonctions graphiques de Windows par exemple qui dépendent également de l'électronique.

D'une part, TIGA représente l'électronique notamment une carte graphique compatible TIGA reposant sur le processeur graphique 34010. D'autre part, le standard TIGA regroupe des éléments logiciels variés intégrés en partie sur la carte TIGA, dans la mémoire RAM du PC ou dans l'application compatible TIGA.

Citons tout d'abord la fameuse interface d'application (AI = Application Interface). Elle fournit des routines en langage évolué pour appeler les divers services offerts par TIGA. Elle est intégrée exactement comme les routines d'une bibliothèque prédéfinie lors du linkage dans le fichier programme exécutable. Les routines de cette interface fonctionnent indépendamment du type de la carte TIGA installée et n'accèdent pas à l'électronique de la carte par leurs propres soins.

Cette tâche est assurée par le driver de communication (CD = Communication Driver). Il doit être appelé et installé avant le démarrage d'une application compatible TIGA en tant que programme TSR sous le nom TIGACD.EXE. Il est appelé grâce aux routines de l'interface d'application pendant l'exécution de l'application concernée à travers une interruption prédéfinie (normalement 7Fh) pour entrer en liaison avec la carte TIGA. Au niveau PC, le driver de communication représente la partie dépendante de l'électronique de l'interface logicielle. Chaque constructeur de matériel la fournit donc individuellement avec sa carte TIGA.

Du côté de la carte TIGA, le gestionnaire de graphiques (GM = Graphics Manager) sert au driver de communication comme un programme fonctionnant dans la mémoire RAM de la carte TIGA. Sa principale tâche consiste à traiter les commandes et demandes à

travers le gestionnaire de communication et à préparer les fonctions graphiques fondamentales désignées par core primitives. Il a également la charge d'initialiser la carte et la gérer en mémoire.

Ce gestionnaire est comparable au BIOS en ROM étendu des cartes EGA et VGA. Il n'est toutefois pas stocké dans une structure ROM de la carte TIGA. Il faut par conséquent le charger au préalable dans la RAM TIGA à l'aide d'un programme spécial. C'est aussi la raison pour laquelle nous ne pouvons pas vous présenter des programmes TIGA concrets. La programmation TIGA n'est en effet réalisable qu'après l'achat d'un outil de développement approprié. Nous reviendrons sur ce sujet plus loin. Contrairement aux autres types de cartes vidéo, on ne peut pas implémenter si facilement une carte TIGA dans l'ordinateur et développer un logiciel pour cette carte.

Des fonctions étendues appelées primitives étendues peuvent être intégrées dans les fonctions graphiques de base du gestionnaire de graphiques. Le standard TIGA a prévu une série de fonctions de ce type fournies avec la carte TIGA sous la forme d'un module chargeable. Tout comme les bibliothèques de lien dynamique de Windows et OS/2, ces routines sont d'abord chargées lorsqu'elles sont appelées durant l'exécution d'un programme.

La faculté est en outre donnée à une application de charger ses propres primitives étendues dans la carte TIGA du moment qu'elles lui sont utiles. Les primitives étendues ne sont généralement pas des routines Assembleur faisant appel au processeur graphique. Il s'agit plutôt de fonctions macro destinées à intégrer les core primitives pour en constituer une entité performante. Cela tend à accélérer la sortie écran puisqu'un seul appel suffit pour provoquer les actions souhaitées dans l'électronique TIGA au lieu d'effectuer plusieurs appels TIGA en passant par l'interface d'application, le driver de communication et le gestionnaire de graphiques.

Pour qu'une application n'ait pas à renouveler l'appel TIGA une fois le précédent achevé, la carte TIGA dispose de plusieurs buffers de commande dans lesquels les appels du gestionnaire de graphiques peuvent être manoeuvrés automatiquement. Ainsi, leur traitement s'effectue en mode séquentiel. Cette faculté augmente aussi considérablement les performances d'une application compatible TIGA.

Cela est particulièrement appréciable lorsque les cartes TIGA sont utilisées avec les interfaces graphiques. Ainsi, des drivers Windows sont disponibles pour la carte TIGA de Hercules. Ils augmentent 5 fois plus la vitesse de la sortie écran quelle que soit l'application Windows en cours.

Le programmeur TIGA dispose en tout de 140 fonctions dont un bon tiers appartient à la catégorie des primitives étendues. Elles ne sont disponibles qu'une fois le module adéquat chargé. En pratique, les différentes fonctions sont réparties en 14 groupes dont un groupe spécifique baptisé Graphics Output Functions se compose exclusivement de primitives étendues. Les autres groupes contiennent surtout des core primitives mais qui sont toujours mélangées avec des primitives étendues.

Voici une brève description des différents groupes :

Graphics System Initialization Functions

Ce groupe contient toutes les fonctions concernant de près ou de loin l'initialisation de la carte TIGA ou l'interface TIGA. La principale fonction est set_videomode dont l'appel permet à une application compatible TIGA de commencer à travailler avec une carte TIGA. La fonction install_primitives fait également partie de ce groupe. Elle sert à charger les fonctions TIGA étendues. Ce groupe contient en outre toutes les fonctions liées à la demande du statut TIGA.

Clear Functions

Les fonctions de ce groupe servent à effacer des parties de l'écran ou de la RAM vidéo. A titre d'exemple, citons clear_frame_buffer, clear_page ou clear_screen.

Graphics Attribute Control Functions

Les fonctions de ce groupe définissent ou déterminent les innombrables attributs intervenant dans les opérations de dessin. En dehors de la couleur de premier plan et de fond, il faut nommer la région clipping, l'origine du repère, le motif de remplissage des surfaces ou la transparence des points écran.

Palette Functions

Les fonctions de ce groupe, appartenant à la catégorie des core primitives, concernent la palette de couleurs. Elles servent à initialiser ou lire la palette ou demander seulement un élément de la palette. La technique permettant de déterminer un élément de la palette avec la fonction get_nearest_color est intéressante puisqu'elle adapte au mieux la teinte de la couleur souhaitée.

Graphics Output Functions

Peindre et dessiner, ce sont les domaines gérés par ce groupe de fonctions. Elles servent à dessiner et à remplir des points et des lignes ainsi que des ellipses et rectangles. Le motif de remplissage peut être défini à l'aide de fonctions variées comme celles que l'on rencontre dans les programmes graphiques.

Poly Drawing Functions

Ce groupe de fonctions qui agit sur les polygônes est l'homonyme du groupe Graphics Output concernant les lignes, cercles et rectangles. Ici, il s'agit de dessiner, peindre ou remplir les polygônes par un motif.

Pixel Array Functions

Le processeur 34010 démontre toutes ses performances à travers ces fonctions qui permettent de déplacer des blocs de points rectangulaires (telle une fenêtre Windows) dans la RAM vidéo, de les échanger entre eux ou de les agrandir sur l'écran.

Text Functions

Ce groupe de fonctions s'occupe de gérer les polices et chaînes de caractères. Ses membres représentatifs sont à la fois les core primitives et les primitives étendues. Elles servent à charger et activer les fontes et à afficher les caractères avec une fonte déterminée. Il est également possible d'obtenir des informations sur la fonte actuelle et déterminer les attributs nécessaires pour sortir les caractères sur l'écran.

Cursor Functions

Un curseur écran n'est pas étranger aux cartes TIGA où les fonctions de ce groupe servent à gérer exclusivement le curseur graphique. En dehors de la faculté de définir l'emplacement du curseur, ces fonctions permettent aussi de spécifier son aspect sous la forme d'un motif de point.

Graphics Utility Functions

Ces fonctions entrent dans la catégorie dénommée "divers" dans le contexte des répertoires de bibliothèques des compilateurs en langage évolué. On y trouve des fonctions telles que wait_scan ou page_flip intervenant dans la création d'animations puisqu'elles permettent de concevoir la structure des pages écran en tâche de fond.

Pointer-Based Memory Management Functions

C'est à peine croyable, mais l'interface TIGA reconnaît également des fonctions permettant de gérer la mémoire TIGA ou du moins une partie sous la forme de Heap. Cette gestion Heap se rapproche considérablement de C et c'est pourquoi on rencontre ici des fonctions comme gsp_malloc, gsp_free ou gsp_realloc. Toutes ces fonctions soutiennent celles de la carte TIGA au cours de son travail.

Communications Functions

Les fonctions de ce groupe s'avèrent utiles lorsqu'il s'agit d'échanger des données entre le PC et la carte TIGA ou entre le 34010 et un coprocesseur.

Extensibility Functions

Ces fonctions entrent en jeu lorsqu'il s'agit d'étendre les primitives aux primitives étendues. Elles permettent de charger les modules relogeables, de les supprimer et les remplacer ensuite par de nouveaux.

Pour soutenir le développement matériel et logiciel compatible TIGA, Texas Instruments commercialise trois coffrets dont la qualité et le prix répondent parfaitement aux besoins du développeur.

A un niveau supérieur, le kit du développeur (DDK = Driver Developer's Kit) aide le développeur de logiciels pour la création d'applications devant accéder à l'interface TIGA lors de la sortie écran. Ce coffret contient la documentation et une interface logicielle adaptée en vue d'une utilisation commune avec un compilateur Microsoft C. Le coffret fournit également des exemples pour des drivers logiciels TIGA prédéfinis tels que le code source du driver TIGA pour AutoCAD, version 9. Aux Etats-Unis, ce coffret est disponible pour moins de 500 $.

On doit se plonger davantage dans le sujet si on souhaite étendre les fonctions graphiques TIGA prédéfinies aux primitives étendues pouvant être chargées dans une carte graphique compatible TIGA. L'outil nécessaire à cet effet, le kit du développeur (SDK), coûte plus de 1000 $. Outre le DDK intégral et un Assembleur pour les processeurs 34010, il contient un compilateur C et d'autres outils servant à élaborer des programmes pour ce processeur. Vous pouvez par exemple charger directement votre générateur fractal sur la carte TIGA ou y incorporer un algorithme de ray-tracing.

Mais si vous souhaitez développer non seulement des logiciels compatibles TIGA mais aussi l'électronique, vous devez vous procurer le programme Software Porting Kit (SPK) qui coûte au moins 15000 $. A ce prix, l'acheteur se voit offrir le DDK, le SDK ainsi que le code source intégral de TIGA servant à effectuer une adaptation par rapport à l'électronique 34010 personnelle. Un driver Windows pour Windows 286 ainsi que la licence autorisant à exploiter une carte graphique compatible TIGA sont fournis dans le coffret.

Des standards ont certes apparu dans l'histoire du PC, mais en général ce ne sont pas les caractéristiques techniques d'un composant électronique ou logiciel qui ont favorisé la mise en place d'un standard. C'est plutôt l'influence du constructeur qui a joué dans cette voie. Le standard une fois fort apprécié se voit généralement dépassé technologiquement et ralentit le progrès parce que des efforts considérables sont investis dans l'électronique et qu'il faut d'abord les amortir.

Le phénomène est complètement différent dans le cas TIGA car ce standard procure une grande confiance, il est bien documenté et conçu dans une vision future, ce qui le fera certainement durer pendant quelques années. Nous avons la certitude que TIGA représentera le standard graphique des années 90 et sera approuvé simultanément comme l'interface utilisateur graphique par excellence.

5. Le clavier

Le clavier est un périphérique auquel vous aurez forcément affaire si vous programmez sous DOS. En tant que moyen privilégié d'entrée des données, il se trouve au centre de la plupart des applications, même les programmes résidents ne peuvent éviter de le prendre en compte.

Il est vrai que les programmes résidents ont une manière particulière de s'occuper du clavier. Les programmes d'application ordinaires se contentent d'interroger le clavier pour savoir quelle est la touche actionnée par l'utilisateur. Mais cette tâche n'est pas aussi primaire que l'on croit au premier abord. Il faut par exemple tenir compte de l'état des touches de commandes CAPS, NUM et LOCK et faire une distinction entre les touches qui donnent des caractères visibles (lettres, chiffres et symboles spéciaux) et celles qui déclenchent certaines actions (touches de direction et touches de fonction).

Comme le montrera la section 5.2, ce point est encore assez simple. Mais les choses se compliquent lorsque les programmes TSR ou les routines ISR interceptent les touches pour pouvoir réagir avant le programme d'application en cours d'exécution. La section 5.3 expliquera ce type de possibilité.

La section 5.4 est consacrée à la programmation du clavier car dans une certaine mesure le mode de fonctionnement de ce périphérique peut être contrôlé par logiciel. C'est ainsi que l'on peut fixer la vitesse de répétition des caractères lorsqu'on maintient une touche enfoncée ou commander les diodes témoins.

Pour éclairer le contexte dans lequel se passent ces événements, la section 5.1 montre le chemin parcouru par un caractère depuis la frappe d'une touche jusqu'à sa prise en compte par le programme d'application en cours d'exécution. Vous y apprendrez aussi quels sont les différents types de claviers usuels.

5.1. Fondements de la programmation du clavier

Le chapitre 1 a déjà montré comment les différents niveaux du matériel, du BIOS et de DOS collaborent pour gérer le clavier. Nous allons reparler ici de cette collaboration et l'approfondir pour préparer le terrain aux études ultérieures.

5.1.1. Une longue route : de la touche au programme

Tout commence par l'utilisateur qui actionne une touche. Il en résulte un signal électrique qui exprime la position de la touche frappée. Ce signal est traité par le circuit maître du clavier qui se trouve à l'intérieur de ce dernier. Il s'agit en général d'un processeur Intel dénommé 8048 ou d'un circuit équivalent en provenance d'un autre fabricant. Du côté de l'ordinateur et dans le cas des AT uniquement le clavier est géré par un contrôleur

8042 d'Intel. Ce dernier autorise une communication bilatérale entre l'ordinateur et le clavier ce qui n'est pas possible avec les PC et les XT.

Conversion en Scan code

Le circuit du clavier transforme le signal électrique de la position de la touche en un numéro appelé Scan code. Ce code n'a aucun rapport avec le caractère ou la fonction de la touche enfoncée. Il représente simplement un numéro qui reste à exploiter.

Pour cela le circuit du clavier transmet le Scan code à l'ordinateur. Si c'est un AT, le Scan code est réceptionné par le contrôleur de clavier. La transmission se fait en mode série car le câble qui relie le clavier à l'ordinateur ne comporte qu'une ligne de données. Contrairement à ce qui se passe pour l'interface série du PC, à laquelle on connecte par exemple une imprimante, un modem ou une souris, la communication n'est pas asynchrone mais synchrone. Ceci veut dire qu'à côté de la ligne des données il existe une ligne qui véhicule un signal de synchronisation à fréquence fixe. Le signal varie sans cesse entre les niveaux Hi et Lo (1 et 0), et c'est à ce rythme que sont transmis les différents bits du Scan code. Si plusieurs touches ont été actionnées en même temps, le circuit du clavier commence par stocker les frappes dans un buffer interne qui peut en général contenir 10 caractères. Il ne se remplit jamais car la transmission est bien trop rapide pour que l'utilisateur puisse suivre.

Codes Make et Break

Les Scan codes ne sont pas seulement générés lorsqu'on enfonce une touche mais également lorsqu'on la relâche. Le système sait donc à tout moment si une touche a été libérée ou si elle est restée pressée. Ce point est très important car ce n'est qu'à ce prix que l'action de plusieurs touches simultanées peut être correctement prise en compte. Si cette possibilité n'existait pas on ne pourrait pas afficher de majuscules car des dernières résultent de la frappe simultanée de la touche Majuscule et d'une touche représentant un caractère. Pensez aussi à la célèbre combinaison <Ctrl><Alt><Suppr> qui redémarre le système.

Pour distinguer les Scan codes qui expriment l'enfoncement et le relâchement des touches, on parle de "make codes" et de "break codes"

Les deux ne diffèrent que par le bit 7 qui est égal à 0 dans le code Make, à 1 dans le code Break.

Il en résulte deux conséquences importantes. Les codes Break sont toujours supérieurs ou égaux à 128, tandis que les codes Make sont inférieurs à 128. Par ailleurs un clavier de PC ne peut pas avoir plus de 128 touches, sinon les codes Make déborderaient sur les codes Make, ce qui donnerait un joli mélange.

En pratique la frappe simultanée de plusieurs touches se passe de la façon suivante. Pour taper un A majuscule l'utilisateur enfonce d'abord la touche Majuscule droite puis la touche du caractère "A". A ce moment le contrôleur du clavier a déjà transmis à l'ordinateur le code Make de la touche Majuscule. Quel que soit le système et le clavier, ce code est égal à 36h. A sa suite arrive le code Make de la lettre A, égal à 1Eh. A ce moment comme le système n'a pas encore reçu le code Break de la touche Majuscule droite, il sait que le caractère doit être considéré comme une majuscule.

Mais qui reçoit les Scan codes du côté du PC et dans quelles conditions ?

Le gestionnaire du clavier du BIOS

Chaque fois que le clavier envoie un code Make ou Break à l'ordinateur, il déclenche en même temps une interruption matérielle IRQ1. Ceci conduit à l'appel de l'interruption 09h derrière laquelle se cache une routine du BIOS, le gestionnaire du clavier, qui reçoit les codes Make et Break et les transforme en codes ASCII fournis au programme en cours d'exécution. Mais le cheminement est encore assez long.

Le gestionnaire du clavier doit d'abord se préoccuper de prendre connaissance des codes Make et Break. Il se sert à cet effet d'un port d'entrée-sortie qui possède toujours l'adresse 60h et est spécialisé dans le recueil des codes Make et Break. En étudiant le code, le gestionnaire du clavier détermine le moment où la saisie d'un caractère est achevée.

En effet chaque frappe ne correspond pas forcément à l'entrée d'un caractère, comme nous l'avons déjà vu dans l'exemple du A majuscule. Dans cet exemple le gestionnaire ne générait une lettre que lorsqu'il percevait le code Make de la touche A et non pas au moment où la touche Majuscule était enfoncée. Pensez également à l'introduction d'un code de caractère par la touche ALT combinée avec des chiffres du pavé numérique. De nombreuses touches sont enfoncées et relâchées avant que l'introduction du caractère ne soit achevée.

Lorsque le gestionnaire a reconnu le caractère tapé, il doit le convertir en un code compréhensible par le programme d'application. Le rôle des scan codes s'arrête là car les différents types de claviers existants n'exploitent pas toujours les mêmes codes, encore qu'une certaine unanimité se fasse malgré tout sentir.

Les scan codes sont donc convertis en codes ASCII, un codage universellement admis dans le monde des PC. En toute rigueur, le code ASCII ne comporte que 128 caractères, mais la variante "étendue" en exploite 256. L'annexe M donne la table des codes ASCII.

La caractère converti n'est pas immédiatement transmis au programme d'application mais déposé dans un buffer dont l'organisation et le fonctionnement ont été décrits au chapitre 3. A ce moment le gestionnaire du clavier a rempli son devoir. Il est maintenant de la responsabilité du programme d'application de ressortir le code du buffer et de le

traiter. L'interruption 16h du BIOS propose plusieurs fonctions à cet effet : elles sont décrites au chapitre 5.2. Comme on le voit, leur appel met fin à un long voyage .

Adaptation des claviers étrangers

Mais restons encore un moment avec le gestionnaire du clavier. Il est exact que le BIOS en ROM définit un tel gestionnaire mais lorsqu'on travaille sous DOS il est remplacé par un autre programme. Il s'agit du fameux KEYB (autrefois KEYBFR) qui est probablement mentionné dans votre AUTOEXEC.BAT. Il intervient parce que le gestionnaire en ROM est prévu pour les claviers américains, qui ignorent par exemple les caractères é, è, ù, ç et dont l'emplacement des touches diffère (Q à la place du A, etc).

Faute de charger le gestionnaire KEYB pour remplacer celui du BIOS, on obtiendrait à l'écran un "Q" lorsqu'on tape un "A". L'ordinateur ne sait pas lire ce qui est gravé sur les touches : ce qui compte c'est la conversion du scan code en ASCII prise en charge par le gestionnaire du clavier. Il est facile de comprendre que le scan code d'une touche reste le même, qu'elle s'appelle "Q" ou "A".

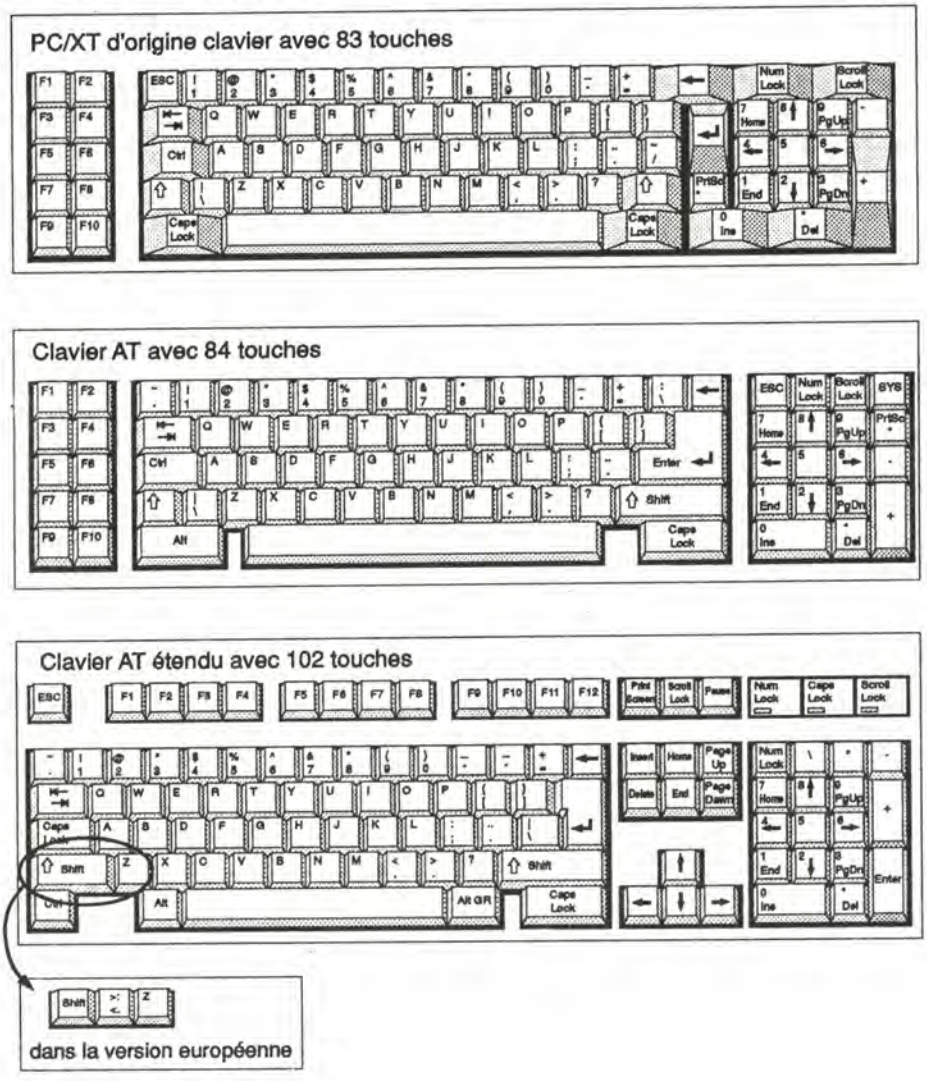
Si certains programmes sous DOS (notamment des jeux) comprennent "Q" lorsque vous tapez "A", le responsable n'en est pas le gestionnaire mais le mode d'interrogation du clavier. Ces programmes en effet, au lieu de prendre en compte le code ASCII tout prêt, se saisissent directement du scan code et font eux-mêmes la conversion ASCII. Comme ils n'ont pas la possibilité de reconnaître la nationalité du clavier, ils ne savent rien de ce qui est gravé sur les touches.

Mais un gestionnaire de clavier n'est pas seulement capable de faire représenter à l'écran les caractères gravés sur les touches. Il sait aussi étendre la fonctionnalité de la touche, comme c'est le cas lorsqu'en langue française on tape le caractère ê à l'aide des touches ^ et e.

5.1.2. Les différents types de claviers

L'affectation des touches d'un clavier peut être très variable : il existe des claviers suédois, français, anglais, allemands. Mais sur le plan des principes il n'existe que trois claviers standard dans le monde du PC. Tous trois ont été introduits par les micro-ordinateurs d'IBM. Le standard concerne le nombre de touches et les scan codes. Les symboles reproduits sur les touches dépendent de la version nationale en cause mais la disposition fondamentale des lettres, des chiffres, des caractères spéciaux, des touches de direction et de fonction reste inchangée. La figure suivante montre les trois standards.

LES TROIS CLAVIERS STANDARD DANS LE DOMAINE DU PC



Le PC est né avec le clavier PC/XT de 83 touches. Du point de vue ergonomique, ce clavier était une vraie catastrophe car la touche Entrée et les deux touches Majuscule étaient beaucoup trop petites.

Cette erreur de conception fut réparée lors de l'introduction de l'AT qui bénéficia d'un clavier amélioré de 84 touches. La touche Entrée était correctement dimensionnée de sorte que l'opérateur ne risquait plus de taper à côté lors d'une frappe soutenue. De même les deux touches Majuscule avaient la bonne taille. Mais comme les dimensions

de l'ensemble n'avaient pas varié, d'autres touches durent réduire leur encombrement. Ce fut le cas des touches de verrouillage numérique (Num Lock) et d'arrêt du défilement (Scroll Lock) qui ne sont pas d'un usage intensif et qui n'avaient pas de raison d'être plus grandes que les touches Majuscule. Le même sort frappa la touche + à droite du pavé numérique. Une nouvelle touche apparut : la touche Sys plus tard rebaptisée "Sys Req". Elle devait constituer une sorte de touche de fonction spécialisée dans les appels au système ou aux programmes résidents, mais elle ne fut jamais exploitée par les concepteurs de logiciels.

Le clavier Etendu

IBM décida de rompre avec le design de son clavier classique en introduisant le clavier appelé "étendu", conçu à l'origine pour l'AT.

Le clavier étendu présente des caractéristiques tout à fait remarquables qui lui ont assuré une percée rapide :

- ✓ un bloc de touches de direction a été rajouté pour décharger le pavé numérique
- ✓ les touches de fonction ont été déplacées dans la rangée supérieure, comme c'est le cas pour les claviers des grands systèmes IBM
- ✓ des touches de fonction supplémentaires F11 et F12 ont été rajoutées, également dans un but de compatibilité avec les grands systèmes
- ✓ la touche de commande Alt a été mise dans la rangée de la barre d'espacement pour être plus accessible. Par ailleurs la touche Alt Gr a été créée pour simuler la frappe simultanée des touches Alt et Ctrl
- ✓ trois diodes électroluminescentes affichent l'état du verrouillage numérique, du verrouillage des majuscules et de l'arrêt du défilement.

Le clavier étendu existe en deux versions différentes : la version US possède 101 touches tandis que la version européenne en comporte 102, soit une de plus. Cette touche supplémentaire a été disposée a droite de la touche Majuscule gauche qui a subi de ce fait une amputation. Elle n'apporte pas beaucoup d'avantages aux utlisateurs européens car les symboles qu'elle porte sont accessibles par une autre touche alors que la réduction de la touche Majuscule gauche restaure l'inconfort du clavier du PC/XT tant décrié à l'époque.

En dépit de ce défaut et du déplacement des touches de fonction, le clavier étendu est aujourd'hui le plus répandu dans le monde du PC : il équipe pratiquement tous les systèmes courants.

Le logiciel, c'est-à-dire essentiellement le gestionnaire du clavier, reconnaît le clavier étendu à une marque de reconnaissance qui est réceptionnée à la suite d'une requête spéciale. Les deux autres claviers ne possèdent pas ce perfectionnement.

Claviers des Laptops et des Notebook

L'apparition des micro-ordinateurs portables de type Laptop ou Notebook a entraîné une nouvelle multiplication des claviers. En tant que programmeur, il n'est cependant pas nécessaire de se bagarrer avec eux car il émulent en principe l'un des trois claviers standard. Dans certains cas les scan codes originaux sont émis à partir de l'intérieur du clavier, dans d'autres cas les scan codes divergents sont convertis en ASCII classique par le gestionnaire du clavier. Cette dernière méthode est plus rare d'emploi car les drivers du genre KEYB doivent être adaptés à des scan codes différents.

Dès qu'un fabricant aquiert la licence de DOS auprès de Microsoft ou Digital Research, il est libre de remanier KEYB à sa guise. Mais il doit s'attendre à ce que certains programmes résidents ne fonctionnent pas correctement sur son clavier car ces derniers l'interrogent généralement au plus bas niveau, celui des scan codes.

Clavier et souris

Il existe aussi des claviers qui comportent des extensions pour commander une souris, par exemple une boule (trackball) ou des touches spéciales. Le programmeur n'a pas à s'en soucier car leur traitement s'effectue par l'interface souris classique (étudiée au chapitre 9).

5.2. Accès au clavier par le BIOS

Pour l'interrogation du clavier à partir des programmes d'application, le BIOS offre trois fonctions différents qui seront examinées dans la première partie de cette section. Ces fonctions ignorent les touches de fonction supplémentaires du clavier étendu et ne font pas de distinction entre les touches de direction situées dans le pavé numérique et celles qui sont disposées à part. La deuxième section est consacrée aux extensions des fonctions du BIOS qui prennent en compte les particularités des claviers étendus.

Les deux dernières sections n'intéresseront que les développeurs de programmes résidents ou d'utilitaires spéciaux liés au clavier. Elles sont consacrées aux différentes variables utilisées par le BIOS pour gérer le clavier et aux scan codes.

5.2.1. Les fonctions de l'interrruption 16h du BIOS

Les codes ASCII générés par le gestionnaire du clavier du BIOS ou de DOS (KEYB) peuvent être obtenus par deux fonctions de l'interruption 16h. Il s'agit des fonctions 0 et 1 auxquelles est associée une autre fonction qui porte le numéro 2. Cette dernière donne en fait l'état des touches de commande.

Interrogation du clavier

La fonction 00h de l'interruption 16h du BIOS permet à un programme de prendre connaissance d'une touche frappée. Comme argument d'entrée, il suffit de mettre le numéro de la fonction dans le registre AH.

Si avant l'appel l'utilisateur a tapé un caractère qui se trouve en attente dans le buffer clavier du BIOS, la fonction renvoie ce caractère sous forme d'un code ASCII mémorisé dans le registre AL.

Si aucun caractère ne se trouve dans le buffer, la fonction atttend que l'utilisateur tape une touche. Ce n'est qu'à ce moment qu'elle rend la main : selon la patience de l'utilisateur, l'attente peut durer des secondes, des minutes, des heures ou des années.

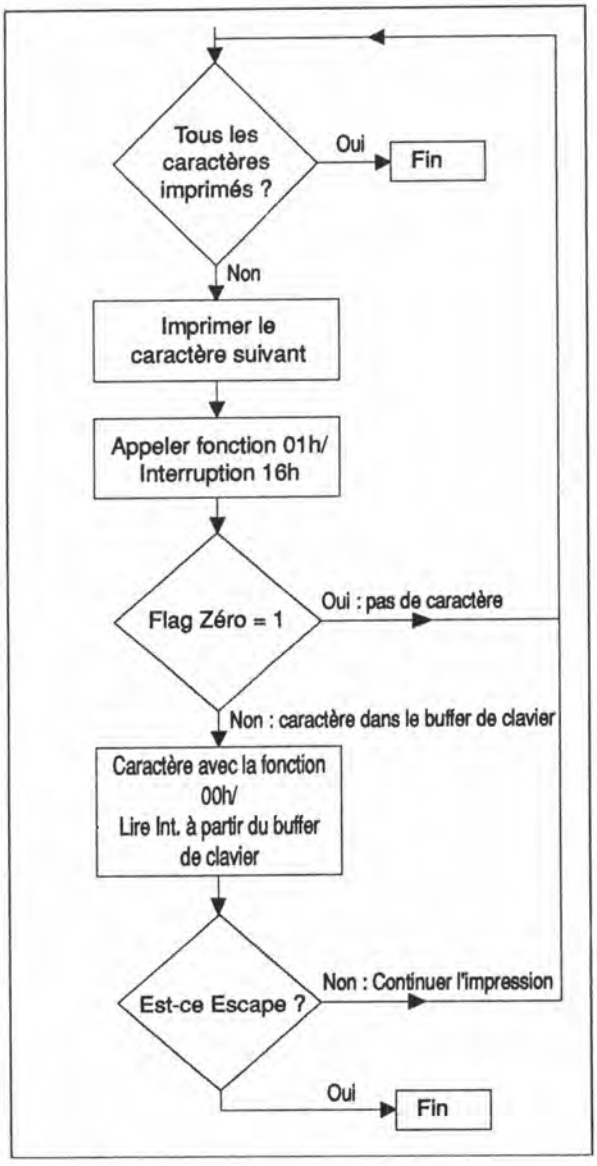
Ce mécanisme n'est pas toujours très pratique. Il existe des situations où un processus doit se poursuivre jusqu'à la frappe d'une touche. Par exemple lorsqu'on imprime un document, il est normal de prévoir l'interruption de la tâche au moyen d'une touche.

La fonction 00H ne permet pas la programmation correspondante car elle ne rend le contrôle que lorsqu'on tape une touche. Ce qu'il faudrait, c'est une fonction qui informe le programme de la frappe d'une touche : tel est précisément le rôle de la fonction 01h.

Une fois appelée, cette dernière rend immédiatement le contrôle à l'utilisateur, qu'un caractère se trouve ou non dans le buffer. L'information recherchée est fournie par l'indicateur de zéro du registre des indicateurs. Cet indicateur est à 1 si le buffer est vide et à 0 s'il contient des caractères. Dans ce cas, le registre AL contient le code ASCII du prochain caractère comme pour la fonction 00h.

Pour appeler la fonction 01h, le seul argument d'entrée nécessaire est le numéro de la fonction, à mettre dans AH. Contrairement à la fonction 00h, elle ne retire pas le caractère renseigné du buffer. Si une touche est identifiée et traitée par la fonction 01h, son code doit ensuite être retiré du buffer par la fonction 00h, sinon il y reste et le BIOS le retournera si l'appel est renouvelé.

L'algorithme suivant montre comment on peut organiser l'impression d'un document en autorisant son interruption par la touche Esc.



Utilisation des fonctions 00 h et 01h pour l'interruption d'une impression

Codage des touches

Il n'est pas tout à fait juste de prétendre que les fonctions 00h et 01h ne servent qu'à renvoyer en AL le code ASCII de la touche frappée. A l'issue de leur appel, le registre AH contient aussi le scan code que le clavier a envoyé au gestionnaire et qui a permis

la conversion en ASCII. La section 5.2.4 vous en dira davantage sur les scan codes. Normalement les programmes d'application n'ont besoin que du code ASCII.

Ici le code ASCII est associé au jeu des caractères standard du PC tels qu'ils peuvent apparaître sur l'écran. Lorsqu'un programme réceptionne le code ASCII 65, le caractère concerné est le "A" majuscule qui peut être transmis tel quel aux fonctions d'affichage. La figure suivante rappelle le jeu de caractères ASCII du PC.

Déc.	Hex.	Caract.	Déc.	Hex.	Caract.	Déc.	Hex.	Caract.	Déc.	Hex.	Caract.	Déc.	Hex.	Caract.	Déc.	Hex.	Caract.	Déc.	Hex.	Caract.	Déc.	Hex.	Caract.
0	00		32	20		64	40	@	96	60	`	128	80	Ç	160	A0	á	192	C0	└	224	E0	α
1	01	☺	33	21	!	65	41	A	97	61	a	129	81	ü	161	A1	í	193	C1	┴	225	E1	ß
2	02	☻	34	22	"	66	42	B	98	62	b	130	82	é	162	A2	ó	194	C2	┬	226	E2	Γ
3	03	♥	35	23	#	67	43	C	99	63	c	131	83	â	163	A3	ú	195	C3	├	227	E3	π
4	04	♦	36	24	$	68	44	D	100	64	d	132	84	ä	164	A4	ñ	196	C4	─	228	E4	Σ
5	05	♣	37	25	%	69	45	E	101	65	e	133	85	à	165	A5	Ñ	197	C5	┼	229	E5	σ
6	06	♠	38	26	&	70	46	F	102	66	f	134	86	å	166	A6	ª	198	C6	╞	230	E6	µ
7	07	•	39	27	'	71	47	G	103	67	g	135	87	ç	167	A7	º	199	C7	╟	231	E7	τ
8	08	◘	40	28	(	72	48	H	104	68	h	136	88	ê	168	A8	¿	200	C8	╚	232	E8	Φ
9	09	○	41	29	)	73	49	I	105	69	i	137	89	ë	169	A9	⌐	201	C9	╔	233	E9	Θ
10	0A	◙	42	2A	*	74	4A	J	106	6A	j	138	8A	è	170	AA	¬	202	CA	╩	234	EA	Ω
11	0B	♂	43	2B	+	75	4B	K	107	6B	k	139	8B	ï	171	AB	½	203	CB	╦	235	EB	δ
12	0C	♀	44	2C	,	76	4C	L	108	6C	l	140	8C	î	172	AC	¼	204	CC	╠	236	EC	∞
13	0D	♪	45	2D	-	77	4D	M	109	6D	m	141	8D	ì	173	AD	¡	205	CD	═	237	ED	φ
14	0E	♫	46	2E	.	78	4E	N	110	6E	n	142	8E	Ä	174	AE	«	206	CE	╬	238	EE	ε
15	0F	☼	47	2F	/	79	4F	O	111	6F	o	143	8F	Å	175	AF	»	207	CF	╧	239	EF	∩
16	10	►	48	30	0	80	50	P	112	70	p	144	90	É	176	B0	▒	208	D0	╨	240	F0	≡
17	11	◄	49	31	1	81	51	Q	113	71	q	145	91	æ	177	B1	▒	209	D1	╤	241	F1	±
18	12	↕	50	32	2	82	52	R	114	72	r	146	92	Æ	178	B2	▓	210	D2	╥	242	F2	≥
19	13	‼	51	33	3	83	53	S	115	73	s	147	93	ô	179	B3	│	211	D3	╙	243	F3	≤
20	14	¶	52	34	4	84	54	T	116	74	t	148	94	ö	180	B4	┤	212	D4	╘	244	F4	⌠
21	15	§	53	35	5	85	55	U	117	75	u	149	95	ò	181	B5	╡	213	D5	╒	245	F5	⌡
22	16	▬	54	36	6	86	56	V	118	76	v	150	96	û	182	B6	╢	214	D6	╓	246	F6	÷
23	17	↨	55	37	7	87	57	W	119	77	w	151	97	ù	183	B7	╖	215	D7	╫	247	F7	≈
24	18	↑	56	38	8	88	58	X	120	78	x	152	98	ÿ	184	B8	╕	216	D8	╪	248	F8	°
25	19	↓	57	39	9	89	59	Y	121	79	y	153	99	Ö	185	B9	╣	217	D9	┘	249	F9	∙
26	1A	→	58	3A	:	90	5A	Z	122	7A	z	154	9A	Ü	186	BA	║	218	DA	┌	250	FA	·
27	1B	←	59	3B	;	91	5B	[	123	7B	{	155	9B	¢	187	BB	╗	219	DB	█	251	FB	√
28	1C	∟	60	3C	<	92	5C	\	124	7C	|	156	9C	£	188	BC	╝	220	DC	▄	252	FC	n
29	1D	↔	61	3D	=	93	5D	]	125	7D	}	157	9D	¥	189	BD	╜	221	DD	▌	253	FD	²
30	1E	▲	62	3E	>	94	5E	^	126	7E	~	158	9E	₧	190	BE	╛	222	DE	▐	254	FE	■
31	1F	▼	63	3F	?	95	5F	_	127	7F	⌂	159	9F	ƒ	191	BF	┐	223	DF	▀	255	FF	

Touches ASCII

Les touches de commande telles que Majuscule (Shift), Contrôle, Alt, Verrouillage numérique (Num Lock), Verrouillage des majuscules (Caps Lock) et arrêt du défilement (Scroll Lock) ne sont pas renvoyées sous forme de code ASCII mais influencent l'action d'autres touches.

Il existe cependant certaines touches de commande qui sont associés à des codes ASCII : elles ne génèrent pas de caractère mais peuvent déclencher certaines actions dans les programmes. Sont ainsi concernées les touches d'Effacement arrière (Backspace), d'Entrée ou d'Echappement (Esc) :

Touches de commande associées à un code ASCII			
Code	**Signification**	**Code**	**Caractère**
8	Backspace	BS	
9	Tabulation	TAB	
10	Linefeed (Ctrl+Entrée)	LF	
13	Entrée	CR	
27	Escape	ESC	

Tous ces codes peuvent être obtenus par combinaison de la touche Ctrl avec une lettre (principe valable pour tous les codes ASCII inférieurs à 33) :

Dec	Symbol	Touches
0	Vide (rien)	Ctrl+2
1	☺	Ctrl+A
2	☻	Ctrl+B
3	♥	Ctrl+C
4	♦	Ctrl+D
5	♣	Ctrl+E
6	♠	Ctrl+F
7	•	Ctrl+G
8	BS	Ctrl+H, Backspace Shift+Backspace
9	TAB	Ctrl+I
10	LF	Ctrl+J Ctrl+↵
11	♂	Ctrl+K
12	♀	Ctrl+L
13	CR	Ctrl+M,↵, Shift+↵
14	♫	Ctrl+N
15	☼	Ctrl+O

Dec	Symbol	Touches
16	►	Ctrl+P
17	◄	Ctrl+Q
18	↕	Ctrl+R
19	!!	Ctrl+S
20	¶	Ctrl+T
21	§	Ctrl+U
22	▬	Ctrl+V
23	↨	Ctrl+W
24	↑	Ctrl+X
25	↓	Ctrl+Y
26	→	Ctrl+Z
27	← ESC	Ctrl+[, ESC, Shift+ESC, Ctrl+ESC
28	└	Ctrl+\
29	↔	Ctrl+]
30	▲	Ctrl+6
31	▼	Ctrl+ -
32	Espace	Touche Espace, Espace+Shift, Espace+Ctrl, Espace+Alt

Introduction de caractères de contrôle avec la touche Ctrl

La signification particulière des codes ASCII 8, 9, 10, 13 et 27 provoque quelques difficultés. Par exemple si l'utilisateur veut taper une petite flèche vers la gauche, ce caractère ayant le code ASCII 27 est interprété par la plupart des programmes comme une commande d'échappement (Esc).

Si vous voulez éviter ce dilemme dans l'un de vos propres programmes, il convient de définir une touche à actionner avant le caractère ambivalent, par exemple F1. Votre programme devra toujours garder en mémoire le dernier caractère tapé et lorsqu'il reçoit le code associé à Esc, Entrée ou Tab, il devra vérifier s'il n'a pas été précédé de F1. Si tel est le cas, la touche ne doit pas être interprétée comme une commande mais comme un caractère ordinaire.

Codes de clavier étendus

A propos de F1, signalons que les deux fonctions du BIOS retournent également des codes de clavier étendus. En effet si les 256 caractères du jeu ASCII comportent quelques caractères de contrôle du genre Tab, Entrée ou Echappement, il n'y a plus de place dans leur jeu pour les touches de fonction et de direction.

Les codes de ces touches sont donc exprimés d'une autre manière. Leur code ASCII proprement dit est 0 mais leur indentification dite "code étendu" est contenue dans le registre AH, c'est-à-dire là où se trouve habituellement le scan code.

Si à l'appel de l'une des deux fonctions 00h et 01h votre programme tombe sur la valeur 0 dans le registre AL, vous devez continuer les recherches en inspectant le registre AH qui contient l'indication de la touche frappée.

Ce processus permet de rajouter 256 codes au jeu ASCII ordinaire, ainsi toutes les possibilités ainsi ouvertes ne sont pas exploitées. Le tableau suivant donne la liste des codes de clavier étendus que vous pouvez réceptionner en interrogeant les fonctions 00h et 01h de l'interruption 16h du BIOS. Les combinaisons de touches qui ne sont pas mentionnées, par exemple celles qui font intervenir Ctrl, Majuscule et une lettre ne sont pas reconnues par le BIOS et ne génèrent donc pas de code propre.

Codes de clavier étendus		
Hexadécimal	**Décimal**	**généré par**
0Fh	15	Majuscule + Tab
2ème rangée		
10h	16	Alt + Q
11h	17	Alt + W
12h	18	Alt + E
13h	19	Alt + R
14h	20	Alt + T
15h	21	Alt + Z
16h	22	Alt + U

Codes de clavier étendus		
Hexadécimal	**Décimal**	**généré par**
17h	23	Alt + I
18h	24	Alt + O
19h	25	Alt + P
3ème rangée		
1Eh	30	Alt + A
1Fh	31	Alt + S
20h	32	Alt + D
21h	33	Alt + F
22h	34	Alt + G
23h	35	Alt + H
24h	36	Alt + J
25h	37	Alt + K
26h	38	Alt + L
4ème rangée		
2Ch	44	Alt + Y
2Dh	45	Alt + X
2Eh	46	Alt + C
2Fh	47	Alt + V
30h	48	Alt + B
31h	49	Alt + N
32h	50	Alt + M
3bh	59	F1
3ch	60	F2
3dh	61	F3
3eh	62	F4
3fh	63	F5
40h	64	F6
41h	65	F7
42h	66	F8
43h	67	F9

Codes de clavier étendus		
Hexadécimal	**Décimal**	**généré par**
44h	68	F10
47h	71	Home
48h	72	Curseur haut
49h	73	Page up
4Bh	75	Curseur gauche
4Dh	77	Curseur droit
50h	80	Curseur bas
51h	81	Page Down
52h	82	Insert
53h	83	Delete
54h	84	Majuscule + F1
55h	85	Majuscule + F2
56h	86	Majuscule + F3
57h	87	Majuscule + F4
58h	88	Majuscule + F5
59h	89	Majuscule + F6
5Ah	90	Majuscule + F7
5Bh	91	Majuscule + F8
5Ch	92	Majuscule + F9
5Dh	93	Majuscule + F10
5Eh	94	Ctrl + F1
5Fh	95	Ctrl + F2
60h	96	Ctrl + F3
61h	97	Ctrl + F4
62h	98	Ctrl + F5
63h	99	Ctrl + F6
64h	100	Ctrl + F7
65h	101	Ctrl + F8
66h	102	Ctrl + F9
67h	103	Ctrl + F10

Codes de clavier étendus		
Hexadécimal	**Décimal**	**généré par**
68h	104	Alt + F1
69h	105	Alt + F2
6Ah	106	Alt + F3
6Bh	107	Alt + F4
6Ch	108	Alt + F5
6Dh	109	Alt + F6
6Eh	110	Alt + F7
6Fh	111	Alt + F8
70h	112	Alt + F9
71h	113	Alt + F10
73h	115	Ctrl + Curseur gauche
74h	116	Ctrl + Curseur droit
75h	117	Ctrl + End
76h	118	Ctrl + Page Down
77h	119	Ctrl + Home
78h	120	Alt + 1 (1re rangée)
79h	121	Alt + 2
7Ah	122	Alt + 3
7Bh	123	Alt + 4
7Ch	124	Alt + 5
7Dh	125	Alt + 6
7Eh	126	Alt + 7
7Fh	127	Alt + 8
80h	128	Alt + 9
81h	129	Alt + 0

Les combinaisons absentes du tableau ne peuvent pas être identifiées par les fonctions de clavier du BIOS car elles ne génèrent pas de code étendu. Tel est notamment le cas des combinaisons de plusieurs touches de commande (Alt, Ctrl et Majuscule) avec les touches de fonction. Il existe des programmes DOS capables de traiter ces touches parce qu'ils possèdent un gestionnaire de clavier propre qui leur donne une puissance accrue.

Touches particulières

Certaines touches ne peuvent pas être traitées lorsqu'elles déclenchent une certaine action à l'intérieur-même du gestionnaire de clavier : leur code n'est donc pas déposé dans le buffer du BIOS.

C'est ainsi que la frappe de Imp Ecran (Prt Scr) provoque automatiquement l'appel de l'interruption 05h qui est une routine d'impression du contenu de l'écran (hard copy).

La touche de pause (Ctrl + Num pour les claviers de PC/XT) agit différemment. Elle fige le système jusqu'à ce qu'une nouvelle touche soit frappée. Ce processus demeure transparent pour le programme en cours d'exécution.

L'enfoncement simultané des touches Ctrl et Break provoque l'appel de l'interruption 1Bh. Normalement le programme en cours d'exécution est interrompu et on se trouve renvoyé au niveau système. Pour empêcher ce mécanisme, vous pouvez détourner cette interruption en lui substituant votre propre routine, qui se réduira en fait à l'instruction IRET. L'exécution du gestionnaire d'interruption s'achève aussitôt et le contrôle est rendu au programme.

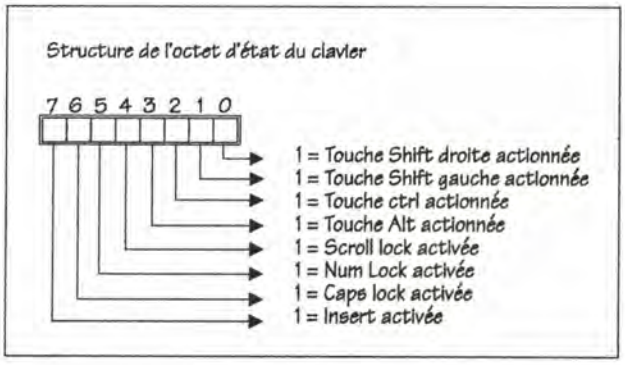
La touche SysRq ou SysReq n'existe que sur les claviers d'AT ou les claviers étendus. Lorsqu'on appuie dessus, on déclenche l'interruption 15h avec une valeur de 8500h dans le registre AX. Le même phénomène se reproduit lorsqu'on relâche la touche, la valeur introduite dans AX étant alors 8501h.

Le nombre 85h en AH signifie que c'est la fonction numéro 85h de l'interruption 15 h qui est appelée. Cette fonction se réduit habituellement à une instruction IRET. Autrement dit, la frappe de la touche demeure sans effet. Mais il ne tient qu'à vous de développer un gestionnaire d'interruption pour l'interruption 15h qui effectue les tâches que vous voudrez bien lui confier. Notez cependant que vous ne devez toucher qu'à la fonction 85h et renvoyer les autres appels à l'ancienne interruption 15h.

Lecture de l'état du clavier

La troisième fonction de l'interruption clavier du BIOS sert à lire l'état du clavier. Elle porte le numéro 02h. Elle retourne dans le registre AL la valeur d'une variable que le BIOS tient à jour à chaque frappe dans son segment de variables. Son nom : l'indicateur d'état du clavier.

Il indique la situation (active ou non) des touches Majuscule, Alt, Ctrl, Verrouillage numérique, Verrouillage des majuscules, Arrêt du défielment et Ins. La figure suivante montre comment les différents bits de l'indicateur expriment l'état des touches. Par exemple si le bit 3 est à 1, l'utilisateur est en train de presser la touche Alt. Si le bit 6 est à 1, l'utilisateur a verrouillé les majuscules.



Vous voyez que l'indicateur fait une distinction entre Majuscule droite et Majuscule gauche mais qu'il ne fait aucune différence entre les diverses touches Alt et Ctrl du clavier étendu. Il est vrai lorsque l'indicateur d'état du clavier et la fonction 02h de l'interruption clavier du BIOS ont été élaborés, le clavier étendu n'existait pas encore.

L'indicateur d'état du clavier intervient fréquemment dans les programmes résidents lorsqu'il s'agit de détecter la touche d'activation qui doit les déclencher. Ce point sera étudié au chapitre 32 qui est entièrement consacré aux programmes résidents et se préoccupe intensément de la lecture du clavier.

Exemples de programmes

Pour clore cette section je voudrais vous présenter trois exemples de programmes écrits en Basic, Pascal et C et qui démontrent l'usage des fonctions du BIOS décrites plus haut. Ils s'appellent TOUCHEB.BAS, TOUCHEC et TOUCHEP.PAS.

Leur tâche est d'afficher en permanence pendant la frappe d'une chaîne de caractères l'état actuel des touches de verouillage numérique, de verrouillage des majuscules et de la touche d'insertion Ins.

Ce programme se justifie surtout pour les possesseurs de claviers de PC/XT ou d'AT car les claviers étendus ont des diodes électroluminescentes qui indiquent l'activation des touches mentionnées (sauf la touche Ins).

Le but poursuivi par le programme a été sélectionné avec beaucoup de soin car la simple lecture des caractères tapés au clavier ne justifie que rarement l'appel direct aux fonctions du BIOS. Tous les langages évolués possèdent en effet des instructions appropriées. En Basic, c'est Input$, en Pascal la fonction ReadKey, en C getch(). Toutes ces instructions font plus ou moins appel aux fonctions du BIOS.

Malgré tout, notre exemple montre qu'il existe des circonstances qui réclament l'exploitation des fonctions du clavier de DOS en langage évolué.

Au centre du programme se trouve une fonction appelée GetKey qui attend que l'utilisateur tape une touche et renvoie le code de la touche tapée à l'appelant. Mais commençons par le commencement.

Au début du programme est appelée la routine INIKEY qui lit l'état des touches de commande par l'intermédiaire de la fonction 02h de l'interruption 16h du BIOS. Dans l'indicateur renvoyé, seuls les trois bits les plus à gauche nous intéressent car ce sont eux qui concernent les touches Ins, Verrouillage numérique, et Verrouillage des majuscules. L'indicateur permet d'initialiser trois variables qui décrivent l'état d'activation de chaque touche surveillée. Ces variables sont soumises à un opérateur de négation de sorte que le mode Ins est activé lorsque la variable associée a pour valeur False (0). La raison de cette transformation va apparaître dans un instant.

Une fois les variables internes initialisées, GETKEY peut être invoqué pour lire le clavier. C'est à GETKEY que nous allons nous intéresser maintenant.

GETKEY prend d'abord connaissance de l'indicateur d'état par la fonction 02h du BIOS. Il le compare aux trois variables internes pour déceler les modifications éventuelles. S'il y a effectivement modification, le nouvel état est affiché en haut à droite de l'écran, pour que l'utilisateur en soit informé. L'action décrite a lieu individuellement pour chacune des variables modifiées. La négation entreprise au début du programme a pour but de forcer la prise en compte d'une modification dès le premier appel à GETKEY. En effet lors de ce premier appel, le programme va se rendre compte qu'aucune des variables internes ne correspond à l'indicateur d'état d'où une mise à jour de l'affichage sans initialisation préalable.

GETKEY peut à présent faire son travail et tester le clavier en se servant de la fonction 01h de l'interruption clavier du BIOS. Une touche a-t-elle été frappée ? Si tel n'est pas le cas, on recommence le test. La même boucle sera parcourue sans cesse aussi longtemps qu'aucune touche n'a été actionnée mais tout changement dans l'état des touches de commande sera affiché immédiatement à l'écran.

Dès qu'une touche a été transférée dans le buffer du BIOS, la boucle est interrompue et le caractère mémorisé est lu par la fonction 00h de l'interruption clavier du BIOS. La dernière partie de la routine teste le code pour voir s'il ne s'agit pas d'un code clavier étendu. Si oui, on ajoute 256 au code transmis pour signaler au programme appelant la présence d'un code étendu. Ainsi les codes ASCII ordinaires et les codes étendus pourront être facilement distingués : un code inférieur ou égal à 256 est forcément un code ASCII ordinaire tandis qu'un code supérieur à 256 est du type étendu.

GETKEY s'achève par le calcul de ce code qui est retourné par la fonction sous la forme d'un nombre entier.

Dans les trois programmes présentés, GETKEY est utilisé pour lire des caractères tapés, les afficher à l'écran et répéter ce processus jusqu'à la frappe de Entrée ou F1. Si pendant l'introduction d'une chaîne de caractères vous actionnez les touches de verrouillage des

nombres (Num Lock), des majuscules (Caps Lock) ou de l'insertion (Ins), le résultat s'affiche aussitôt à l'écran.

Une routine aussi stratégique que GETKEY peut encore réaliser bien d'autres choses. Vous pouvez par exemple faire en sorte que la frappe d'une touche unique génère toute une chaîne, c'est-à-dire implémenter une macro. Vous pouvez aussi afficher un texte d'aide lorsqu'une touche déterminée se trouve détectée pendant la frappe. En examinant les trois listings, vous observerez que la version en langage C n'a pas nécessité la programmation de routines spéciales pour invoquer les trois fonctions du BIOS. Ces fonctions sont en effet prédéfinies dans la bibliothèque standard des compilateurs utilisés. Notez cependant que lesdites fonctions portent des appellations différentes chez Borland et chez Microsoft. Des macros situées au début du programme prennent en compte ces différences.

Voici les listings des trois programmes :

Listing : TOUCHEB.BAS

```
'*****************************************************************
'*                          TOUCHEB                              *
'*---------------------------------------------------------------*
'*  Fonction : Implémente une fonction qui permet de lire un caractère *
'*             en affichant l'état des touches de bascule INSERT, CAPS *
'*             NUM                                               *
'*---------------------------------------------------------------*
'* Auteur                : MICHAEL TISCHER                       *
'* Développé le          : 09.06.1991                            *
'* Dernière modification : 01.01.1992                            *
'*****************************************************************
'
DECLARE SUB PrintInvers (Texte AS STRING)
DECLARE SUB WriteChar (Caractere AS INTEGER, Couleur AS INTEGER)
DECLARE FUNCTION GetPage% ()
DECLARE SUB Inikey ()
DECLARE FUNCTION NegFlag%(Flag%,FlagReg%,Colonne%,Ligne%,Text AS STRING)
DECLARE FUNCTION GetKey% ()

'$INCLUDE: 'QB.BI'    'Fichier contenant les déclarations des registres

CONST TRUE = -1                                  'Valeur booléenne vraie
CONST FALSE = NOT TRUE

CONST FL = 1                            'Ligne d'affichage des indicateurs
CONST FC = 65                        'Colonne d'affichage des indicateurs
 n
'-- Bits de la variable d'état du clavier dans le BIOS ----------------
CONST SCRL = 16                                          'Bit Scroll-Lock
CONST NUML = 32                                            'Bit Num-Lock
CONST CAPL = 64                                           'Bit Caps-Lock
CONST INS = 128                                             'Bit Insert

'-- Code retourné par GETKEY pour quelques touches --------------------

CONST BEL = 7                                      'Code du signal sonore
CONST BS = 8                                  'Code de la touche Backspace
CONST TB = 9                               'Code de la touche de tabulation
CONST LF = 10                                'Code de la touche Line-Feed
CONST CR = 13                                   'Code de la touche Entrée
CONST ESC = 27                            'Code de la touche d'échappement
CONST F1 = 315                                        'Touches de fonction
CONST F2 = 316
CONST F3 = 317
CONST F4 = 318
CONST F5 = 319
CONST F6 = 320
CONST F7 = 321
CONST F8 = 322
CONST F9 = 323
CONST F10 = 324
CONST CUP = 328                                       'Touches de direction
CONST CLEFT = 331                                            'Code A gauche
CONST CRIGHT = 333                                           'Code A droite
CONST CDOWN = 329                                         'Code Vers le bas

'-- Variables globales -------------------------------------------------

DIM SHARED Insert AS INTEGER                  'Etat de la touche Insert
DIM SHARED Caps   AS INTEGER                'Etat de la touche Caps-Lock
DIM SHARED Num    AS INTEGER                 'Etat de la touche Num-Lock

'-- Programme principal ------------------------------------------------

DIM Touche AS INTEGER                             'code ASCII d'une touche

Inikey                              'Initialise les indicateurs du clavier
CLS                                                      'Efface l'écran
PRINT "TOUCHEB  (C) 1987, 91 by MICHAEL TISCHER"
PRINT
PRINT "Tapez quelques caractères en activant ou désactivant";
PRINT "les touches INSERT, CAPS ou NUM"
PRINT "L'état de ces touches va être affiché ";
PRINT " à tout moment "
PRINT "dans le coin supérieur droit de l'écran"
PRINT "La frappe de  <Entrée> ou <F1> termine ";
PRINT "le programme ..."
PRINT : PRINT "Votre saisie : ";
DO                                                     'Boucle de saisie
  Touche = GetKey                                        'Lit une touche
  IF Touche < 256 THEN                        'Pas de code clavier étendu ?
    PRINT (CHR$(Touche));                              'Affiche le caractère
  END IF
LOOP UNTIL (Touche = CR) OR (Touche = F1)  'répéter jusqu'à <F1> ou <CR>
END

'*****************************************************************
'* GetKey  : Lit un caractère et affiche l'état de s indicateurs *
'* Entrée : néant                                                *
'* Sortie : Code de la touche frappée                            *
'*                  < 256 : touche ordinaire                     *
'*                 >= 256 : touche à code étendu                 *
'*****************************************************************
'
FUNCTION GetKey%

DIM Reg AS RegType           'registres du processeur pour l'interruption

DO
  Reg.ax = &H1200          'Numéro de la fonction "Lire l'état du clavier"
  CALL INTERRUPT(&H16, Reg, Reg)          'Déclenche l'interruption BIOS

  '-- Ajuste les indicateurs en fonction du nouvel état ----------------
Insert = NegFlag(Insert, ((Reg.ax MOD 256) AND INS), FC+9, FL, "INSERT")
Caps = NegFlag(Caps, ((Reg.ax MOD 256) AND CAPL), FC + 3, FL, " CAPS ")
```

```
Num = NegFlag(Num, ((Reg.ax MOD 256) AND NUML), FC, FL, "NUM")

  Reg.ax = &H1100                'Numéro de la fonction "Caractère prêt ?"
  CALL INTERRUPT(&H16, Reg, Reg)       'Déclenche l'interruption du BIOS
  IF (Reg.flags AND 64) = 0 THEN                          'Touche prête
    Reg.ax = &H0                  'Numéro de la fonction "Lire la touche"
    CALL INTERRUPT(&H16, Reg, Reg)     'Déclenche l'interruption du BIOS
    IF (Reg.ax MOD 256) = 0 THEN              'Est-ce un code étendu ?
      GetKey = (Reg.ax \ 256) OR &H100                            'oui
    ELSE                                                        'non
      GetKey = (Reg.ax MOD 256)
    END IF
    EXIT DO                                          'Quitte la boucle
  END IF
LOOP                          'Répétition jusqu'à réception d'une frappe
END FUNCTION

'*************************************************************************
'* GetPage : Lit la page d'écran courante                                *
'* Entrée  : néant                                                       *
'* Sortie  : Page d'écran courante                                       *
'*************************************************************************
'
FUNCTION GetPage%

DIM Registre AS RegType     'registres du processeur pour l'interruption

Registre.ax = &H1500                          'AH = Numéro de la fonction
CALL INTERRUPT(&H10, Registre, Registre) 'Déclenche interruption du BIOS
GetPage = Registre.bx \ 256                'Registre.BH indique la page

END FUNCTION

'*************************************************************************
'* InitKey  : Initialise les indicateurs des touches                     *
'* Entrée : néant                                                        *
'* Sortie : néant                                                        *
'* Info   : Les indicateurs sont inversés par rapport à eur état         *
'*          actuel pour que ce dernier puisse être affiché au            *
'*          prochain appel de GETKEY                                     *
'*************************************************************************
'
SUB InitKey

SHARED Insert AS INTEGER                        'Etat de la touche Insert
SHARED Caps   AS INTEGER                      'Etat de la touche Caps-Lock
SHARED Num    AS INTEGER                        'Etat de la touche Num-Lock

DIM Registre AS RegType     'Registres du processeur pour l'interruption

Registre.ax = &H0200       'Numéro de la fonction "Lire l'état du clavier"
CALL INTERRUPT(&H16, Registre, Registre) 'Déclenche interruption du BIOS

IF (Registre.ax AND INS) THEN                     'Fixe l'indicateur INSERT
  Insert = FALSE
ELSE
  Insert = TRUE
END IF

IF (Registre.ax AND CAPL) THEN                'Fixe l'indicateur Caps-Lock
  Caps = FALSE
ELSE
  Caps = TRUE
END IF

IF (Registre.ax AND NUML) THEN                'Fixe l'indicateur Num-Lock
  Num = FALSE
ELSE
  Num = TRUE
END IF

END SUB

'*************************************************************************
'* NegFlag : Inverse un indicateur si nécessaire et affiche le texte     *
'*           associé                                                     *
'* Entrées:                                                              *
'* Sortie: nouvel état indicateur (True = actif, False = désactivé )     *
'*************************************************************************
'
FUNCTION NegFlag% (Flag%, FlagReg%, Colonne%, Ligne%, Texte AS STRING)

DIM LigneCour AS INTEGER       'Mémorise la position courante du curseur
DIM ColCour AS INTEGER

'-- Teste si l'état s'est modifié ------------------------------------

IF Flag% AND (FlagReg% = 0) OR (NOT Flag%) AND (FlagReg% <> 0) THEN
  LigneCour = CSRLIN                                 'Oui: mémorise la ligne
  ColCour = POS(0)                                              'et la colonne
  LOCATE Ligne%, Colonne%               'Position du nom de l'indicateur
  IF FlagReg% = 0 THEN                              'Si indicateur désactivé
    NegFlag = FALSE                                 'résultat de la fonction
    PRINT SPACE$(LEN(Texte))                              'Efface indication
  ELSE                                                     'Indicateur actif
    NegFlag% = TRUE                                   'Résultat de la fonction
    PrintInvers (Texte)                    'Affiche le nom de l'indicateur
  END IF
  LOCATE LigneCour, ColCour        'Remet le curseur à son ancienne place
ELSE                                                ' l'état n'a pas changé
  NegFlag% = Flag%                                          'Il reste le même
END IF

END FUNCTION

'*************************************************************************
'* PrintInvers : Affiche un chaîne de caractères en inverse vidéo        *
'*               à la position et sur la page courantes                  *
'* Entrées :                                                             *
'* Sortie     : néant                                                    *
'*************************************************************************
'
SUB PrintInvers (Texte AS STRING)

CONST INVERS = &H70                              'Attribut vidéo inverse
DIM Compteur AS INTEGER                           'Compteur d'itérations

FOR Compteur = 1 TO LEN(Texte)          'Pour tous les caractères du texte
  CALL WriteChar(ASC(MID$(Texte, Compteur, 1)), INVERS)
  LOCATE CSRLIN, POS(0) + 1                              'Avance le curseur
NEXT

END SUB

'*************************************************************************
'* WriteChar : Affiche un caractère avec un attribut donné               *
'*             à la position et sur la page courantes                    *
'* Entrée    :                                                           *
'* Sortie    : néant                                                     *
'*************************************************************************
'
SUB WriteChar (Caractere AS INTEGER, Couleur AS INTEGER)

DIM Registre AS RegType     'Registres du processeur pour l'interruption

Registre.ax = &H9 * 256 + Caractere  'AH = fonction "Afficher caractère"

Registre.bx = GetPage * 256 + Couleur     'BH = Page écran ; BL = Couleur
Registre.cx = 1                                          'Un seul caractère
CALL INTERRUPT(&H10, Registre, Registre) 'Déclenche interruption du BIOS

END SUB
```

Listing : TOUCHEP.PAS

```
{*********************************************************************}
{*                          T O U C H E P                            *}
{*-------------------------------------------------------------------*}
{* Fonction: Implémente une fonction qui permet de lire un caractère *}
{*           au clavier en affichant l'état des touches de bascule   *}
{*           INSERT, CAPS et NUM                                     *}
{*                                                                   *}
{*-------------------------------------------------------------------*}
{*   Auteur               : MICHAEL TISCHER                          *}
{*   Développé le         : 8.07.1987                                *}
{*   Dernière modification : 1.01.1992                               *}
{*********************************************************************}

program TASTP;

Uses Crt, Dos;                          { Inclut les unités Crt et Dos }

{$V-}                       { Exclut le contrôle de la longueur des chaînes }

type FlagText = string[6]; { Pour transmettre les noms des indicateurs }


const FL      = 1;                      { Ligne d'affichage des indicateurs }
      FC      = 65;                  { Colonne d'affichage des indicateurs }
      FlagCara = 0;                        { Couleur des caractères }
      FlagFond = 7;                    { Couleur de fond des indicateurs }


      (** Disposition des bits dans la variable d'état du clavier *****}
      SCRL =  16;                                        { Scroll-Lock }
      NUML =  32;                                        { Num-Lock }
      CAPL =  64;                                        { Caps-Lock }
      INS  = 128;                                        { Insert }

      (** Quelques codes de touche retournés par GETKEY ***************}
      BEL     = 7;                                  { Code du signale sonore }
      BS      = 8;                          { Code de la touche Backspace }
      TAB     = 9;                     { Code de la touche de tabulation }
      LF      = 10;                        { Code de la touche Linefeed }
      CR      = 13;                           { Code de la touche Entrée }
      ESC     = 27;                  { Code de la touche d'échappement }
      F1      = 315;                             { Touches de fonction }
      F2      = 316;
      F3      = 317;
      F4      = 318;
      F5      = 319;
      F6      = 320;
      F7      = 321;
      F8      = 322;
      F9      = 323;
      F10     = 324;
      CUP     = 328;                          { Touches de direction }
      CLEFT   = 331;
      CRIGHT  = 333;
      CDOWN   = 328;

var Insert,                                  { Etat de l'indicateur INSERT}
    Num,                                      { Etat de l'indicateur NUM }
    Caps     : boolean;                       { Etat de l'indicateur CAPS }
    CoulCara,                             { Couleur de caractère courante }
    CoulFond,                                { Couleur de fond courante }
    Touche   : integer;                         { Code de la touche lue }

{*********************************************************************}
{* NegFlag: Inverse un flag si nécessaire et affiche le texte associé *}
{* Entrées: cf infra                                                 *}
{* Sortie : Nouvel état de l'indicateur (true=actif, false=inactif)  *}
{*********************************************************************}

function NegFlag(Flag   : boolean;      { dernier état de l'indicateur }
                 FlagReg, { état courant de l'indicateur (0 = inactif) }
                 Colonne, { Colonne d'affichage du nom de l'indicateur }
                 ligne  : integer;      { Ligne d'affichage du nom ...}
                 Texte  : FlagText) : boolean;  { Nom de l'indicateur }

var LigneCour,                                    { Ligne courante }
    ColCour : integer;                          { Colonne courante }

begin
  if (Flag and (FlagReg = 0)) or                      { Teste si l'état }
     (not(Flag) and (FlagReg <> 0)) then     { de l'indicateur a changé }
  begin                                                       { OUI }
    LigneCour := WhereY;                        { Mémorise ligne courante }
    ColCour := WhereX;                            { et colonne courante }
    gotoxy(Colonne, ligne);                   { Repositionne le curseur }
    if FlagReg = 0 then                   { L'indicateur est il inactif }
      begin                                                   { OUI }
        NegFlag := false;                       { Résultat de la fonction }
        TextColor( Black );                  { Couleur des caractères = noir }
        TextBackground( Black );                        { Fond = noir }
      end
    else
      begin                                             { Indicateur actif }
        NegFlag:=true;                                { Résultat de la fonction }
        TextColor(FlagCara);              { Couleur des caractères = FLAGCARA }
        TextBackground(FlagFond)                { Couleur de fond = FLAGFOND }
      end;
    write(Texte);                          { Affiche le nom de l'indicateur }
    GotoXY(ColCour, LigneCour);                  { Repositionne le curseur }
    TextColor(CoulCara);                { Rétablit la couleur des caractères }
    TextBackground(CoulFond)                    { Rétablit la couleur de fond }
  end
  else
    NegFlag := Flag                          { Etat de l'indicateur inchangé }
end;

{*********************************************************************}
{* GetKey : Lit un caractère et affiche l'état des indicateurs       *}
{* Entrée : néant                                                    *}
{* Sortie : Code de la touche frappée < 256 : touche ordinaire       *}
{*                                    >= 256 : touche étendue        *}
{*********************************************************************}

function GetKey : integer;

var Regs       : Registers;              { Registres pour l'interruption }
    TouchePrete : boolean;                  { Indique si touche frappée }

begin
  TouchePrete := false;                          { Pas de touche frappée }
  repeat
    Regs.ah := $2;          { Numéro de la fonction "Lire l'état du clavier "}
    intr($16, Regs );              { Déclenche l'interruption clavier du BIOS }

    {** Ajuste les indicateurs en fonction du nouvel état du clavier   **}
    Insert := NegFlag(Insert, Regs.al and INS, FC+9, FL, 'INSERT');
    Caps := NegFlag(Caps, Regs.al and CAPL, FC+3, FL, ' CAPS ');
    Num := NegFlag(Num, Regs.al and NUML, FC, FL, 'NUM');
    Regs.ah := $1;                { Numéro de la fonction "Touche frappée ? " }
    intr($16, Regs);              { Déclenche l'interruption clavier du BIOS }
    if (Regs.flags and FZero = 0) then          { Indicateur de zéro à 1 ? }
      begin                                          { OUI (Touche frappée ) }
        TouchePrete := true;                                { Lit la touche }
        Regs.ah := 0;                  { Numéro de la fonction "Lire une touche" }
        intr($16, Regs );               { Apelle l'interruption clavier du BIOS }
        if (Regs.al = 0)                      { Est-ce une touche étendue ? }
          then GetKey := Regs.ah or $100                              { Oui }
          else GetKey := Regs.al;                                     { Non }
      end;
  until TouchePrete;    { Répète l'opération jusqu'à frappe d'une touche }
end;

{*********************************************************************}
{* IniKey : Initialise les indicateurs des touches                   *}
{* Entrée : néant                                                    *}
{* Sortie : néant                                                    *}
{* Info   : Les indicateurs sont inversés par rapport à leur état    *}
{*          actuel pour que ce dernier puisse être affiché au        *}
{*          prochain appel de GETKEY                                 *}
{*********************************************************************}

procedure IniKey;

var Regs       : Registers;              { Registres pour l'interruption }

begin
  Regs.ah := $2;          { Numéro de la fonction "Lire l'état du clavier"}
  intr($16, Regs );              { Déclenche l'interruption clavier du BIOS }
  if (Regs.al and INS <> 0) then Insert := false    { Fixe l'indicateur }
                            else Insert := true;                 {INSERT}
  if (Regs.al and CAPL <> 0) then Caps := false     { Fixe l'indicateur }
                             else Caps := true;                  { CAPS }
  if (Regs.al and NUML <> 0) then Num := false      { Fixe l'indicateur }
                             else Num := true                { NUM LOCK }
end;

{*********************************************************************}
{* Couleur : Fixe les couleurs des car. et du fond pour l'affichage  *}
{* Entrée  : cf infra                                                *}
{* Sortie  : néant                                                   *}
{* Var.    : Les couleurs sont mémorisées par les variables COULFOND *}
{*           et COULCARA                                             *}
{* Info    : Cette procédure doit être appelée pour qu'à l'issue     *}
{*           d'un affichage de l'état des indicateurs la couleur des *}
{*           caractères courante puisse être restaurée. Turbo Pascal *}
{*           ne dispose pas d'un fonction qui permette de lire cette *}
```

```
{*          couleur                                                  *}
{************************************************************************}

procedure Couleur(PremierPlan, ArrierePlan : integer);

begin
  CoulCara := PremierPlan;          ( mémorise la couleur des caractères )
  CoulFond := ArrierePlan;             ( Mémorise la couleur du fond )
  TextColor(PremierPlan);            ( Fixe la couleur des caractères )
  TextBackground(ArrierePlan);          ( Fixe la couleur du fond )
end;

{************************************************************************}
{*                          PROGRAMME PRINCIPAL                         *}
{************************************************************************}

begin
  InitKey;                              ( Initialise les indicateurs )
  Couleur(7,0);                                  ( Couleur = blanc / noir )
  ClrScr;                                              ( Efface l'écran )
  writeln(@13@10'TOUCHEP (c) 1987, 92 by Michael Tischer');
  writeln(@13@10'Tapez quelques caractères en activant ou désactivant'+
          @13@10'les touches INSERT, CAPS ou NUM');
  writeln('L''état de ces touches va être affiché en permanence ');
  writeln('dans le coin supérieur gauche de l''écran.');
  writeln('Pour sortir du programme tapez <Entrée> ou <F1>');
  write(@13@10'Votre saisie : ');
  repeat                                              ( Boucle de saisie )
    Touche := GetKey;  ( Lit une touche )
    if (Touche < 256) then write(chr(Touche))        ( Affiche la touche )
  until (Touche = CR) or (Touche = F1);                    ( si ordinaire )
                                ( Répète l'opération jusqu'à F1 ou <Entrée> )
  writeln;
end.
```

Listing : TOUCHEC.C

```
/************************************************************************/
/*                          T O U C H E C                               */
/*----------------------------------------------------------------------*/
/*  Fonction: Implémente une fonction qui permet de lire un caractère */
/*            au clavier en affichant l'état des touches de bascule   */
/*            INSERT, CAPS et NUM                                     */
/*----------------------------------------------------------------------*/
/*    Auteur               : MICHAEL TISCHER                          */
/*    Développé le         : 13.08.1987                               */
/*    Dernière modification : 01.01.1992                              */
/*----------------------------------------------------------------------*/
/*    Modèle de mémoire : SMALL                                       */
/************************************************************************/

/*-- Fichiers d'inclusion ------------------------------------------------*/

#include <dos.h>
#include <bios.h>
#include <stdio.h>

/*-- Typedefs ------------------------------------------------------------*/

typedef unsigned char BYTE;             /* bricolage d'un type BYTE */

/*-- Macros --------------------------------------------------------------*/

#ifdef __TURBOC__                       /* Définitions pour TURBO C */

  #define GetKbKey()      ( bioskey( 0 ) )
  #define GetKbReady()    ( bioskey( 1 ) != 0 )
  #define GetKbEtat()     ( bioskey( 2 ) )

#else                   /* Définitions pour le compilateur Microsoft C*/

  #define GetKbKey()      ( _bios_keybrd( _KEYBRD_READ ) )
  #define GetKbReady()    ( _bios_keybrd( _KEYBRD_READY ) != 0 )
  #define GetKbEtat()     ( _bios_keybrd( _KEYBRD_SHIFTSTATUS ) )

#endif

/*-- Constantes ----------------------------------------------------------*/

/*-- Disposition des bits dans la variable d'état du clavier du BIOS -*/

#define SCRL  16                                /* Bit Scroll-Lock */
#define NUML  32                                /* Bit Num-Lock */
#define CAPL  64                                /* Bit Caps-Lock */
#define INS  128                                /* Bit Insert */

#define TRUE  ( 0 == 0 )                /* Constantes pour faciliter */
#define FALSE ( 0 == 1 )                /* la compréhension */

#define FL      0               /* Ligne d'affichage des indicateurs */
#define FC      65              /* Colonne d'affichage des indicateurs */
#define CoulIndic 0x70          /* Couleur indicateur = noir / blanc */

/*-- Code retourné par GETKEY pour quelques touches ----------------------*/
#define BEL     7                       /* Code du signal sonore */
#define BS      8                       /* Code de la touche Backspace */
#define TAB     9                       /* Code de la touche de tabulation */
#define LF      10                      /* Code de la touche LineFeed */
#define CR      13                      /* Code de la touche Entrée */
#define ESC     27                      /* Code de la touche d'échappement */

#define F1      315                     /* Touches de fonction */
#define F2      316
#define F3      317
#define F4      318
#define F5      319
#define F6      320
#define F7      321
#define F8      322
#define F9      323
#define F10     324
#define CUP     328                     /* Touches de direction */
#define CLEFT   331
#define CRIGHT  333
#define CDOWN   328

/*-- Variables globales --------------------------------------------------*/

BYTE Insert,                            /* Etat de la touche INSERT */
     Num,                               /* Etat de la touche NUM */
     Caps;                              /* Etat de la touche CAPS */

/************************************************************************/
/* GETPAGE : Lit la page d'écran courante                               */
/* Entrées : néant                                                      */
/* Sortie  : néant                                                      */
/************************************************************************/

BYTE GetPage( void )
{
  union REGS Registre;          /* Variables registres pour l'interruption */

  Registre.h.ah = 15;                       /* Numéro de la fonction */
  int86(0x10, &Registre, &Registre); /* Déclenche l'interruption 10(h) */
  return(Registre.h.bh);            /* Numéro de la page d'écran courante */
}

/************************************************************************/
/* SETPOS : Fixe la position du curseur dans la page d'écran courante */
/* Entrées : COLONNE= nouvelle colonne                                  */
/*           LIGNE  = nouvelle ligne                                    */
/* Sortie  : néant                                                      */
/* Info    : La position du curseur clignotant ne se modifie          */
/*           que si la page d'écran indiquée est la page courante     */
/************************************************************************/

void SetPos(BYTE Colonne, BYTE Ligne)
{
  union REGS Registre;          /* Variables registres pour l'interruption */

  Registre.h.ah = 2;                        /* Numéro de la fonction */
  Registre.h.bh = GetPage();                        /* page d'écran */
  Registre.h.dh = Ligne;                          /* Ligne d'écran */
  Registre.h.dl = Colonne;                      /* Colonne d'écran */
  int86(0x10, &Registre, &Registre);    /* Appel de l'interruption 10h */
}

/************************************************************************/
/* GETPOS : Lit la position du curseur dans la page d'écran courante  */
/* Entrées : néant                                                      */
/* Sortie  : Colonne = Pointeur sur la variable colonne courante      */
/*           Ligne = Pointeur sur la variable ligne courante          */
/************************************************************************/
```

```
void GetPos(BYTE * Colonne, BYTE * Ligne)
{
 union REGS Registre;       /* Variables registres pour l'interruption*/

 Registre.h.ah = 3;                          /* Numéro de la fonction */
 Registre.h.bh = GetPage();                        /* Page d'écran */
 int86(0x10, &Registre, &Registre);     /* Appelle l'interruption 10h */
 *Colonne = Registre.h.dl;        /* Lit les résultats de la fonction */
 *Ligne = Registre.h.dh;                      /* dans les registres */
}

/*****************************************************************/
/* WRITECHAR : Affiche un caractère avec un attribut donné à la  */
/*             position et dans la page courantes                */
/* Entrées : - CARACTERE = Code ASCII du caractère à afficher    */
/*           - Couleur   = Attribut du caractère                 */
/* Sortie : néant                                                */
/*****************************************************************/

void WriteChar(char Caractere, BYTE Couleur)
{
 union REGS Registre;      /* Variables registres pour l'interruption */

 Registre.h.ah = 9;                          /* Numéro de la fonction */
 Registre.h.al = Caractere;                   /* Caractère à afficher */
 Registre.h.bh = GetPage();                        /* Page d'écran */
 Registre.h.bl = Couleur;           /* Couleur du caractère à afficher */
 Registre.x.cx = 1;                          /* Un seul caractère */
 int86(0x10, &Registre, &Registre);   /* Déclenche l'interruption 10h */
}

/*****************************************************************/
/* WRITETEXT: affiche une chaine de caractère avec un attribut   */
/*            constant à partir d'une position donnée sur la page*/
/*            d'écran courante                                   */
/* Entrées : - COLONNE= colonne d'affichage                      */
/*           - LIGNE  = ligne d'affichage                        */
/*           - TEXTE  = pointeur sur la chaine à afficher        */
/*           - Couleur  = Attribut des caractères                */
/* Sortie  : néant                                               */
/* Info    : Texte est un pointeur référençant un vecteur de     */
/*           caractères qui contient le texte à afficher avec un '\0' */
/*           terminal                                            */
/*****************************************************************/

void WriteTexte(BYTE Colonne, BYTE Ligne, char *Texte, BYTE Couleur)
{
 union REGS InRegistre,    /* Variables registres pour l'interruption */
            OutRegistre;

 SetPos(Colonne, Ligne);              /* Fixe la position du curseur */
 InRegistre.h.ah = 14;                       /* Numéro de la fonction */
 InRegistre.h.bh = GetPage();                      /* Page d'écran */
 while (*Texte)                              /* Afficher jusqu'à '\0' */
  {
   WriteChar(' ', Couleur);                  /* Attribut du caractère */
   InRegistre.h.al = *Texte++;               /* Caractère à afficher */
   int86(0x10, &InRegistre, &OutRegistre); /* Déclencher interruption */
  }
}

/*****************************************************************/
/* CLS : Efface la page d'écran courante                         */
/* Entrée : néant                                                */
/* Sortie : néant                                                */
/*****************************************************************/

void Cls( void )
{
 union REGS Registre;      /* Variables registres pour l'interruption */

 Registre.h.ah = 6;                /* Numéro de la fonction Scroll-UP */
 Registre.h.al = 0;                          /* 0 pour effacer */
 Registre.h.bh = 7;                        /* noir / fond blanc */
 Registre.x.cx = 0;                      /* Coin supérieur gauche */
 Registre.h.dh = 24;                      /* Coordonnées du coin */
 Registre.h.dl = 79;                        /* inférieur droit */
 int86(0x10,&Registre,&Registre);/* Déclenche interruption BIOS vidéo */
}

/*****************************************************************/
/* NEGFLAG : Inverse un indicateur si nécessaire et affiche le texte */
/*           associé                                             */
/* Entrées : FLAG    = Dernier état de l'indicateur              */
/*           FLAGREG = Etat actuel de l'indicateur (0 = inactif) */
/*           COLONNE = Colonne d'affichage du nom de l'indicateur */
/*           LIGNE   = Ligne d'affichage du nom de l'indicateur  */
/*           TEXTE   = Nom de l'indicateur                       */
/* Sortie : nouvel état de l'indicateur(TRUE = actif,FALSE = inactif) */
/*****************************************************************/

BYTE NegFlag(BYTE Flag, unsigned int FlagReg,
     BYTE Colonne, BYTE Ligne, char * Texte)
{
 BYTE LigneCour,                                  /* Ligne courante */
      ColCour;

 if ((Flag == (FlagReg != 0)))              /* Indicateur modifié ? */
  {                                                      /* Oui */
   GetPos(&ColCour, &LigneCour);/* Lit la position courante du curseur*/
   WriteTexte(Colonne, Ligne, Texte, (BYTE) ((Flag) ? 0 : CoulIndic));
   SetPos(ColCour, LigneCour);          /* Repositionne le curseur */
   return(Flag ^1);                /* Change le bit 0 de l'indicateur */
  }
 else return(Flag);                        /* sinon tout reste pareil */
}

/*****************************************************************/
/* GETKEY : Lit un caractère et affiche l'état des indicateurs   */
/* Entrée : néant                                                */
/* Sortie : Code de la touche frappée                            */
/*                < 256 : touche ordinaire                       */
/*               >= 256 : touche à code étendu                   */
/*****************************************************************/

unsigned int GetKey( void )
{
 int Touche,                                    /* Touche retournée */
     Etat;                                       /* Etat du clavier */

 do
  {
   Etat = GetKbEtat();                        /* Lit l'état du clavier */
   Insert = NegFlag(Insert, Etat & INS, FC+9, FL, "INSERT");
   Caps = NegFlag(Caps, Etat & CAPL, FC+3, FL, " CAPS ");
   Num = NegFlag(Num, Etat & NUML, FC, FL, "NUM");
  }
 while ( !GetKbReady() );/* Recommence jusqu'à détection d'une frappe */

 Touche = GetKbKey();                                /* Lit la touche */
 return ((Touche & 255) == 0) ? (Touche >> 8) + 256 : Touche & 255;
}

/*****************************************************************/
/* INITKEY : Initialise les indicateurs des touches              */
/* Entrée : néant                                                */
/* Sortie : néant                                                */
/* Info   : Les indicateurs sont inversés par rapport à leur état */
/*          actuel pour que ce dernier puisse être affiché au    */
/*          prochain appel de GETKEY                             */
/*****************************************************************/

void IniKey( void )
{
 int Etat;                                        /* Etat du clavier */

 Etat = GetKbEtat();                          /* Lit l'état du clavier */
 Insert = (Etat & INS) ? FALSE : TRUE;       /* Inverse les contenus */
 Caps = (Etat & CAPL)? FALSE : TRUE;                    /* courants */
 Num = (Etat & NUML) ? FALSE : TRUE;
}

/*****************************************************************/
/**                        PROGRAMME PRINCIPAL                  **/
/*****************************************************************/

void main( void )
{
 unsigned int Touche;

 Cls();                                          /* Efface l'écran */
 SetPos(0,0);                          /* Curseur en haut à gauche */
 printf("TOUCHEC  -  (c) 1987, 92 by Michael Tischer\n\n");
 printf("Tapez quelques caractères en activant ou désactivant\n");
 printf("les touches INSERT, CAPS ou NUM.\n");
 printf("L'état de ces touches va être affiché à tout moment\n");
 printf("dans le coin supérieur droit de l'écran.\n");
 printf("La frappe de <Entrée> ou <F1> termine le programme...\n\n");
 printf("Votre saisie : ");

 IniKey();                    /* Initialise les indicateurs du clavier */
 do
  {
   if ((Touche = GetKey()) < 256)                 /* Lit une touche */
    printf("%c", (char) Touche);  /* Affiche la touche (si ordinaire) */
  }
 while ((Touche == CR || Touche == F1));       /* Répète l'opération */
                                                /* jusqu'à F1 ou CR */
 printf("\n");
}
```

5.2.2. Lecture des claviers étendus

Avez-vous remarqué que les touches F11 et F12 ne figuraient pas dans la liste des codes étendus présentée plus haut ? Cette absence se justifiait par l'impossibilité de lire ces touches avec ls fonctions 00h et 01h. Les concepteurs du BIOS d'IBM ont en effet mal interprété la notion de compatibilité et un raisonnement fallacieux les a conduits à ne pas prendre en compte les touches citées dans les fonctions 00h et 01h.

Sachez que le gestionnaire du clavier transfère les scan codes de ces touches dans le buffer du clavier mais que les fonctions 00h et 01h les ignorent. Elles se comportent comme si ces touches n'existaient pas, comme si le buffer était vide malgré la frappe de F11 et F12.

Les nouvelles fonctions du BIOS

Les trois nouvelles fonctions du BIOS introduites sur l'AT et qui se trouvent dans la plupart des ROM aujourd'hui n'ont pas ce comportement outrageant et tiennent compte des touches rajoutées. Depuis la version 3.3 de DOS, elles sont aussi implémentées par le driver KEYB pour peu qu'il ne les trouve pas déjà en ROM.

Pour faciliter la conversion des programmes, les nouvelles fonctions portent les numéros 10h, 11h et 12h et leur usage est semblable à celui des fonctions 00h, 01h, 02h. Seule la fonction 12h retourne une valeur différente de celle de la fonction 02h, nous en parlerons dans un instant.

Les nouveaux codes du clavier

La seule différence entre les fonctions 00h et 01h d'une part et les fonctions 10h et 11h d'autre part tient aux codes renvoyés. D'ailleurs seuls quelques codes étendus sont concernés, et la plupart du temps il s'agit de rajouts. La nouvelle version prend en compte des touches qui n'existaient pas sur les claviers de PC/XT ou d'AT ou des combinaisons nouvelles (par exemple Ctrl-TAb, Ctrl-Cursor Up, ou Alt-Echappement).

Les codes qui ont été modifiés sont surtout les codes des touches de direction grises, ceci pour pouvoir les distinguer de leurs homologues du pavé numérique. Il ne retournent plus comme code ASCII (en AL) la valeur 00h mais E0h. Si vous exploitez les fonctions 10h et 11h du BIOS, vos programmes devront considérer que les codes étendus ne s'expriment plus seulement par un code ASCII 00h mais aussi par E0h.

S'il ne s'agit que de reconnaître les touches de fonction F11/F12 sans opérer la distinction entre touches de directions grises et blanches, vous pouvez vous épargner bien du travail en convertissant dès réception le code ASCII E0h en 00h. Ensuite vous traiterez de la même façon les touches de direction blanches et grises.

Le tableau suivant indique les touches ou combinaisons de touches qui ne renvoient plus le même code avec les nouvelles fonctions 10h/11h

Combinaisons de touches étendues renvoyées par les fonctions 10h/11h du BIOS								
Touches de fonction			**par Shift**		**par Ctrl**		**par Alt**	
	Anc. AH/AL	Nouv. AH/AL	Anc. AH/AL	Nouv. AH/AL	Anc. AH/AL	Nouv. AH/AL	Anc. AH/AL	Nouv. AH/AL
F11	---	85/00	---	87/00	---	89/00	---	8B/00
F12	---	86/00	---	88/00	---	8A/00	---	8C/00
Touches de direction grises dans le bloc séparé			**par Shift**		**par Ctrl**		**par Alt**	
	Anc. AH/AL	Nouv. AH/AL	Anc. AH/AL	Nouv. AH/AL	Anc. AH/AL	Nouv. AH/AL	Anc. AH/AL	Nouv. AH/AL
Home	47/00	47/E0	47/00	47/E0	77/00	77/E0	---	97/00
Curseur haut	48/00	48/E0	48/00	48/E0	---	8D/E0	---	98/00
Page Up	49/00	49/E0	49/00	49/E0	84/00	84/E0	---	99/00
Curseur gauche	4B/00	4B/E0	4B/00	4B/E0	73/00	73/E0	---	9B/00
Curseur droit	4D/00	4D/E0	4D/00	4D/E0	74/00	74/E0	---	9D/00
Fin	4F/00	4F/E0	4F/00	4F/E0	75/00	75/E0	---	9F/00
Curseur bas	50/00	50/E0	50/00	50/E0	---	91/E0	---	A0/00
Page Down	51/00	51/E0	51/00	51/E0	76/00	76/E0	---	A1/00
Insert	52/00	52/E0	52/00	52/E0	---	92/E0	---	A2/00
Delete	53/00	53/E0	53/00	53/E0	---	93/E0	---	A3/00
Autres touches grises			**par Shift**		**par Ctrl**		**par Alt**	
	Anc. AH/AL	Nouv. AH/AL	Anc. AH/AL	Nouv. AH/AL	Anc. AH/AL	Nouv. AH/AL	Anc. AH/AL	Nouv. AH/AL
/	35/2F	E0/2F	---	E0/2F	---	95/00	---	A4/00
*	37/2A	37/2A	---	37/2A	---	96/00	---	37/00
-	4A/2D	4A/2D	---	4A/2D	---	8E/00	---	4A/00
+	4E/2B	4E/2B	---	4E/2B	---	90/00	---	4E/00
Entrée	1C/0D	E0/0D	---	E0/0D	---	E0/0A	---	A6/00

Combinaisons de touches étendues renvoyées par les fonctions 10h/11h du BIOS

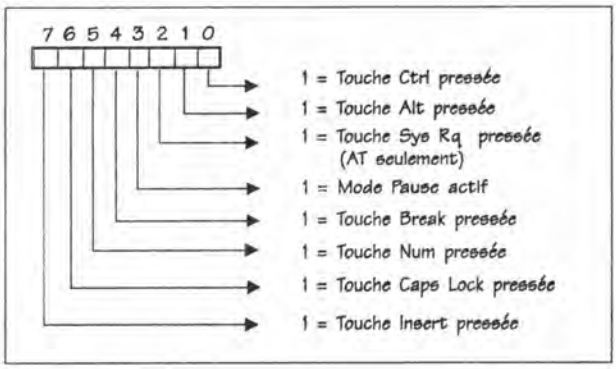
Combinaisons supplémentaires des touches blanches			par Shift		par Ctrl		par Alt	
	Anc. AH/AL	Nouv. AH/AL	Anc. AH/AL	Nouv. AH/AL	Anc. AH/AL	Nouv. AH/AL	Anc. AH/AL	Nouv. AH/AL
Tab					---	94/00	---	A5/00
5 sur pavé numérique					---	8F/00		
Curseur haut blanc					---	8D/00		
Curseur bas blanc					---	91/00		
Insert blanc					---	92/00		
Delete blanc					---	93/00		
Escape							---	01/00
Backspace							---	0E/00
Tab							---	A5/00
[(dt. ü)							---	1A/00
] (dt. *+)							---	1B/00
Enter							---	1C/00
; (dt. ö)							---	27/00
' (dt. ä)							---	28/00
‘ (dt. ^º)							---	29/00
\ (dt. #')							---	2B/00
,					---	33/00		
.							---	34/00
/ (dt. -_)							---	35/00

(codes exprimés en système hexadécimal)

La nouvelle fonction de lecture de l'état du clavier

Alors que les fonctions 10h et 11h sont identiques à leurs prédécesseurs, la fonction 12h est quelque peu différente de l'ancienne fonction 02h. En plus de l'indicateur d'état mémorisé dans AL, la fonction renvoie une information supplémentaire en AH. Il s'agit de l'indicateur d'état étendu, une variable du BIOS qui n'est gérée qu'avec les claviers étendus. Un champ de bits y indique l'état présent des différentes touches de verrouil-

lage, ainsi que l'état des touches Alt et Ctrl droites et gauches (que ne distingue pas l'indicateur d'état standard ordinaire).



Les fonctions sont-elles disponibles ?

Comme il a été mentionné précédemment, les fonctions étendues de gestion du clavier par le BIOS sont fournies par la plupart des fabricants de BIOS ainsi que par certains drivers comme KEYB. Mais on ne peut pas toujours être sûr qu'elles sont présentes.

Il est prudent de disposer au début des programmes qui désirent les exploiter un test de disponibilité. Mais ce test n'est pas si simple à inventer car le BIOS ne donne pas de fonction appropriée.

Il faut donc employer une petite astuce. Nous appellerons la fonction 12h en disposant le numéro de cette fonction (12h) dans AH et détail très important le nombre 0 dans AL. Si à l'issue de l'appel le registre AX contient toujours 1200h, c'est que la fonction 12h n'est pas en service. Il existe en effet une règle intangible du BIOS qui veut que lorqu'une fonction inconnue est appelée, le contrôle est rendu sans que le registre AX ait subi la moindre modification. Notez cependant que cette règle ne s'applique pas à l'indicateur de retenue, sinon la tâche du programmeur aurait été encore plus facile.

Lorsque la fonction 12h est implémentée, elle ne peut pas renvoyer la valeur 1200h en Ax. Car si on examine les deux indicateurs d'état du clavier, on voit que la valeur 12h de l'indicateur étendu est incompatible avec la valeur 00h de l'indicateur standard. D'après l'indicateur étendu, il faudrait que l'utilisateur ait appuyé sur la touche d'arrêt du défilement et la touche Majuscule gauche. Mais alors les 3ème et 4ème bits de l'indicateur standard devraient être à 1, ce qui dénote une valeur minimale de 24, mais sûrement pas 0.

Le programme suivant montre comment effectuer concrètement ce test.

Exemples de programmes

L'usage des fonctions étendues du BIOS est illustré par trois programmes écrits en Basic, Pascal et C et appelés : CLETENDB.BAS, CLETENDP.PAS et CLETENDC.C.

Ils sont relativement simples et se contentent de lire le clavier par la fonction 10h puis d'afficher le code ASCII et le scan code trouvés. Les affichages se font dans le système hexadécimal pour que vous puissiez les confronter au tableau précédent.

Le but de ce programme est en effet de vous montrer les codes supplémentaires rendus disponibles par le clavier étendu. Le programme se termine en actionnant la touche d'échappement.

Vous pouvez entreprendre une comparaison avec les fonctions ordinaires du BIOS en notant les codes renvoyées par une sélection de touches, puis en imposant au programme à l'usage de l'ancienne fonction 00h et en frappant une nouvelle fois les mêmes touches. Vous devrez simplement modifier dans GetMFKey l'instruction de chargement du registre AX.

La lecture du clavier ne commence qu'à partir du moment où la fonction TestMF a décelé la présence d'un clavier MF. C'est l'occasion de rencontrer à nouveau le test décrit précédemment qui exploite la fonction 12h étendue.

Listing : CLETENDB.BAS

```
'*****************************************************************
'*                        C E T E N D B                          *
'*---------------------------------------------------------------*
'*  Montre comment lire les touches additionnelles d'un clavier étendu *
'*---------------------------------------------------------------*
'* Auteur                : MICHAEL TISCHER                       *
'* Développé le          : 01.01.1992                            *
'* Dernière modification : 01.01.1992                            *
'*****************************************************************
'
'$INCLUDE: 'QB.BI'       'Fichier contenant les déclarations de registres
'
DECLARE FUNCTION MakeWord! (Nombre AS INTEGER)
DECLARE FUNCTION HexByte$ (valeur AS INTEGER)
DECLARE FUNCTION GetCEKey% ()
DECLARE FUNCTION TestCE% ()
'
CONST TRUE = -1                 'Définition des constantes booléennes
CONST FALSE = NOT TRUE
'
'-- Programme principal -------------------------------------------
'
DIM touche AS INTEGER
DIM ENTREE AS STRING
'
CLS
ENTREE = CHR$(13)
PRINT "CLETENDB  -  (c) 1992 by Michael Tischer": ENTREE
IF TestCE THEN
  PRINT "Les extensions du BIOS pour claviers étendu";
  PRINT "ont été décelées! " + ENTREE + ENTREE
  PRINT "Actionnez les touches ou les combinaisons de"
  PRINT "touches dont vous voulez connaître les codes." + ENTREE
  PRINT "Pour sortir du programme, tapez <Esc> " + ENTREE
'
  DO                                            'Boucle de saisie
    touche = GetCEKey                             'Lit la touche
    PRINT "Scan : "; HexByte(MakeWord((touche) / 256); "  ";
    PRINT "ASCII: "; HexByte(touche AND 255);
    IF ((touche AND 255) = &HE0) AND ((touche / 256) <> 0) THEN
      PRINT "  <----Touche étendue"
    ELSE
      PRINT
    END IF
  LOOP UNTIL (touche = &H11B)                   ' Répète jusqu'à ESCAPE
  PRINT ENTREE
ELSE
  PRINT "Il n'y a pas d'extension du BIOS pour clavier étendue !"
END IF
END
'
'*****************************************************************
'* GetCEKey  : Lit une touche avec la fonction étendue &H10      *
'* Entrée : néant                                                *
'* Sortie : code de la touche frappée                            *
'*****************************************************************
'
FUNCTION GetCEKey%
'
DIM reg AS RegType          'Registres du processeur pour l'interruption
reg.ax = &H1000           'Fonction de lecture étendue pour clavier étendu
CALL INTERRUPT(&H16, reg, reg)          'Déclenche l'interruption du BIOS
GetCEKey% = reg.ax                        'Retourne le code de la touche
'
END FUNCTION
'
'*****************************************************************
'* HexByte : Convertit un octet en nombre hexadécimal à deux chiffres *
'* Entrée : VALEUR = octet à convertir                           *
'* Sortie : nombre hexadécimal sous forme de chaîne              *
'*****************************************************************
'
FUNCTION HexByte$ (valeur AS INTEGER)
'
```

```
IF valeur < 16 THEN                              'un seul chiffre ?
  HexByte$ = "0" + HEX$(valeur)          'Oui, un 0 en première position
ELSE                                             'Non, on a deux chiffres
  HexByte$ = HEX$(valeur)
END IF

END FUNCTION

'*************************************************************************
'* Makeword : convertit un nombre entier en entier long                  *
'*            En basic, les décalages de bits qui réalisent les          *
'*            divisions entières donnent des résultats erronés pour      *
'*            les nombres négatifs                                       *
'* Entrée   : nombre entier à convertir                                  *
'* Sortie   : nombre entier long                                         *
'*************************************************************************
'
FUNCTION MakeWord& (Nombre AS INTEGER)

IF Nombre < 0 THEN
  MakeWord = 65536! + Nombre
ELSE
  MakeWord = Nombre
END IF

END FUNCTION

'*************************************************************************
'* TestCE : Teste si les fonctions étendues du BIOS                      *
'* pour lire un clavier étendu sont disponibles                          *
'* Entrée : néant                                                        *
'* Sortie : TRUE, si les fonctions sont disponibles , sinon FALSE        *
'*************************************************************************
'
FUNCTION TestCE%

DIM reg AS RegType          'Registres du processeur pour l'interruption

reg.ax = &H1200             'Fonction d'état étendue pour clavier étendu
CALL INTERRUPT(&H16, reg, reg)          'Déclenche l'interruption du BIOS
PRINT HEX$(reg.ax)
TestCE% = (reg.ax <> &H1200)                'AX = &h1200 : Fonction absente

END FUNCTION
```

Listing : CLETENDP.PAS

```
{*************************************************************************}
{*                          C L E T E N D P                              *}
{*-----------------------------------------------------------------------*}
{*  Montre comment lire les touches additionnelles d'un clavier          *}
{*  étendu                                                               *}
{*-----------------------------------------------------------------------*}
{*  Auteur              : MICHAEL TISCHER                                *}
{*  Développé           :  01.01.1992                                    *}
{*  Dernière modification :  01.01.1992                                  *}
{*************************************************************************}

program cletend;

uses Dos, Crt;

const Entree = #13#10;                              { Nouvelle ligne }

{*************************************************************************}
{* HexByte : Convertit un octet en nombre hexadécimal à 2 chiffres       *}
{* Entrée : VALEUR = octet à convertir                                   *}
{* Sortie : chaine hexadécimale à 2 chiffres                             *}
{*************************************************************************}

function HexByte( valeur : byte ) : string;

const HexDigits : array [0..15] of char = '0123456789ABCDEF';

var dummy : string[2];                              { variable de travail }

begin
  dummy[0] := chr(2);                           { a priori deux chiffres }
  dummy[1] := HexDigits[ valeur shr 4 ]; { Convertit les deux quartets }
  dummy[2] := HexDigits[ valeur and $0F ];                  { en Hex }
  HexByte := dummy;
  HexByte := dummy;
end;

{*************************************************************************}
{* TestCE : Teste si les fonctions étendues du BIOS pour lire un         *}
{*          étendu sont disponibls                                       *}
{* Entrée : néant                                                        *}
{* Sortie : TRUE, si les fonctions sont disponibles, sinon FALSE         *}
{*************************************************************************}

function TestCE : boolean;

var Regs : Registers;       { pour le déclenchement de l'interruption }

begin
  Regs.AX := $1200; { Fonction d'état étendue pour les claviers étendu }
  Intr( $16, Regs );
  TestCE := ( Regs.AX <> $1200 );       { AX=$1200 : Fonction absente }
end;

{*************************************************************************}
{* GetCEKey : Lit une touche avec la fonction étendue $10                *}
{* Entrée : néant                                                        *}
{* Sortie : code de la touche frappée                                    *}
{*************************************************************************}

function GetCEKey : word;

var Regs : Registers;                           { pour l'interruption }

begin
  Regs.AH := $10;          { Fonction de lecture étendue pour clavier }
  Intr( $16, Regs );
  GetCEKey := Regs.AX;                  { Renvoie le code de la touche }
end;

{*************************************************************************}
{*                     PROGRAMME PRINCIPAL                               *}
{*************************************************************************}

var Touche : word;

begin
  clrscr;
  writeln( 'CLETENDP  -  (c) 1992 by Michael Tischer' + Entree );
  if ( TestCE ) then
    begin
      writeln( 'Les extensions du BIOS pour claviers étendu'+
      ' ont été décalées!'+Entree+
      'Actionnez les touches ou les combinaisons de touches'+Entree+
      'dont vous voulez connaître les codes'+Entree+Entree
      'Pour sortir du programme tapez <Esc>'+Entree );

      repeat                                        { Boucle de saisie }
        Touche := GetCEKey;                          { Lit une touche }
        write( 'Scan : ', HexByte(hi(Touche)), '  ',
               'ASCII: ', HexByte(lo(Touche)) );
        if ( (lo(Touche) = $ED) and (hi(Touche) <> 0 ) ) then
          write( '  <---- Touche étendu' );
        writeln;
      until ( Touche = $011b );     { Répéter l'opération jusqu'à ESCAPE }
      writeln( Entree );
    end
  else
    writeln( 'Il n''y a pas d''extension du BIOS pour clavier étendu !');
end.
```

Listing : CLETENDC.C

```
/*****************************************************************/
/*                        C L E T E N D C                        */
/*---------------------------------------------------------------*/
/*   Montre comment lire les touches additionnelles d'un clavier */
/*   étendu                                                      */
/*---------------------------------------------------------------*/
/*   Autor          : MICHAEL TISCHER                            */
/*   entwickelt am  :  1.01.1992                                 */
/*   letztes Update :  1.01.1992                                 */
/*****************************************************************/

/*-- Fichiers d'inclusion ---------------------------------------*/

#include <stdio.h>
#include <dos.h>

/*-- Typedefs ---------------------------------------------------*/

typedef unsigned char BYTE;             /* Bricolage d'un type BYTE */
typedef unsigned int WORD;

/*-- Constantes -------------------------------------------------*/

#define TRUE  ( 0 == 0 ) /* Constantes pour faciliter la compréhension*/
#define FALSE ( 0 == 1 )

/*-- Routines d'écran pour Microsoft C --------------------------*/

#ifndef __TURBOC__                                  /* Microsoft C? */

  /*************************************************************/
  /* Gotoxy        : Positionne le curseur                     */
  /* Entrée        : Coordonnées du curseur                    */
  /* Sortie        : néant                                     */
  /*************************************************************/

  void gotoxy( int x, int y )
  {
   union REGS Register;                     /* Pour les interruptions */
   regs.h.ah = 0x02;                        /* Numéro de la fonction  */
   regs.h.bh = 0;                                        /* Couleur */
   regs.h.dh = y - 1;
   regs.h.dl = x - 1;
   int86( 0x10, &regs, &regs );           /* Déclenche l'interruption */
  }

  /*************************************************************/
  /* clrscr        : Efface l'écran                            */
  /* Entrée        : néant                                     */
  /* Sortie        : néant                                     */
  /*************************************************************/

  void clrscr( void )
  {
   union REGS regs;                              /* Pour l'interruption */

   regs.h.ah = 0x07;                         /* Numéro de la fonction */
   regs.h.al = 0x00;
   regs.h.ch = 0;
   regs.h.cl = 0;
   regs.h.dh = 24;
   regs.h.dl = 79;
   int86( 0x10, &regs, &regs );           /* Déclenche l'interruption */
   gotoxy( 1, 1 );                                /* Place le curseur */
  }

#endif

/*****************************************************************/
/* HexByte : Convertit un octet en nombre hexadécimal            */
/* Entrée  : VALEUR = octet à convertir                          */
/* Sortie  : chaîne hexadécimale à deux chiffres                 */
/*****************************************************************/

char *HexByte( BYTE wert )
{
 char HexDigits[16] = "0123456789ABCDEF";
 static char dummy[3] = "00";

 dummy[0] = HexDigits[ wert >> 4 ];              /* Transforme les deux */
 dummy[1] = HexDigits[ wert & 0x0F ];              /* quartets en Hex */
 return dummy;
}

/*****************************************************************/
/* TestCE : Teste si les fonctions étendues du BIOS pour lire    */
/*          un clavier étendu sont disponibles                   */
/* Entrée : néant                                                */
/* Sortie : TRUE, si les fonctions sont disponibles , sinon FALSE*/
/*****************************************************************/

int TestCE( void )
{
 union REGS regs;                                /* Pour l'interruption */

 regs.x.ax = 0x1200;   /* Fonction d'état étendue pour clavier étendu */
 int86( 0x16, &regs, &regs );
 return ( regs.x.ax != 0x1200 );       /* AX=0x1200 : Fonction absente */
}

/*****************************************************************/
/* GetCEKey : Lit une touche avec la fonction étendue 0x10       */
/* Entrée   : néant                                              */
/* Sortie   : code de la touche frappée                          */
/*****************************************************************/

WORD GetCEKey( void )
{
 union REGS regs;                                /* pour l'interruption */

 regs.h.ah = 0x10; /* Fonction de lecture étendue pour clavier étendu */
 int86( 0x16, &regs, &regs );
 return regs.x.ax;                          /* renvoie le code de la touche */
}

/*****************************************************************/
/*                      PROGRAMME PRINCIPAL                      */
/*****************************************************************/

void main( void )
{
 WORD touche;

 clrscr();
 printf( "CLETENDC  -  (c) 1992 by Michael Tischer\n\n" );
 if ( TestCE() )
 {
  printf("Les extensions du BIOS pour clavier étendu ont été décelées"\
   "\n\nActionnez les touches ou les combinaisons de touches dont"
   "\nvous voulez connaître les codes."
   "\n\nPour sortir du programme, tapez <Esc>\n\n");

  do                                                /* Boucle de saisie */
  {
   touche = GetCEKey();                              /* Lit une touche */
   printf( "Scan : %s ", HexByte((BYTE) (touche >> 8)) );
   printf( "ASCII: %s", HexByte((BYTE) (touche & 255)) );
   if ( ((touche & 255) == 0xe0) && ((touche & 65280 ) != 0 ) )
    printf( " <---- Touche étendue" );
   printf( "\n" );
  }
  while ( touche != 0x011b );                  /* Répète jusqu'à ESCAPE */
  printf( "\n\n" );
 }
 else
  printf( "Il n'y a pas d'exentsion du BIOS pour clavier étendu !");
}
```

5.2.3. Les variables de l'interruption clavier du BIOS

Pour gérer le clavier et assurer la communication entre le gestionnaire d'interruption du clavier (Int 09h) et les fonctions clavier du BIOS (Int 16h), le BIOS exploite dans un segment particulier huit variables données par le tableau suivant. La connaissance de ces variables sera surtout profitable aux programmeurs désireux de développer des programmes résidents et de modifier les deux interruptions concernées. Mais les programmes ordinaires peuvent aussi tirer profit de la manipulation directe de ces variables comme nous le verrons à l'issue de cette section.

Variables du BIOS pour gérer le clavier

Offset	Signification	Type
17h	Indicateur d'état du clavier	1 BYTE
18h	Indicateur d'état étendu	1 BYTE
19h	Réception d'un code ASCII	1 BYTE
1Ah	Caractère suivant dans le buffer	1 WORD
1Ch	Dernier caractère dans le buffer	1 WORD
1Eh	Buffer	16 WORD
80h	Adresse de début du buffer	1 WORD
82h	Adresse de fin du buffer	1 WORD

Deux de ces variables, les indicateurs d'état du clavier, vous sont déjà connues : ce sont les deux octets renvoyés par les fonctions 02h et 12h du BIOS.

Un peu plus loin, à l'adresse 19h, se trouve un octet qui est exploité lorsqu'on introduit un code ASCII à l'aide de la touche ALT combinée avec le pavé numérique. Chaque fois que l'on tape ainsi un chiffre avec ALT, le code ASCII réceptionné est mis à jour à cet endroit.

Gestion du buffer du clavier

Les trois variables situées aux offsets 1Ah, 1Ch et 1Eh servent à gérer le buffer dans lequel le gestionnaire de l'interruption 09h mémorise les touches frappées pour les mettre à la disposition du programme d'application qui peut les lire par l'intermédiaire de l'interruption 16h du BIOS.

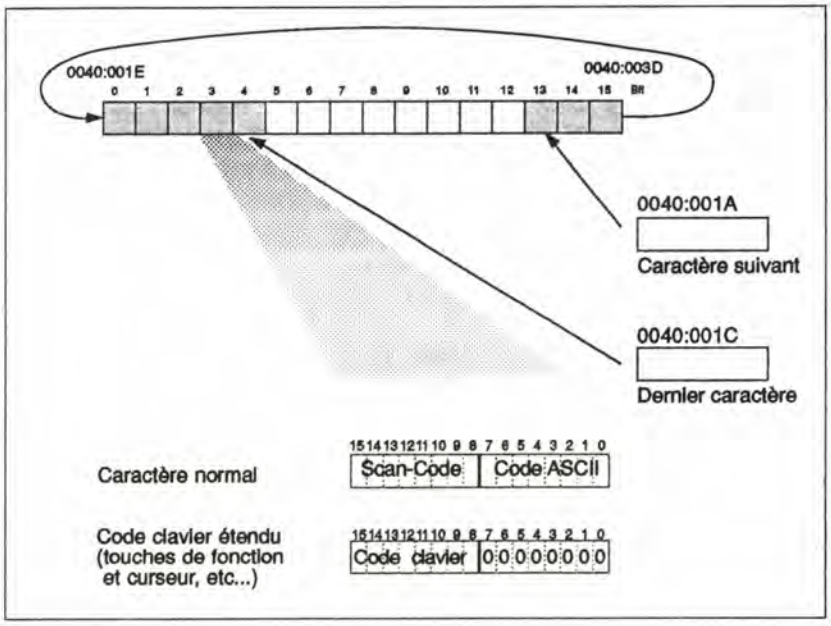
Pour comprendre la signification des deux premières variables, il faut savoir que le buffer est refermé sur lui-même à la façon d'un anneau. Cette technique de file d'attente circulaire est souvent utilisée pour gérer des événements asynchrones, c'est-à-dire survenant à n'importe quel moment, ici il s'agit de l'arrivée des caractères tapés.

A première vue il semblerait normal de disposer les caractères dans l'ordre de leur arrivée, à partir du début du buffer. Le premier caractère se situerait en première position, le deuxième caractère se mettrait derrière en deuxième position et ainsi de suite. Le prélèvement d'un caractère se ferait toujours en première position, après quoi on décalerait tous les caractères d'une position pour que le caractère transmis ne le soit pas une deuxième fois. Ce mécanisme fonctionne correctement mais le décalage des caractères est trop coûteux en temps d'exécution. Il est préférable de travailler avec deux pointeurs. Le premier pointeur indique l'endroit où doit être prélevé le prochain caractères émis, tandis que le deuxième marque la position de réception du prochain caractère reçu.

A chaque lecture ou insertion de caractères, il n'est plus nécessaire de décaler le buffer, il suffit d'ajuster le pointeur concerné. Au début les deux pointeurs référencent le début du buffer mais au fur et à mesure des lectures et des insertions, ils se déplacent vers la fin du buffer. S'ils dépassent cette limite, ils sont repositionnés au début : le buffer est donc bien circulaire.

Très concrètement, le pointeur mémorisé à l'offset 1Ah indique l'adresse de lecture du prochain caractère. Lorsque la lecture est effectuée, ce pointeur se déplace de deux octets vers la fin du buffer : chaque caractère occupe en effet deux octets (code ASCII + scan code). Lorque le contenu de la dernière mémoire du buffer est ainsi prélevé, le pointeur pointe à nouveau sur le début du buffer.

Le pointeur en mémoire 1Ch se comporte de la même façon. Il référence le dernier caractère du buffer. Si l'utilisateur appuie sur une touche, le code tapé est mémorisé à l'endroit référencé. Ensuite il est incrémenté de deux octets en direction de la fin du fichier. Si un nouveau caractère est stocké dans le dernier mot du buffer, le pointeur 1Ch est également redirigé sur le début du fichier.



Le rapport entre le pointeur de début et le pointeur de fin montre l'état du buffer. Ainsi lorsque les pointeurs situés aux offsets 1Ah et 1Ch sont identiques, le buffer est vide. Lorsqu'à l'arrivée d'un caractère, le pointeur incrémenté référence le début du fichier, c'est que le buffer est plein et ne peut plus réceptionner quoi que ce soit.

Le buffer comme file d'attente circulaire

La mémoire commençant à l'offset 1Eh est occupée par le buffer proprement dit. Chaque caractère y prend deux octets, donc avec ses 32 octets il peut contenir 16 caractères.

Pour un caractère ASCII ordinaire c'est d'abord le code ASCII qui est mémorisé puis le scan code. Lorsque le code est étendu, la partie ASCII est 0 car elle ne se trouve réellement que dans l'octet suivant.

Même si les adresses et la signification des deux dernières variables du BIOS laissent penser que le buffer puisse être décalé , il se trouve toujours en 1Eh. Il est vrai que les développeurs avaient effectivement imaginé cette possibilité pour accroître sa taille. Mais elle n'a jamais été exploitée car les deux variables en question ont été introduites avec l'AT et son BIOS modifié, et à cette époque-là il existait déjà un grand nombre de programmes résidents qui ne pensaient pas à chercher le buffer ailleurs qu'à l'offset 1Eh. L'idée finalement n'était pas mauvaise mais elle ne fut jamais convertie en actes.

Exemples de programmes

Pour démontrer quelques manipulations de variables du BIOS, je voudrais vous présenter trois programmes écrits en Basic, Pascal et C qui se chargent de réinitialiser le buffer du clavier, c'est-à-dire d'effacer son contenu. Cette opération est souhaitable dans certaines circonstances spéciales, lorsque par exemple l'utilisateur doit répondre à une question critique du genre "Voulez-vous effacer le fichier, ou formater le disque dur ?". Il faut alors éviter qu'une frappe prématurée ne provoque un désastre.

Une telle fonction n'existe pas dans le BIOS, c'est pourquoi il faut mettre soi-même la main à la pâte. La programmation n'est pas difficile car le buffer circulaire étant géré par deux pointeurs, il suffit de remanier ces derniers en les rendants égaux. Le BIOS croit alors que le buffer est vide. Pendant cette manipulation il est essentiel d'inhiber les interruptions pour que des caractères n'arrivent pas subrepticement dans l'intervalle.

Les programmes NOKEYP.PAS (en Pascal) et NOKEY.C (en C) appliquent concrètement ces idées. Ils commencent par pratiquer une sorte de compte à rebours sur l'écran pour vous donner le temps de taper quelques caractères. Ce n'est qu'après que le buffer du clavier est effacé. Ensuite des appels aux fonctions 000 et 01h du BIOS permettent d'afficher tous les caractères qui restent présents dans le buffer : vous vérifierez qu'il n'en reste aucun.

Listing : NOKEYP.PAS

```
{*********************************************************************}
{*                          N O K E Y P                              *}
{*-------------------------------------------------------------------*}
{*   Montre comment effacer le buffer du clavier                     *}
{*   pour protéger l'utilisateur contre des saisies résiduelles      *}
{*   lorsqu'il doit répondre à des questions                         *}
{*   importantes (par ex "Voulez-vous supprimer tel fichier ?")      *}
{*-------------------------------------------------------------------*}
{*   Auteur              : MICHAEL TISCHER                           *}
{*   Développé le        :  01.01.1992                               *}
{*   Dernière modification :  01.01.1992                             *}
{*********************************************************************}

program NoKeyP;

uses Crt;

{*********************************************************************}
{* ClearKbBuffer : efface le contenu du buffer du clavier            *}
{* Entrée : néant                                                    *}
{* Sortie : néant                                                    *}
{*********************************************************************}

procedure ClearKbBuffer;

begin
  inline( $fa );              { CLI, inhibe les interruptions matérielles }
  memw[$40:$1A] := memw[$40:$1C];   { Plus de caractère dans le buffer }
  inline( $fb );              { STI, rétablit les interruptions matérielles }
end;

{*********************************************************************}
{*                      Programme principal                          *}
{*********************************************************************}

var i,                                    { Compteur d'itérations }
    ccount : integer;                     { Nombre de caractères dans le buffer }
    ch     : char;                        { mémorise les touches }

begin
  clrscr;
  writeln( 'NOKEYP  -  (c) 1992 by Michael Tischer' );
  writeln;
  writeln( 'A 0 les caractères du buffer vont être effacés .' );
  writeln;

  for i := 10 downto 0 do    { Laisse le temps de taper des caractères }
    begin
      write( i:5 );
      delay( 750 );
    end;

  ClearKbBuffer;                          { Vide le buffer du clavier }

  {-- Affiche le nombre de caractères qui restent dans le buffer ------}

  ccount := 0;                            { Pas encore de caractère }
  writeln;
  writeln;
  writeln( 'Caractères dans le buffer :' );

  while KeyPressed do       { Reste-t-il un caractère dans le buffer ? }
    begin                          { Oui, lit le caractère et l'affiche }
      ch := ReadKey;
      write( '   ', ord(ch):5 );            { Affiche d'abord le code }
      if ord(ch) > 32 then                    { caractère spécial ? }
        write ( '(', ch, ')' );          { Non, affiche le caractère  }
      writeln;
      inc( ccount );
    end;
  if ccount = 0 then                              { Pas de caractère ? }
    writeln( '(Aucun)' );                                    { Non }
  writeln;
end.
```

Listing : NOKEYC.C

```
/*****************************************************************/
/*                          N O K E Y C                          */
/*---------------------------------------------------------------*/
/*   Montre comment effacer le buffer du clavier pour protéger   */
/*   l'utilisateur contre des saisies résiduelles lorsqu'il      */
/*   doit répondre à des questions importantes                   */
/*    (par ex "Voulez-vous supprimer tel fichier ?")             */
/*---------------------------------------------------------------*/
/*   Auteur                : MICHAEL TISCHER                     */
/*   Développé le          : 01.01.1992                          */
/*   Dernière modification : 01.01.1992                          */
/*****************************************************************/

#include <stdio.h>
#include <dos.h>
#include <bios.h>

/*--- Macros ------------------------------------------------------*/

#ifndef MK_FP           /* Définit la macro MK_FP si elle n'existe pas */
  #define MK_FP(seg,ofs) \
    ((void far *) (((unsigned long)(seg) << 16) | (unsigned)(ofs)))
#endif

#ifdef __TURBOC__                       /* Définitions pour TURBO C */

  #define GetKbKey()        ( bioskey( 0 ) )
  #define GetKbReady()      ( bioskey( 1 ) != 0 )
  #define GetBiosTime(x)    ( x = biostime( 0, NULL ) )
  #define CLI()             ( disable() )
  #define STI()             ( enable() )

#else                   /* Définitions pour le compilateur  Microsoft C */

  #define GetKbKey()        ( _bios_keybrd( _KEYBRD_READ ) )
  #define GetKbReady()      ( _bios_keybrd( _KEYBRD_READY ) != 0 )
  #define GetBiosTime(x)    ( _bios_timeofday( _TIME_GETCLOCK, &x) )
  #define CLI()             ( _disable() )
  #define STI()             ( _enable() )

#endif

/*--- Routines d'affichage pour Microsoft C -----------------------*/

#ifndef __TURBOC__                                  /* Microsoft C? */

  /*************************************************************/
  /* Gotoxy       : Positionne le curseur                      */
  /* Entrées      : Coordonnées du curseur                     */
  /* Sortie       : néant                                      */
  /*************************************************************/

  void gotoxy( int x, int y )
  {
    union REGS regs;                    /* Registres pour l'interruption */

    regs.h.ah = 0x02;                           /* Numéro de la fonction */
    regs.h.bh = 0;                                      /* Couleur */
    regs.h.dh = y - 1;
    regs.h.dl = x - 1;
    int86( 0x10, &regs, &regs );                /* Interruption */
  }

  /*************************************************************/
  /* clrscr       : Efface l'écran                             */
  /* Entrée       : néant                                      */
  /* Sortie       : néant                                      */
  /*************************************************************/

  void clrscr( void )
  {
    union REGS regs;                    /* Registres pour l'interruption */
    regs.h.ah = 0x07;                           /* Numéro de la fonction */
    regs.h.al = 0x00;
    regs.h.ch = 0;
    regs.h.cl = 0;
    regs.h.dh = 24;
    regs.h.dl = 79;
    int86( 0x10, &regs, &regs );                /* Interruption */
    gotoxy( 1, 1 );                             /* Place le curseur */
  }

#endif

/*****************************************************************/
/* Delay :  Fige l'exécution du programme pendant un certain temps */
/*          indépendamment de la cadence du système              */
/* Entrée  : PAUSE = temps d'arrêt en tops d'horloge             */
/* Sortie  : néant                                               */
/* Info    : un top = 1/18,2 Secondes                            */
/*****************************************************************/

void delay( unsigned int pause )
{
  long temps,                                       /* temps courant */
       tempsfinal;                                  /* temps final */

  if ( pause )                                  /* Pause non nulle ? */
  {                                                     /* Non */
    GetBiosTime( tempsfinal );
    tempsfinal += (long) pause;             /* Calcule le temps final */

    do                                      /* Boucle, lit le temps courant */
      GetBiosTime( temps );
    while ( temps <= tempsfinal );          /* temps final atteint ? */
  }                                                     /* Oui, terminé */
}

/*****************************************************************/
/* ClearKbBuffer : Efface le contenu du buffer du clavier        */
/* Entrée  : néant                                               */
/* Sortie  : néant                                               */
/*****************************************************************/

void ClearKbBuffer( void )
{
  CLI();                                /* Inhibe les interruptions matérielles */
  *(int far *) MK_FP(0x40,0x1a) =  /* Plus de caractère dans le buffer */
  *(int far *) MK_FP(0x40,0x1C);
  STI( );                               /* Rétablit les interruptions matérielles */
}

/*****************************************************************/
/*                      Programme principal                      */
/*****************************************************************/

void main( void )
{
  int           i,                                  /* Compteur d'itérations */
        ccount;         /* Nombre de caractères dans le buffer */
  unsigned char ch;     /* Mémorise les touches */

  clrscr();
  printf( "NOKEYC  -  (c) 1992 by Michael Tischer\n\n" );
  printf( "A 0 les caractères du buffer vont être effacés \n\n" );

  for ( i = 10; i; --i )  /* Laisse le temps de saisir des caractères */
  {
    printf( "%5d", i );
    delay( 13 );                                    /* Pause de 3/4 de sec */
  }

  ClearKbBuffer();                          /* Vide le buffer du clavier */

  /*-- Efface les caractères qui restent dans le buffer du clavier ----*/

  ccount = 0;                   /* Initialise le compteur de caractères */
  printf( "\n\nCaractères dans le buffer :\n" );

  while GetKbReady()        /* Reste-t-il un caractère dans le buffer ? */
  {                                 /* Oui, lit le caractère et l'affiche */
    ch = GetKbKey();
    printf( "   %3d   ", (int) ch );        /* Affiche d'abord le code */
    if ( (int) ch > 32 )                        /* Caractère spécial ? */
      printf ( "(%c)", ch );                /* Non, affiche le caractère */
    printf("\n");
    ++ccount;
  }

  if ( ccount == 0 )                            /* Pas de caractère ? */
    printf( "(Aucun)\n" );                                  /* Non */
  printf( "\n" );
}
```

La version BASIC

La version écrite en Basic adopte une autre stratégie car ce langage ne permet pas d'inhiber simplement les interruptions. Le processus consiste à installer une boucle pour répéter les appels à la fonction 00h jusqu'à ce que la fonction 01h ne renvoie plus de caractère. C'est un peu primitif mais ça marche !

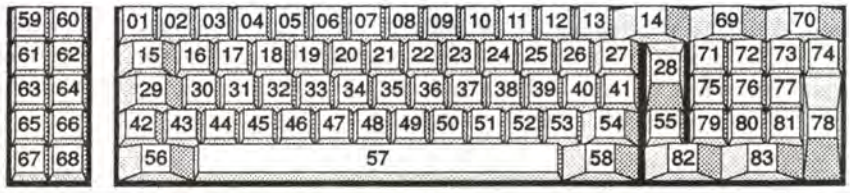
Listing NOKEYB.BAS

```
'*************************************************************************
'*                            N O K E Y B                                *
'*-----------------------------------------------------------------------*
'*   Montre comment effacer le buffer du clavier pour protéger           *
'*   l'utilisateur contre des saisies résiduelles lorsqu'il doit         *
'*   répondre à des questions importantes (par ex "Voulez-vous           *
'*   supprimer tel fichier ?")                                           *
'*-----------------------------------------------------------------------*
'* Auteur                : MICHAEL TISCHER                              *
'* Développé le          : 01.01.1992                                   *
'* Dernière modification : 01.01.1992                                   *
'*************************************************************************
'
DECLARE SUB ClearKbBuffer ()
'-- Programme principal -------------------------------------------------
'
DIM i AS INTEGER                                'Compteur d'itérations
'
CLS
PRINT ("NOKEYB  -  (c) 1992 by Michael Tischer")
PRINT
PRINT ("A 0 les caractères du buffer vont être effacés.")
PRINT
'
FOR i = 10 TO 0 STEP -1                 'le temps de taper des caractères
  PRINT i; "    "
  SLEEP 1
NEXT
'
ClearKbBuffer                                   'Vide le buffer du clavier
'
'--- Affiche les caractères qui restent dans le buffer-------------------
'
ccount = 0                                      'Pas encore de caractère
PRINT
PRINT
PRINT ("Caractères dans le buffer:")
'
DO                                              'En reste-t-il ?
  a$ = INKEY$
  IF a$ <> "" THEN
    FOR i = 1 TO LEN(a$)
      PRINT "   "; ASC(MID$(a$, i, 1)),       'Affiche d'abord le code
      IF ASC(MID$(a$, i, 1)) > 32 THEN              'Caractère spécial ?
        PRINT "("; MID$(a$, i, 1); ")";       'Non, affiche le caractère
      END IF
      PRINT
      ccount = ccount + 1
    NEXT
  END IF
LOOP WHILE a$ <> ""
'
IF ccount = 0 THEN                              'Pas de caractère ?
  PRINT ("(Aucun)")                                     'Non
END IF
PRINT
'
END
'
'*************************************************************************
'* ClearKbBuffer : Efface le contenu du buffer du clavier               *
'* Entrée: néant                                                        *
'* Sortie: néant                                                        *
'*************************************************************************
'
SUB ClearKbBuffer
'
DO          'Retire les caractères du buffer jusqu'à ce qu'il soit vide
LOOP WHILE INKEY$ <> ""
'
END SUB
```

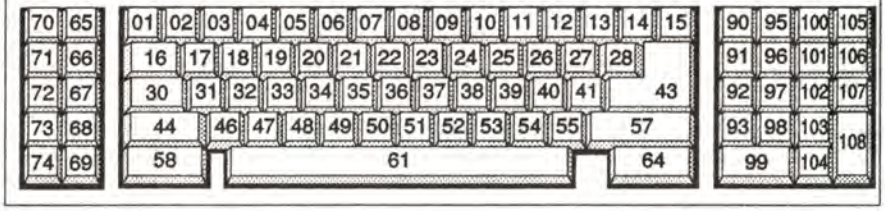
5.2.4. L'affaire des scan codes

Alors que le jeu de caractères ASCII et les codes étendus sont standardisés, le domaine des scan codes baigne dans une certaine incohérence. Les trois types de clavier standard fonctionnent avec des jeux de scan codes complètement différents. Les tableaux suivants mettent en évidence les différences entre les scan codes du clavier de l'XT et les scan codes du clavier de l'AT.

Les Scan-Codes du clavier PC/XT



Les Scan-Codes du clavier AT



Même le principe selon lequel le code break est égal au scan code + 80h n'est pas toujours respecté. C'est ainsi que lorsqu'on relâche une touche le clavier de l'AT émet deux octets : d'abord F0h comme code break et ensuite le scan code de la touche.

Ces incompatibilités sont lissées par le contrôleur qui surveille le clavier de l'AT. Les données présentées au driver du clavier par l'intermédiaire du port 60h sont réagencées de manière traditionnelle de sorte que le principe de base se trouve rétabli.

Les scan codes du clavier étendu

C'est grâce au contrôleur mentionné que les claviers étendus (par le port 60h) renvoient les mêmes scan codes que les claviers des AT, même s'il sont quelque peu étendus. Le clavier étendu supporte trois jeux de scan codes différents qui ne correspondent à aucun de ceux connus jusqu'ici.

Le tableau suivant montre les scan codes additionnels du clavier étendu. Ils se caractérisent surtout par le préfixe E0h qui signale qu'ils ne doivent pas être traités comme les codes ordinaires. En effet la plupart des scan codes sont déjà réservés à d'autres touches et pourraient être mal interprétés. Le principe selon lequel le code break est égal au code make + 80h reste valable, mais le préfixe E0h n'est pas affecté. Losqu'une touche de direction grise est actionnée en coordination avec Majuscule, il faut d'abord neutraliser la touche Majuscule. C'est pourquoi le code break de la touche Majuscule est envoyé avant le scan code concerné.

Avec la touche Majuscule gauche, l'émission préliminaire en cas de code make est E0h AAh, en cas de code Break E0h D2h. Avec la touche Majuscule droite, l'émission préliminaire en cas d'enfoncement est E0h B6h, en cas de relâchement E0h D2h.

Scan Codes du clavier étendu		
Touches de fonction	**Make**	**Break**
F11	57	D7
F12	58	D8
Touches de direction	**Make**	**Break**
Home	E0 47	E0 C7
Curseur haut	E0 48	E0 C8
Page Up	E0 49	E0 C9
Curseur gauche	E0 4B	E0 CB
Curseur droit	E0 4D	E0 CD
Fin	E0 47	E0 C7
Curseur bas	E0 50	E0 D0
Page Down	E0 51	E0 D1
Insert	E0 52	E0 D2
Delete	E0 53	E0 D3
Indications en hexadécimal		

Les développeurs obligés de travailler avec les scan codes, notamment ceux qui développent des programmes résidents, se trouvent confrontés à un grave problème qui ne se résout qu'en demandant à l'utilisateur lors de l'installation de taper la touche d'activation désirée et en enregistrant le scan code émis. Si vous pouvez vous passer des scan codes, faites-le sans hésiter : il vaut mieux se servir des codes ASCII ou des codes de clavier étendus.

Quels sont les scan codes retournés par votre clavier (ou votre contrôleur de clavier) ? C'est ce que va vous révéler un programme étudié dans le cadre de la section suivante consacrée au gestionnaire d'interruption du clavier.

5.3. Le gestionnaire d'interruption du clavier

Le clavier est la cible préférée des programmes résidents. Enregistreurs de macros, utilitaires d'extension du buffer du clavier, drivers spécialisés, programmes TSR variés et divers font assaut d'imagination pour s'immiscer dans le gestionnaire d'interruption du clavier.

Selon le domaine, l'interruption concernée est l'interruption 16h ou l'interruption 09h. La section qui suit donne des exemples pour l'une et l'autre interruption, sans toutefois entrer trop profondément dans la substance des programmes résidents. Car ce sujet est traité à part au chapitre 32, qui s'occupe également des problèmes de lecture du clavier.

5.3.1. Modification de l'interruption 16h du BIOS

Pour étendre ou modifier l'interruption clavier du BIOS, il faut la détourner sur un gestionnaire personnel appelé à la place de l'ancien. Ce mécanisme se justifie par exemple pour ajouter de nouvelles fonctions ou pour reproduire les fonctions 00h et 01h dans les fonctions 10h et 11h. Il est surtout intéressant pour les programmes en langage évolué qui lisent le clavier à travers les fonctions d'une bibliothèque de routines.

Ces routines en effet se servent souvent des fonctions 00h et 01h de l'interruption clavier du BIOS et rendent par exemple impossible la prise en compte des touches F11 et F12 d'un clavier étendu. En détournant l'appel de ces fonctions, on peut malgré tout accéder aux touches en question sasn être obligé de développer ses propres routines de lecture du clavier.

Un utilitaire macro

Mais ce que je voudrais vous présenter ici, c'est un programme d'un tout autre genre, un petit utilitaire de type "macro" écrit en assembleur. Lancé à partir de la ligne de commande de DOS, il se fixe en mémoire de manière résidente jusqu'à être rappelé. Dans sa forme actuelle, il fait apparaître un nom sur l'écran lorsqu'au niveau de la ligne de commande de DOS, de l'éditeur EDIT de DOS, ou même à l'intérieur de Turbo Pascal on tape la combinaison Alt+N. Bien entendu si vous le désirez vous adapterez ce programme de façon qu'il réagisse à une autre combinaison et qu'il affiche un autre texte.

Si vous vous donnez quelque peine, il est possible d'étendre ce programme jusqu'à en faire un macro-générateur complet susceptible d'émettre plusieurs chaînes ou de faire des enregistrements pendant une saisie. Mais ce développement réclame un certain investissement.

Le nouveau gestionnaire d'interruption du clavier du BIOS

Au centre du programme MACROKEY.ASM se trouve le nouveau gestionnaire d'interruption associé à l'interruption 16h du BIOS. C'est sur lui que nous concentrerons notre attention car tout le reste consiste essentiellement à rendre le programme résident et ce point sera étudié en détail plus loin dans cet ouvrage. Examinons donc le nouveau

gestionnaire d'interruption. Appelé NOUVI16, il est situé tout à fait au début du programme.

La routine correspondante commence par autoriser les interruptions matérielles. Leur interdiction, décidée automatiquement au déclenchement de l'instruction INT, n'est pas nécessaire ici. Il s'agit de travailler sur une interruption logicielle et non matérielle.

La branchement qui suit ne sert qu'à sauter par-dessus les caractères "MT" qui signalent que l'installation du programme résident est déjà effectuée. Les choses sérieuses ne démarrent donc qu'au label NI1.

La première tâche consiste à exploiter le numéro de fonction trouvé en AH car le programme ne se sent responsable que pour les fonctions 00h, 01h, 10h et 11h. Il ne veut rien savoir des fonctions 02h et 12h, c'est pourquoi il les repasse à l'ancien gestionnaire dont l'adresse a été stockée à l'installation du programme. Vous voyez donc que le nouveau gestionnaire ne remplace pas l'ancien , mais qu'il le modifie en le complétant, de sorte qu'il est obligé de collaborer avec lui.

Si un appel à l'une des fonctions traitées a été détecté, l'exécution est renvoyée à deux endroits, selon la fonction. Le label FCT0 correspond aux fonctions 00h et 10h, tandis que FCT1 correspond aux fonctions 01h et 11h. Les fonctions sont traitées par paires car elles visent la même tâche.

Dans tous les cas, le programme teste si la macro est déjà en cours d'exécution mais nous étudierons ce point un peu plus loin. Nous supposerons que ce n'est pas le cas. La nouvelle fonction doit alors détecter la frappe de la touche d'activation. Il faut pour cela appeler la fonction correspondante de l'ancien gestionnaire qui mémorise une touche en AX dès qu'elle a été frappée (01h/11h uniquement).

La touche en question est comparée avec la touche d'activation de la macro stockée dans la variable MKEY. A priori, il s'agit de Alt + N, soit 3100h.

Si cette touche n'est pas repérée, le nouveau gestionnaire renvoie à l'appelant le résultat de la fonction de l'ancien gestionnaire, le détournement a donc été sans effet. Notez que lorsqu'il appelle les fonctions 01h ou 11h le gestionnaire ne se termine pas comme d'habitude par l'instruction machine IRET. Si tel était le cas, le registre des indicateurs qui a été empilé au moment de l'exécution de l'instruction INT serait repris sur la pile. Le contenu de l'indicateur de zéro en serait affecté, or cet indicateur est utilisé par les deux fonctions pour signaler la disponibilité d'une touche. On a donc préféré écrire l'instruction VAR RET 2 qui tout comme IRET provoque un retour FAR à l'appelant et enlève le registre des indicateurs de la pile (2 octets), sans toutefois charger.

Lorsque la touche d'activation est détectée, la macro commence son exécution. A chacun des appels de l'une des fonctions 00h/10h ou 01h/11h, ce n'est plus l'ancien gestionnaire qui est invoqué pour lire un caractère du buffer : c'est le buffer de la macro qui livre le caractère.

Ce dernier se trouve dans la variable MSTART et peut contenir autant de caractères qu'on le désire. Il doit se terminer par le label MEND qui marque l'adresse d'offset de fin. Ce n'est qu'à l'issue de son vidage que les fonctions d'origine de l'ancien gestionnaire sont rétablies pour traiter d'autres entrées enregistrées dans le buffer du clavier.

Comme vous pouvez le voir sur le listing, le buffer de la macro contient uniquement les codes ASCII des caractères à afficher, et non pas leurs scan codes. Le scan code renvoyé est toujours 0 car la plupart des programmes n'en tiennent pas compte.

Cette simplification nous épargne bien du travail mais elle rend impossible le traitement de codes étendus par la macro. Vous pouvez cependant, si vous le désirez, entreprendre les améliorations nécessaires pour que le buffer ne restitue pas seulement des codes ASCII mais aussi des scan codes.

Listing : MACROKEY.ASM

```
;*****************************************************************;
;*                          M A C R O K E Y                      *;
;*-------------------------------------------------------------- *;
;*   Fonction     : Montre comment on définit des touches de     *;
;*                  macro commande en reprogrammant le gestionnaire *;
;*                  de l'interruption 16h du BIOS               *;
;*-------------------------------------------------------------- *;
;*   Auteur               : MICHAEL TISCHER                      *;
;*   Développé le         : 03.01.1992                           *;
;*   Dernière modification : 03.01.1992                          *;
;*-------------------------------------------------------------- *;
;*   Assemblage     : MASM MACROKEY                              *;
;*                    LINK MACKROKEY                             *;
;*                    EXE2BIN MACROKEY MACROKEY.COM              *;
;*                                                               *;
;*                    ou                                         *;
;*                                                               *;
;*                    TASM MACROKEY                              *;
;*                    TLINK /T MACROKEY                          *;
;*-------------------------------------------------------------- *;
;*   Appel          : MACROKEY                                   *;
;*****************************************************************;

;-- Début du programme

code  segment para 'CODE'                   ;Définition du segment de code

      org 100h

      assume cs:code, ds:code, es:code, ss:code

start: jmp  mkinit                  ;Appel des routines d'initialisation

;-- Données (restant en mémoire ) ------------------------------------

ancint    equ this dword              ;Ancien vecteur d'interruption 16h
intancofs dw (?)                    ;Offset du vecteur d'interruption 16h
intancseg dw (?)                    ;Segment du vecteur d'interruption 16h

mkey      dw 3100h                  ;Touche d'activation : ALT + N
mstart    db "Micro Application"                ;Texte de la macro
mend      equ this byte

mofs      dw OFFFFh                       ;Pointe sur le caractère
                                    ;suivant du texte de la macro
                                            ;Texte de la macro

;-- Ceci est la nouvelle interruption       16h (reste en mémoire  ) -

nouvi16   proc far

      assume cs:code, ds:nothing, es:nothing, ss:nothing

      sti                           ;Autorise à nouveau les interruptions
      jmp  short ni1

      db "MT"                               ;Identification du programme

ni1:          ;-- teste d'abord si le nouveau gestionnaire est compétent
                                            ;-- pour la fonction appelée

      or   ah,ah                                ;Fonction 00h ou 10h?
      je   fct0
      cmp  ah,10h
      je   fct0

      cmp  ah,01h                               ;fonction 01h ou 11h?
      je   fct1
      cmp  ah,11h
      je   fct1

      jmp  cs:[ancint]                  ;non, faire exécuter la fonction
                                              ;par l'ancien gestionnaire

fct0:     ;-- fonction 00h/10h ----------------------------------------

      cmp  mofs,offset mend           ;Macro déjà en cours d'exécution ?
      jae  check0                               ;non, teste la touche

      ;-- La macro est déjà en exécution, passe au caractère suivant --

fct0p: push bx                                  ;BX va être modifié
      mov  bx,mofs                          ;Position du caractère
      mov  al,cs:[bx]                       ;Charge le code ASCII
      xor  ah,ah                                ;0 comme Scan-Code
      inc  mofs                       ;MOFS pointe sur le caractère suivant
      pop  bx                                       ;Rétablit BX

iret0:    iret                                  ;Retour à l'appelant

check0:               ;-- teste si la touche d'activation a été pressée

      pushf                                 ;Appelle l'ancien gestionnaire
      call cs:[ancint]
      cmp  ax,mkey                  ;A-t-on frappé la touche d'activation ?
      jne  iret0                                ;Non, retour à l'appelant

      ;-- La touche d'activation a été détectée  ---------------------

      mov  mofs,offset mstart           ;Pointeur sur le début du texte
      jmp  fct0p                          ;Renvoie le premier caractère

fct1:     ;-- fonction 01h/11h ----------------------------------------

      cmp  mofs,offset mend                         ;Macro terminée ?
      jae  check1                                   ;Non, teste touche

      ;-- La macro est déjà en exécution, passe au caractère suivant --

fct1p: push bx                                      ;BX va être modifié
      mov  bx,mofs                              ;Position du caractère
      mov  al,cs:[bx]                           ;Charge le code ASCII
      xor  ah,ah                                    ;0 comme Scan-Code
      pop  bx                                           ;Rétablit BX
```

```
;      cmp  ah,1              ;Indicateur de zéro=0: caractère disponible
;
niret1:  ret 2                          ;Retour à l'appelant avec
;                                         : nettoyage de la pile
;
checkl: ;-- teste si la touche d'activation a été pressée---------------
;
;      pushf                          ;Appelle l'ancien gestionnaire
;      call cs:[ancint]
;      je   niret1             ;Pas de touche dans le buffer du clavier
;
;      cmp  ax,mkey             ;A-t-on frappé la touche d'activation ?
;      je   checkla                                          ;Oui
;
;      cmp  ax,0          ;Indicateur de zéro = 0 : caractère disponible
;      ret 2 ;Retour à l'appelant;les indicateurs sont retirés de la pile
;
checkla:  ;-- Touche d'activation détectée, exécute la macro-----------
;
;      xor  ah,ah                     ;Retire la touche d'activation
;      pushf
;      call cs:[ancint]
;
;      mov  mofs,offset mstart          ;Pointe sur le début du texte
;      jmp  fctlp                       ;renvoie le premier caractère
;
nouvi16    endp
;
;-- Fin de la partie résidente ----------
;
finstend   equ this byte
;
;-- Données (susceptibles d'être écrasées par DOS ) ----------
;
installm  db "MACROKEY  -  (c) 1992 by Michael Tischer", 13, 10, 13, 10
          db "Programme installé ", 13, 10
          db "Peut être désactivé par un nouvel appel"
          db 13, 10, "$"
;
retire  db "MACROKEY a été désinstallé.", 13, 10, "$"
;
;-- Programme (susceptible d'être écrasé par DOS )----------
;
mkinit    label near                                    ;Initialisation
;
      assume cs:code, ds:code, es:code, ss:code
;
         ;-- teste d'abord si le programme a déjà été installé     ----
;
      mov  ax,3516h        ;Lit le contenu du vecteur d'interruption 16h
      int  21h                          ;Appelle la fonction de DOS
      cmp  word ptr es:[bx+3],"TM"                ;Teste si  MACROKEY
      jne  mkinstall                              ;a déjà été installé
;
         ;-- Le programme a déjà été installé - Il faut le désinstaller
                                        ;-- et le retirer de la mémoire
;
      mov  dx,es:intancofs          ;Offset de l'interruption     16h
      mov  ax,es:intancseg               ;Segment de l'interruption 16h
      mov  ds,ax                                        ;chargé en DS
      mov  ax,2516h                  ;Remet la vieille routine  dans le
      int  21h                         ;vecteur de l'interruption 16h
;
         ;-- Libère la mémoire de l'ancien MACROKEY      -------------
;
      mov  bx,es                        ;Mémorise le segment du programme
      mov  es,es:[2Ch]          ;Lit le segment environnement dans le PSP
      mov  ah,49h             ;Libère la mémoire de l'ancien environnement
      int  21h
;
      mov  es,bx                                  ;Exploite la fonction 49h
      mov  ah,49h                                  ;pour libérer la mémoire
      int  21h                                       ;de l'ancien MACROKEY
;
      push cs                                                 ;Empile CS
      pop  ds                                          ;et le reprend comme DS
;
      mov  ah,09h                       ;Message: Programme désinstallé
      mov  dx,offset retire
      int  21h
;
      mov  ax,4C00h                                   ;Terminaison normale
      int  21h
;
mkinstall:  ;-- Installe le programme ---------------------------------
;
      mov  intancseg,es              ;Mémorise le segment et l'offset -
      mov  intancofs,bx                    ;du vecteur d'interruption 16h
;
      mov  dx,offset nouvi16               ;Offset nouvelle interruption
      mov  ax,2516h                             ;Détourne le vecteur de
      int  21h                                        ;l'interruption 16h
;
      mov  dx,offset installm              ;Message : Programme installé
      mov  ah,09h                  ;Numéro fonction affichage la chaîne
      int  21h                           ;Appelle la fonction de DOS
;
            ;-- Seuls restent résidents le PSP, la nouvelle routine
                        ;-- d'interruptioon et les données associées.
;
      mov  dx,offset finstend              ;Calcule le nombre paragraphes
      add  dx,15                                        ;nécessaires
      mov  cl,4                                                    ;
      shr  dx,cl
      mov  ax,3100h                ;Termine le programme avec le code 0
      int  21h                                  ;en le laissant résident
;
;-- Fin ------------------------------------------------------------------
;
code  ends                                          ;Fin du segment de CODE
      end  start
;
```

5.3.2. Interception d'interruptions matérielles

Si on désire intervenir au plus bas niveau dans la gestion du clavier, on ne peut pas éviter de manipuler l'interruption 09h. Mais là aussi il faudra continuer de travailler en coopération avec l'ancien gestionnaire car le développement d'un gestionnaire complet est un travail de Sysiphe : vous pouvez vous en persuader en mettant en oeuvre DEBUG pour jeter un coup d'oeil sur le driver du clavier de DOS, KEYB.COM.

Interception des scan codes

Le petit programme appelé GETSCAN respecte bien ce principe. Laissez-moi vous le présenter. Il ne s'agit pas d'un programme résident mais d'un programme tout à fait ordinaire qui sert à afficher des scan codes. Grâce à lui vous pouvez visualiser les scan

codes duclavier avant même qu'ils ne soient pris en charge et éventuellement modifiés par le gestionnnaire du clavier. Avec un clavier étendu, on voit ainsi clairement les deux séquences de codes Make de deux octets de longs qui sont générés par les touches F11/F12 et les touches de direction grises (séquences autrement invisibles).

Normalement seuls les codes Make sont affichés mais si vous démarrez le programme avec en ligne de commande le paramètre /r vous verrez aussi les codes Break. Au centre du programme se trouve le nouveau gestionnaire de l'interruption 09h qui est appelé à chaque pression ou relâchement de touche. Il lit les codes Make et Break sur le port 60h et les transfère dans un buffer interne du programme. Ce dernier est incarné par la variable SCANBUF et géré par deux pointeurs selon un chaînage circulaire, tout comme le buffer du clavier du BIOS. Les pointeurs en question s'appellent SCANNEXT et SCANLAST.

Une fois le scan code mémorisé, le programme se branche sur l'ancien gestionnaire d'interruption, dont l'adresse a été retenue au début du programme. Les touches sont donc traitées de la manière habituelle et après conversion elles sont transférées dans le buffer du clavier. Dans le programme principal, le contenu de ce dernier est lu à l'aide des fonctions 01h et 00h du BIOS, jusqu'à détection de la touche Entrée qui clôture l'exécution.

Mais tant que la touche Entrée n'a pas été lue, le programme principal interroge sans cesse, entre deux appels au BIOS, le buffer interne des scan codes. Tout ce qu'y dépose le nouveau gestionnaire d'interruptions est aussitôt extrait par le programme principal. Mais les scan codes et les codes Break ne disparaissent pas avant d'avoir été affichés. Ce programme vous permet donc de prendre connaissance des scan codes de votre clavier et de détecter des particularités éventuelles. Ce sont surtout les claviers bon marché en provenance d'extrême-orient qui ne sont pas conformes au standard, mais les anomalies ne portent généralement que sur des combinaisons de touches rarement utilisées.

Listing : GETSCAN.ASM

```
;*****************************************************************;
;*                          G E T S C A N                        *;
;*---------------------------------------------------------------*;
;*   Fonction : Affiche les scan codes du clavier avant qu'ils aient *;
;*              pu être filtrés et modifiés par un driver de clavier *;
;*---------------------------------------------------------------*;
;*   Auteur                : MICHAEL TISCHER                     *;
;*   Développé le          : 08.12.1990                          *;
;*   Dernière modification : 02.01.1992                          *;
;*---------------------------------------------------------------*;
;*   Assemblage   : MASM GETSCAN                                 *;
;*                  LINK GETSCAN                                 *;
;*                  EXE2BIN GETSCAN GETSCAN.COM                  *;
;*                                                               *;
;*                    ou                                         *;
;*                                                               *;
;*                  TASM GETSCAN                                 *;
;*                  TLINK /T GETSCAN                             *;
;*---------------------------------------------------------------*;
;*   Appel        : GETSCAN                                      *;
;*****************************************************************;

;-- Constantes et structures ---------------------------------------

INT_CTR   = 20h                 ;Port du contrôleur d'interruption
EOI       = 20h                 ;Fin d'interruption
KB_PORT   = 60h                 ;Port du clavier

;-- Début du programme proprement dit ------------------------------

code      segment para 'CODE'   ;Définition du segment de code

          org 100h

          assume cs:code, ds:code, es:code, ss:code

start:    jmp  getscan          ;Branchement au début du programme

;-- Données --------------------------------------------------------

int9_ptr  equ this dword        ;Ancien vecteur d'interruption 9h
int9_ofs  dw 0                  ;Offset de l'ancien gestionnaire
int9_seg  dw 0                  ;Segment de l'ancien gestionnaire

allscan   db 0                  ;Faut-il afffficher les codes Break ?
```

```
scanbuf   db 32 dup (0)                         ;Buffer pour scan codes
scanend   equ this byte

scannext  dw offset scanbuf       ;Suivant dans le buffer des scan codes
scanlast  dw offset scanbuf       ;Dernier dans le buffer des scan codes

copyr     db  "GETSCAN - (c) 1990, 92 by MICHAEL TISCHER",13,10,13,10
          db  "Appuyez sur une touche quelconque pour afficher "
          db  "son scan code  ",13,10
          db  "ou sur <Entrée> pour sortir du programme."
          db  13,10,13,10, "$"

hexdigits db  "0123456789ABCDEF"   ;Chiffres pour convertir un nb hexa.

scarmes   db  "Scancode: "                              ;Message
scandecf  db  "000 ("
scanhex   db  "xx)", 13, 10, "$"

;== Programmcode ==============================================

getscan:  mov  ah,09h                           ;Message de copyright
          mov  dx,offset copyr
          int  21h

          ;-- Recherche le paramètre /R pour afficher tous les codes----

          cmp  word ptr ds:[130], "R" * 256 + "/"
          je   gsall
          cmp  word ptr ds:[130], "r" * 256 + "/"
          jne  gsnall

gsall:    mov  allscan,1                ;Affiche aussi les codes Release

gsnall:   ;-- Installe le nouveau gestionnaire d'interruption  --------

          mov  ax,3509h           ;Contenu du vecteur d'interruption 09h
          int  21h                              ;Interruption de DOS
          mov  int9_seg,es              ;Mémorise le segment et l'offset
          mov  int9_ofs,bx                ;du vecteur d'interruption 9h

          mov  dx,offset nouv19   ;Offset de la nouvelle routine d'int.
          mov  ax,2509h                         ;Détourne le vecteur de
          int  21h                                ;l'interruption 09h

gs1:      ;-- Boucle de lecture ------------------------------------

          mov  ah,01h             ;Y a-t-il un  caractère qui soit prêt
          int  16h                                                ;
          je   gs2                                        ;Non --> GS2

          xor  ah,ah                                  ;Oui, lit le caractère
          int  16h
          cmp  al,13                                    ;Est-ce <Entrée> ?
          je   gs4                              ;Oui--> Terminer le programme

gs2:      mov  di,scannext              ;Pointe sur le prochain scan code
          cmp  di,scanlast                                  ;Buffer vide
          je   gs1                                        ;Oui, recommence
          ;== Transforme le scan code en ASCII et l'affiche ============

          mov  word ptr scandecf,   32 shl 8 + 32
          mov  byte ptr scandecf+2, 32

          mov  si,offset scandecf+2
                        ;SI pointe sur le dernier chiffre dans le buffer
          mov  al,[di]                          ;Charge le scan code en AL
          mov  bl,10                          ;Le diviseur est toujours 10

gs3:      xor  ah,ah                ;Annule octet poids fort du dividende
          div  bl                                     ;Divise AX par 10
          or   ah,'0'                         ;Convertit AH au format ASCII
          mov  [si],ah                                ;Stocke dans buffer
          dec  si                                     ;Caractère suivant
          or   al,al                                ;y a-t-il un reste ?
          jne  gs3                                ;Oui --> prochain chiffre

          ;-- Affiche aussi le code en hexadécimal ---------------------

          mov  bx,offset hexdigits  ;BX pointe sur table chiffres hexa.
          mov  al,[di]                ;Isole la partie inf. du scan code
          and  al,15
          xlat                          ;Recherche le chiffre hexa dans table
          mov  ah,al                     ;Transfère le chiffre inférieur dans
                                                          ;l'octet supérieur
          mov  al,[di]                  ;Isole la partie haute du scan code
          mov  cl,4
          shr  al,cl
          xlat                          ;Recherche le chiffre hexa dans table

          mov  word ptr scanhex,ax ;Met les deux chiffres dans le buffer

                    ;-- Avance le pointeur dans le buffer des scan codes

          inc  di
          cmp  di,offset scanend                            ;Trop loin ?
          jne  gsnowrap                           ;Non ---> note la position

          mov  di,offset scanbuf                    ;Oui, recommence au début

gsnowrap: mov  scannext,di            ;Mémorise pos. du prochain caractère

          ;-- Affichage ------------------------------------------------

          mov  ah,09h                                       ;Affiche la chaîne
          mov  dx,offset scarmes
          int  21h

          jmp  gs1                                          ;Recommence

gs4:      ;-- Prépare la fin du programme ------------------------------

          lds  dx,int9_ptr              ;Rétablit l'ancien gestionnaire -
          mov  ax,2509h                   ; de l'interruption du clavier
          int  21h

          mov  ax,4C00h                             ;tout va bien, on termine
          int  21h

;== Gestionnaire de l'interruption 09h (Clavier) ==============

nouv19    proc far

          assume cs:code, ds:nothing, es:nothing, ss:nothing

          push ax                                           ;Empile AX
          in   al,K8_PORT               ;Lit le scan code sur port clavier

          cmp  al,128                               ;Est-ce un code Release ?
          jb   i9note                                ;Non ---> on prend note

          cmp  allscan,0                            ;Oui, en tenir compte ?
          je   i9end                                          ;Non ---> retour

i9note:   ;-- mémorise le scan code dans le buffer des scan codes--------

          push di                                   ;DI va être modifié, on le sauve
          mov  di,scanlast        ;DI pointe sur prochaine position buffer
          mov  cs:[di],al                          ;Y mémorise le scan code
          inc  di                 ;DI pointe sur la position suivante
          cmp  di,offset scanend                            ;Trop loin?
          jne  i9nowrap                           ;Non ---> note la position

          mov  di,offset scanbuf                    ;Oui reprend au début

i9nowrap: mov  scanlast,di      ;Mémorise position du prochain caractère
          pop  di                                           ;restaure DI
i9end:    pop  ax         ;Transmet le scan code à l'ancien gestionnaire
          jmp  [int9_ptr]                                   ;du clavier

nouv19    endp

;== Fin =======================================================

code      ends                                      ;Fin du segment de code
          end  start
```

5.4. Le contrôleur du clavier et sa programmation

Depuis l'apparaition de l'AT, le calvier est assisté du côté du système par un processeur Intel de type 8042 qui pilote la communication entre le clavier et système. Avec le PC et l'XT il était juste possible de réceptionner des caractères en provenance du clavier. Ce circuit autorise la transmission d'informations en sens inverse. Car entre le BIOS et le clavier étendu existe une coopération active qui permet au système d'influencer le comportement du clavier.

Communication avec le clavier

Du côté du clavier, les pièces maîtresses de la communication sont constituées par un registre d'état et des buffers d'entrée-sortie.

Le buffer de sortie sert à transmettre :

- ✓ les codes du clavier liés à l'enfoncement ou au relâchement des touches
- ✓ des données réclamées par le système par une commande

il peut être lu par l'intermédiaire du port 60h.

Le buffer d'entrée peut être atteint par l'un des ports 60h ou 64h. Le port à utiliser dépend du type d'information concerné. Lorsque le système désire envoyer un code de commande au clavier, il doit le mettre dans le port 60h. L'octet de données associé à la commande doit être communiqué au port 64h.

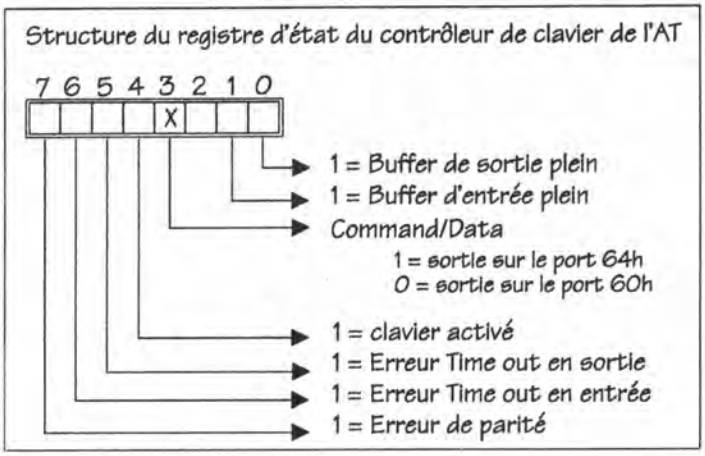
Les deux octets parviennent ainsi dans le buffer d'entrée du clavier, cependant qu'un indicateur dans le registre d'état signale selon le port adressé s'il s'agit d'un octet de commande (port 64h) ou d'un octet de données (port 60h).

Le registre d'état du clavier

En plus de cet indicateur, le registre d'état contient deux bits particulièrement importants pour la gestion de la communication : les bits 0 et 1.

Le bit 0 indique l'état du buffer de sortie ? S'il est à 1, le buffer de sortie contient des informations que le système n'a pas encore lues sur le port 60h. Lorsque le port subit une lecture, ce bit est automatiquement annulé.

Le bit 1 fonctionne de la même façon. Il est à 1 lorsque le système a placé dans le buffer d'entrée un caractère que n'a pas encore traité le clavier. Le buffer d'entrée ne peut donc recevoir des informations que lorsque ce bit est à 0.



Paramétrage de la vitesse de répétition

Parmi les différentes commandes que le système est susceptible d'envoyer au clavier, il en est deux qui se révèlent intéressantes dans les programmes d'application, car elles jouent un rôle en dehors du gestionnaire d'interruption. La première de ces commandes concerne la concerne la vitesse et le délai de répétition du clavier. La vitesse de répétition ou vitesse Typematic (Typematic Rate) est le nombre de codes Make envoyés au système à chaque seconde lorsqu'une touche est maintenue enfoncée. Cette vitesse varie entre 2 et 30 caractères par seconde au maximum. Pour que la répétition d'un caractère ne se déclenche pas de façon intempestive, elle ne se met en route qu'après un certain délai codé en binaire de la façon suivante :

Codage des délais de répétition du clavier de l'AT

Code	Délai de répétition
00b	1/4 Seconde
01b	1/2 Seconde
10b	3/4 Seconde
11b	1 Seconde

Ce délai ne peut pas toujours être respecté avec une précision absolue. Les valeurs du tableau sont à interpréter avec une tolérance de 20%.

La vitesse de répétition est également codée en binaire, comme le montre le tableau suivant :

Codage des vitesses de répétition du clavier de l'AT							
Code	C/s*	Code	C/s*	Code	C/s*	Code	C/s*
11111b	2,0	10111b	4,0	01111b	8,0	00111b	16,0
11110b	2,1	10110b	4,3	01110b	8,6	00110b	17,1
11101b	2,3	10101b	4,6	01101b	9,2	00101b	18,5
11100b	2,5	10100b	5,0	01100b	10,0	00100b	20,0
11011b	2,7	10011b	5,5	01011b	10,9	00011b	21,8
11010b	3,0	10010b	6,0	01010b	12,0	00010b	24,0
11001b	3,3	10001b	6,7	01001b	13,3	00001b	26,7
11000b	3,7	10000b	7,5	01000b	15,0	00000b	30,0
* Caractères par seconde							

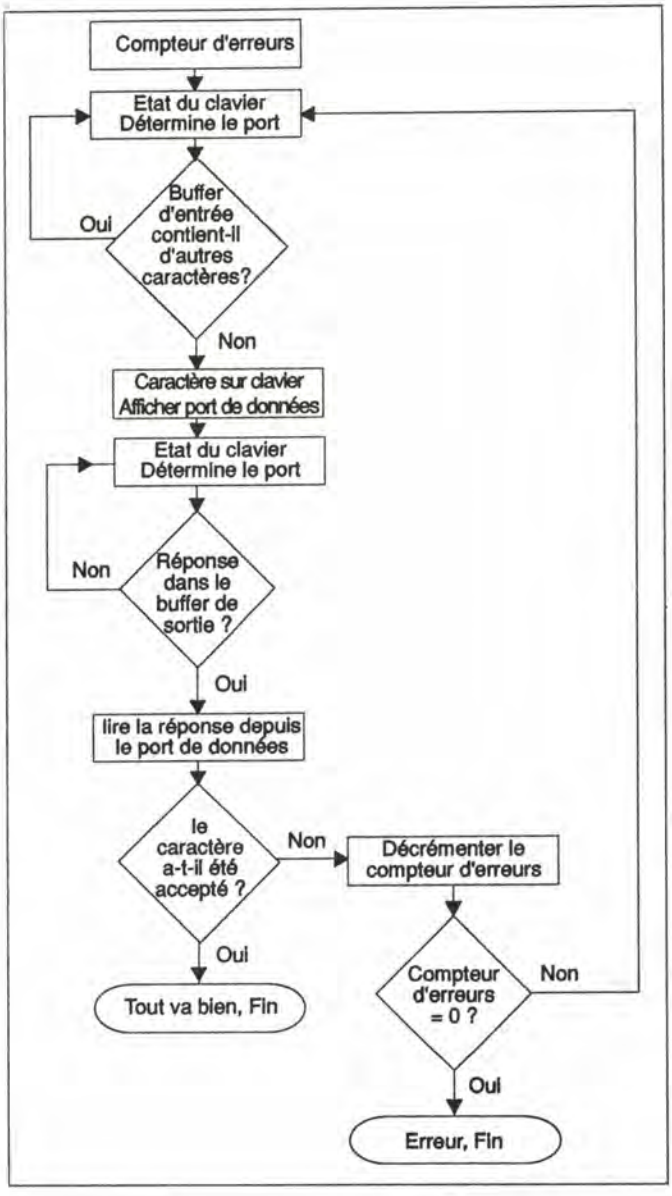
Ce tableau peut sembler arbitraire au premier abord. En fait il repose sur une formule mathématique. Si A représente la valeur binaire des bits 0, 1 et 2 , et B celle des bits 3 et 4, la formule qui donne la vitesse de répétition s'écrit :

$(8 + A) * 2^B * 0.00417 * 1 / \text{seconde}.$

Le délai et la vitesse sont ensuite regroupés dans un même octet, le délai étant placé avant les 5 bits de la vitesse. L'octet "Typematic" ainsi obtenu ne peut toutefois pas encore être communiqué directement au clavier car il doit être précédé du code de la commande associée (F3h). Les deux octets doivent être envoyés de pair au clavier à travers le port 60h mais cela ne se fait pas en émettant tout simplement chacun d'eux à l'aide d'une instruction OUT. La transmission de chaque octet doit en effet obéir à un protocole de transmission qui inclut un test de l'état du clavier. Il est possible en effet que la transmission ne se déroule pas sans erreur dès le premier essai. Comme le même travail doit être effectué pour chacun des deux octets, il est conseillé d'utiliser un sous-programme pour transmettre chaque octet. C'est ce que montre l'organigramme ci-dessous.

Transmission d'octets au clavier

Comme vous le voyez, on commence par mettre en place un compteur d'erreurs qui permettra à la routine de répéter trois fois une transmission infructueuse. Le port d'état du clavier est ensuite testé dans le cadre d'une sorte de boucle, jusqu'à ce que le bit 0 soit nul, ce qui indique que le buffer d'entrée est vide. Ce n'est qu'à ce moment que le caractère à transmettre pourra être envoyé au clavier à travers le port 60h. Pour s'assurer que le caractère est arrivé à bon port (une erreur de parité pourrait très bien se produire), le clavier renvoie une code de réponse, qui n'est cependant pas disponible tant que le bit 1 du port d'état du clavier ne vaut pas 1.



Ce registre est donc lu et relu à travers le port 64h, à l'intérieur d'une boucle, jusqu'à ce que cette condition soit remplie. La réponse du clavier à notre transmission peut alors être lue sur le port de données du clavier. Cette réponse est le code 0FAh si la transmission a fonctionné correctement, ce code faisant donc office "d'accusé de réception" ou "Acknowledge". Tout autre code signale une erreur, ce qui conduit le sous-programme à décrémenter le compteur d'erreurs et à répéter toute l'opération, à

moins qu'il n'ait atteint la valeur 0. Dans ce dernier cas, le sous-programme se termine en signalant au programme appelant qu'une erreur est apparue.

Programmes d'exemple

Pour illustrer ces explications, vous trouverez dans les pages suivantes deux programmes écrits en Pascal (TYPMP.PAS) et en C (TYPMC.C) qui vous permettront de régler les paramètres Typematic à votre guise. Le coeur de ces programmes est constitué par une routine en assembleur qui reçoit la vitesse Typematic souhaitée et la transmet au clavier. Cette routine contient le sous-programme que nous venons d'étudier et qui est d'abord appelé pour communiquer au clavier le code de la commande Set Typematic. Le même sous-programme est ensuite appelé une seconde fois pour envoyer la vitesse Typematic désirée au clavier.

La vitesse Typematic est communiquée aux deux programmes par le moyen de paramètres de la ligne de commande. Si ces paramètres font défaut, les programmes affichent un message qui décrit le mode d'appel correct.

Les deux programmes font intervenir des modules en assembleur qui permettent la fixation proprement dite de la vitesse Typematic. Ces modules s'appellent TYPMPA.ASM et TYPMCA.ASM. Les programmeurs qui travaillent en Pascal peuvent éviter l'inclusion du module en assembleur, les instructions correspondantes étant codées sous forme INLINE. Les programmeurs qui travaillent en C devront assembler le module approprié puis le relier au module de haut niveau TYPM.C.

Pour tester l'effet d'un changement de la vitesse Typematic, nous vous conseillons de prendre tout d'abord une valeur minimale (0) puis maximale (30). Maintenez une touche appuyée : la différence constatée sera sensible.

Listing : TYPMP.PAS

```
{*****************************************************************}
{*                          T Y P M P                            *}
{*---------------------------------------------------------------*}
{*   Fixe la vitesse Typematic d'un clavier étendu               *}
{*---------------------------------------------------------------*}
{*   Auteur              : MICHAEL TISCHER                       *}
{*   Développé le        : 27.08.1988                            *}
{*   Dernière modification : 03.01.1992                          *}
{*****************************************************************}

program TYPMP;

{*****************************************************************}
{* SetTypm : Transmet la vitesse Typematic au contrôleur du clavier  *}
{* Entrée  : RATE : Vitesse à fixer                              *}
{* Sortie  : TRUE, si la vitesse a pu être fixé                  *}
{*           FALSE en cas d'erreur d'accès au contrôleur         *}
{* Info    : Cette fonction peut être incluse dans une unité     *}
{*****************************************************************}

{$F+}                                   { Exploite le modèle FAR-Call }

function SetTypm( Rate : byte ) : boolean;

begin
  inline(
        $32/$D2/$B4/$F3/$FA/$EB/$13/$00/$75/$0A/$BA/$66/$06/$EB/
        $08/$00/$75/$02/$FE/$C2/$FB/$BB/$56/$FF/$EB/$27/$90/$51/
        $53/$B3/$03/$33/$C9/$E4/$64/$A8/$02/$E0/$FA/$BA/$C4/$E6/
        $60/$E4/$64/$A8/$01/$E1/$FA/$E4/$60/$3C/$FA/$74/$07/$FE/
        $CB/$75/$E6/$80/$C8/$01/$5B/$59/$C3
      );
end;

{$F-}

{*****************************************************************}
{**                  Programme principal                        **}
{*****************************************************************}

var  Delay,                                  { Mémorise le délai }
     Speed,                                  { et la fréquence de répétition }
     Fpos1,
     FPos2   : integer;                      { Erreur de conversion de chaîne }
     ParErr : boolean;                       { Erreur de transmission des paramètres }

begin
  writeln(#13#10,'TYPMP  -  (c) 1988, 1992 by MICHAEL TISCHER');
```

```
  ParErr := true;                              { Saisie a priori erronée }
  if ParamCount = 2 then                { Dispose-t-on de 2 paramètres ? }
    begin                                                      { Oui }
      val(ParamStr(1), Delay, FPos1);            { Conversion en entier }
      val(ParamStr(2), Speed, FPos2);                 { des paramètres }
      if ((FPos1=0) and (FPos2=0)) then       { Erreur de conversion ? }
  if ((Delay < 4) and (Speed <32)) then        { Non tout va bien  }
    ParErr := false;                  { Les paramètres sont corrects }
    end;
  if ( ParErr ) then                        { En cas d'erreur de paramètre }
    begin
      writeln('Appel : TYPMP  Délai Vitesse ');
      writeln('                ',@30,'     ',@30);
      writeln('                          ');
      writeln('  ');
      writeln('  0 : 1/4 seconde       0 : 30,0 rép./s  ');
      writeln('  1 : 1/2 seconde       1 : 26,7 rép./s. ');
      writeln('  2 : 3/4 seconde       2 : 24,0 rép./s. ');
      writeln('  3 : 1 seconde         3 : 21,8 rép./s. ');
      writeln('           .          ');
      writeln(' Précision   20%            .          ');
      writeln('           .          ');
      writeln('                         28 :  2,5 rép./s. ');
      writeln('                         29 :  2,3 rép./s. ');
      writeln('                         30 :  2,1 rép./s. ');
      writeln('                         31 :  2,0 rép./s. ');
      writeln('                        ');
    end
  else                                    { Les paramètres sont corrects }
  begin
    if (SetTypm( (Delay shl 5) + Speed )) then ( Fixe vitesse Typematic)
      writeln('La vitesse Typematic a été fixée .')
    else
      writeln('ATTENTION: Erreur d''accès au contrôleur');
  end;
end.
```

Listing : TYPMPA.ASM

```
;*****************************************************************;
;*                          T Y P M P A                          *;
;*---------------------------------------------------------------*;
;*   Fonction       : Routine en assembleur à inclure dans un    *;
;*                    programme en Turbo Pascal permettant de fixer *;
;*                    la vitesse Typematic du clavier étendu     *;
;*---------------------------------------------------------------*;
;*   Auteur               : MICHAEL TISCHER                     *;
;*   Développé le         : 27.08.1988                          *;
;*   Dernière modification : 01.03.1992                         *;
;*---------------------------------------------------------------*;
;*   Assemblage  : MASM TYPMPA;                                 *;
;*                 LINK TYPMPA                                  *;
;*                 EXE2BIN TYPMPA TYPMPA.BIN                    *;
;* ...transformer en instructions INLINE et inclure dans        *;
;*                 le programme TYPMP.PAS                       *;
;*****************************************************************;

;-- Constantes ---------------------------------------------------

KB_STATUS_P   equ 64h                       ;Port d'état du clavier
KB_DATA_P     equ 60h                       ;Port de données du clavier

OB_FULL       equ 1                 ;Bit 0 dans le port d'état du clavier
                                    ;un caractère dans le buffer de sortie
IB_FULL       equ 2                 ;Bit 1 dans le port d'état du clavier
                                    ;un caractère dans le buffer d'entrée

ACK_SIGNAL    equ 0fah          ;Signal d'accusé de réception du clavier
SET_TYPEM     equ 0f3h                      ;Commande Set Typematic

MAX_TRY       equ 3                     ;Nombre de tentatives permises


;-- Programmcode -------------------------------------------------

code      segment para 'CODE'             ;Définition du segment de code

          org  100h

          assume cs:code, ds:code, ss:code, es:code

;-----------------------------------------------------------------
;-- SET_TYPM: Transmet la vitesse Typematic au contrôleur du clavier ---
;--
;-- Info    : On suppose que le modèle est du type FAR CALL

set_typm  proc near

sframe0   struc                       ;Structure pour accéder à la pile
bp0       dw ?                                    ;Mémorise BP
ret_adr0  dd ?                        ;Adresse de retour à l'appelant
                                                  ;(Adresse FAR )
trate0    dw ?                              ;Vitesse Typematic à fixer
sframe0   ends                                    ;Fin de la structure

frame     equ [ bp - bp0 ]        ;Adresse les éléments de la structure

          ;-- Les deux instuctions suivantes sont déjà effectuées------
          ;-- par TURBO                                    --------

                                    ;push bp                ;Empile BP
                          ;mov  bp,sp             ;Transfère SP dans BP

          xor  dl,dl                    ;Transmission a priori défectueuse
          mov  ah,SET_TYPEM       ;Commande de fixation vitesse Typmatic
          cli                                     ;Inhibe les interruptions
          call send_kb                                ;Vers le contrôleur
          jne  error                                ;Erreur? Oui--> Error

          mov  ah,byte ptr frame.trate0       ;Lit la vitesse Typematic
          call send_kb                                ;Vers le contrôleur
          jne  error                                ;Erreur? Oui -> Error

          inc  dl                                 ;tout va bien, renvoie TRUE

error:    sti                                     ;Rétablit les interruptions
          mov  [bp-1],dl                ;Met la valeur de retour sur la pile
                              ;pop  bp                ;Restaure BP (TURBO)
          jmp  fin                          ;Rend la main au programme Pascal

set_typm  endp

;-----------------------------------------------------------------
;-- SEND_KB: Envoie un octet au contrôleur du clavier  ----------
;-- Entrée   : AH = octet à envoyer
;-- Sortie   : Indicateur de zéro: 0=Erreur, 1=o.k.
;-- Registres: AX le registre des indicateurs sont modifiés
;-- Info     : Routine à usage interne, ne doit petre appelée qu'à
;--            partir du présent module

send_kb   proc near

          push cx                                 ;Empile tous les registres
          push bx                                   ;modifiés par la routine

          mov  bl,MAX_TRY                   ;Nombre de tentatives tolérées

          ;-- Attend que le contrôleur puisse recevoir des données -----

skb_1:    xor  cx,cx                              ;65536 itérations au maximum
skb_2:    in   al,KB_STATUS_P                 ;Lit le contenu du port d'état
          test al,IB_FULL             ;Reste-t-il un caractère dans buffer ?
          loopne skb_2                                        ;Oui--> SKB_2

          ;-- Envoie un caractère au contrôleur ----------------------

          mov  al,ah                            ;Transfère le caractère en AL
          out  KB_DATA_P,al                     ; puis dans le port de données
skb_3:    in   al,KB_STATUS_P                 ;Lit le contenu du port d'état
          test al,OB_FULL                ;Réponse dans le buffer de sortie ?
          loope skb_3                                       ;Non  --> SKB_3

          ;-- Lit et exploite la réponse du contrôleur   --------------

          in   al,KB_DATA_P                           ;Lit le port de données
          cmp  al,ACK_SIGNAL                            ;Caractère accepté ?
          je   skb_end                                  ;Oui -> tout va bien

          ;-- Le caractère n'a pas été accepté  ----------------------

          dec  bl                               ;Décrémente le compteur d'erreurs
          jne  skb_2                                    ;Peut-on recommencer ?
                                                        ;Oui--> SKB_2
          or   bl,1                     ;Non, mettre à 0 l'indicateur de zéro
```

```
                                        ;pour signaler l'erreur
skb_end:  pop  bx                   ;Dépile les registres sauvegardés
          pop  cx                                   ;
          ret                               ;Retourne à l'appelant

send_kb   endp
```

```
;-------------------------------------------------------------------
ffin      label near

;-- Fin ----------------------------------------------------------

code      ends                                  ;Fin du segment de code
          end  set_typm
```

Listing : TYPMC.C

```
/*****************************************************************/
/*                          T Y P M C                            */
/*---------------------------------------------------------------*/
/*   Fonction       : Permet de choisir la vitesse Typematic     */
/*                    d'un clavier étendu                        */
/*---------------------------------------------------------------*/
/*   Auteur              : MICHAEL TISCHER                       */
/*   développé le        : 28.08.1988                            */
/*   Dernière modification : 03.01.1992                          */
/*---------------------------------------------------------------*/
/*   Modèle mémoire : SMALL                                      */
/*---------------------------------------------------------------*/
/*   Modules        : TYPMC.C + TYPMCA.ASM                       */
/*****************************************************************/

/*-- Fichiers d'inclusion ---------------------------------------*/

#include <stdlib.h>
#include <stdio.h>

/*-- Typedefs ---------------------------------------------------*/

typedef unsigned char BYTE;             /* Bricolage d'un type Byte */
typedef BYTE BOOL;                    /* Prend la valeur TRUE ou FALSE */

/*-- Constantes -------------------------------------------------*/

#define TRUE  ( 1 == 1 )                        /* Constantes de type BOOL */
#define FALSE ( 0 == 1 )

/*-- Inclusion de fonctions externes issues du module en assembleur --*/

extern BOOL set_typm( BYTE trate );     /* Fixe la vitesse Typematic */

/*****************************************************************/
/**                       Programme principal                   */
/*****************************************************************/
```

```
void main(int argc, char *argv[] )
{
 int delay,                                     /* Mémorise le délai */
     speed;                              /* et la fréquence de répétition */

 printf("\nTYPMC  -  (c) 1988, 1992 by MICHAEL TISCHER\n");
 if (argc!=3 || ( (delay = atoi(argv[1]))<0 || delay>3 ) ||
                                ( (speed = atoi(argv[2]))<0 || speed>31 ))
 {              /* Il n'y a pas deux paramètres ou ils sont faux    */
   printf("Appel : TYPMC Délai Vitesse \n");
   printf("                \xle        \xle\n");
   printf("                                \n");
   printf(" \n");
   printf(" 0 : 1/4 Seconde       0 : 30,0 rép./s. \n");
   printf(" 1 : 1/2 Seconde       1 : 26,7 rép./s. \n");
   printf(" 2 : 3/4 Seconde       2 : 24,0 rép./s. \n");
   printf(" 3 : 1 Seconde         3 : 21,8 rép./s. \n");
   printf("              .                 \n");
   printf(" Précision    20\x\x            .          \n");
   printf("              .                 \n");
   printf("                          28 :  2,5 rép./s. \n");
   printf("                          29 :  2,3 rép./s. \n");
   printf("                          30 :  2,1 rép./s. \n");
   printf("                          31 :  2,0 rép./s. \n");
   printf("                          \n");
 }
 else  /* Les paramètres sont corrects */
 {
   if (set_typm( (BYTE) ((delay << 5) + speed ))) /* Fixe la vitesse Type-
matic */
     printf("La vitesse Typematic a été fixée .\n");
   else
     printf("ATTENTION ! Erreur d'accès au contrôleur du clavier\n");
 }
}
```

Listing : TYPMCA.ASM

```
;*****************************************************************;
;*                          T Y P M C A                          *;
;*---------------------------------------------------------------*;
;*   Fonction       : Routine en assembleur à inclure dans un    *;
;*                    programme en C permettant de fixer la vitesse *;
;*                    Typematic du clavier étendu                *;
;*---------------------------------------------------------------*;
;*   Auteur              : MICHAEL TISCHER                       *;
;*   Développé le        : 27.08.1988                            *;
;*   Dernière modification : 27.08.1988                          *;
;*---------------------------------------------------------------*;
;*   Assemblage     : MASM TYPMCA;     ou      TASM TYPMCA       *;
;*                    ... puis lier avec TYPMC.C                 *;
;*****************************************************************;

;-- constantes -------------------------------------------------

KB_STATUS_P   equ 64h                       ;Port d'état du clavier
KB_DATA_P     equ 60h                     ;Port de données du clavier

OB_FULL       equ 1                 ;Bit 0 dans le port d'état du clavier
                                   ;un caractère dans le buffer de sortie
IB_FULL       equ 2                 ;Bit 1 dans le port d'état du clavier
                                    ;un caractère dans le buffer d'entrée
```

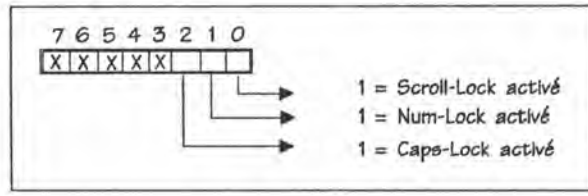
```
ACK_SIGNAL    equ 0fah              ;Signal accusé de réception du clavier
SET_TYPEM     equ 0f3h                          ;Code Set Typematic

MAX_TRY       equ 3                          ;Nombre de tentatives permises

;-- Déclaration des segments pour le programme en C ------------

IGROUP group _text                ;Regroupement des segments de programme
DGROUP group _bss, _data          ;Regroupement des segments de données
        assume CS:IGROUP, DS:DGROUP, ES:DGROUP, SS:DGROUP

_BSS    segment word public 'BSS'         ;Segment accueillant toutes les
_BSS    ends                            ;variables statiques non initialisées

_DATA   segment word public 'DATA'              ;Segment accueillant les
                                              ;variables globales et statiques
                                                          ;initialisées
_DATA   ends

;-- Programme --------------------------------------------------

_TEXT   segment byte public 'CODE'                  ;Segment de programme

public    _set_typm
```

```
;-------------------------------------------------------------------
;-- SET_TYPM: Transmet la vitesse Typematic au contrôleur du clavier ---
;
;-- Appel depuis C: bool set_typem( byte trate );
;-- Renvoie : TRUE, si la vitesse Typematic a pu être fixée
;--           FALSE si une erreur est survenue
;
_set_typm  proc near
;
sframe0    struc                        ;Structure pour accéder à la pile
bp0        dw ?                                    ;Mémorise BP
ret_adr0   dw ?                         ;Adresse de retour à l'appelant
trate0     dw ?                             ;Vitesse Typematic à fixer
sframe0    ends                                 ;Fin de la structure
;
frame      equ [ bp - bp0 ]          ;Adresse les éléments de la structure
;
       push bp                                              ;Empile BP
       mov  bp,sp                                 ;Transfère SP dans BP
;
       xor  dx,dx                        ;Transmission a priori défectueuse
       mov  ah,SET_TYPEM          ;Commande de fixation vitesse Typmatic
       cli                                        ;Inhibe les interruptions
       call send_kb                                    ;Vers le contrôleur
       jne  error                                 ;Erreur ? Oui --> Error
;
       mov  ah,byte ptr frame.trate0          ;Lit la vitesse Typematic
       call send_kb                                    ;Vers le contrôleur
       jne  error                                    ;Erreur Oui --> Error
;
       inc  dl                                  ;tout va bien ,renvoie TRUE
;
error: sti                                      ;Rétablit les interruptions
       mov  ax,dx                          ;Charge la valeur de retour en AX
       pop  bp                                          ;Restaure BP
       ret                                  ;Rend la main au programme en C
;
_set_typm  endp
;
;-------------------------------------------------------------------
;-- SEND_KB: envoie un octet au contrôleur du clavier ----------------
;-- Entrée   : AH = octet à evoyer
;-- Sortie : Indicateur de zéro : 0=Erreur , 1=o.k.
;-- Registres : AX et le registres des indicateurs
;;    sont modifiés
;-- Info     : Routine à usage interne, ne peut être appelée
;--            qu'à partir du présent module
;
send_kb   proc near
;
      push cx                                   ;Empile tous les registres
      push bx                                     ;modifiés par la routine
;
      mov  bl,MAX_TRY                      ;Nombre de tentatives tolérées
;
            ;-- Attend que le contrôleur puisse recevoir des données
;
skb_1:   xor  cx,cx                              ;65536 itérations au maximum
skb_2: in   al,KB_STATUS_P                  ;Lit le contenu du port d'état
       test al,IB_FULL          ; Reste-t-il un caractère dans le buffer ?
       loopne skb_2                                          ;Oui --> SKB_2
;
                              ;-- Envoie un caractère au contrôleur
;
       mov  al,ah                             ;Transfère le caractère en AL
       out  KB_DATA_P,al                      ;puis dans le port de données
skb_3:   in   al,KB_STATUS_P                ;Lit le contenu du port d'état
       test al,OB_FULL                  ;Réponse dans le buffer de sortie ?
       loope skb_3                                             ;Non  --> SKB_3
;
            ;-- Lit et exploite la réponse du contrôleur ----------
;
       in   al,KB_DATA_P                          ;Lit le port de données
       cmp  al,ACK_SIGNAL                           ;Caractère accepté ?
       je   skb_end                                 ;Oui--> tout va bien
;
         ;-- Le caractère n'a pas été accepté  ----------------------
;
       dec  bl                               ;Décrémente le compteur d'erreurs
       jne  skb_2                                 ;Peut-on recommencer  ?
                                                          ;Oui--> SKB_2
       or   bl,1                        ;Non , Mettre à 0 l'indicateur de zéro
                                                  ;pour signaler l'erreur
;
skb_end: pop  bx                              ;Dépile les registres sauvegardés
         pop  cx                                                       ;
         ret                                          ;Retourne à l'appelant
;
send_kb   endp
;
;-------------------------------------------------------------------
;
_text  ends                                          ;Fin du segment de code
       end                                                ;Fin du programme
```

Commande des diodes électroluminescentes du clavier

Tout comme la vitesse Typematic est paramétrable, les diodes électroluminescentes ou LEDs du clavier de l'AT peuvent également être allumées ou éteintes par programme. Le code de la commande associée "Set/Reset Mode Indicator" porte le numéro 0EDh.

Lorsque ce code de commande a pu être correctement transmis au clavier, ce dernier attend un octet reflétant l'état des trois diodes. Dans cet octet, chaque diode est associée à un bit, et la valeur 1 exprime l'activation, comme le montre la figure suivante :



L'allumage ou l'extinction de ces diodes n'a de sens qu'en coordination avec l'indicateur d'état du clavier. Cet indicateur cependant n'est pas géré à l'intérieur du clavier mais

par le BIOS. Ce n'est donc pas dans le clavier qu'une lettre actionnée est transformée en majuscule lorsque le verrouillage des majuscules (Caps Lock) est activé. Le clavier en serait en effet totalement incapable puisqu'il n'associe absolument aucun caractère à chaque touche, mais seulement un numéro de touche virtuelle. Ce n'est qu'au niveau du BIOS que ce numéro de touche est ensuite converti en un code ASCII ou en un code clavier étendu. La touche de verrouillage des majuscules Caps Lock est donc à cet égard une touche semblable aux autres. Le fait de l'actionner entraîne simplement la transmission au système du code Make correspondant. C'est le BIOS qui affecte à cette touche la fonction de verrouillage des majuscules (Caps Lock), en fixant un indicateur interne qui marque l'activation du mode et en envoyant au clavier l'instruction "Set/Reset Mode Indicators" pour que la LED correspondante soit allumée.

Ces modes de fonctionnement du clavier sont normalement activés et désactivés par l'utilisateur lorsque celui-ci actionne la touche correspondante. Il peut cependant, dans certains cas, être souhaitable d'activer un mode à l'aide d'un programme, par exemple sur les claviers disposant d'un pavé de touches de direction séparé et d'un pavé numérique. Comme ce pavé numérique, sur la plupart des claviers, ne génère des chiffres que dans le cadre du mode de verrouilage numérique (Num Lock), il peut être intéressant d'activer le mode Num Lock automatiquement dès le lancement du système.

D'après les explications précédentes, il suffit pour cela de fixer un indicateur du BIOS et d'allumer la diode correspondante sur le clavier. (Du point de vue fonctionnel cette dernière action n'est d'ailleurs pas indispensable puisqu'elle sert uniquement à avertir l'utilisateur que ce mode est activé).

Dans la pratique toutefois, un programme peut se contenter de la première opération puisque le BIOS se charge automatiquement d'allumer les diodes du clavier. En effet, chaque fois que l'une des fonctions de l'interruption clavier du BIOS est appelée, elle examine si l'état des diodes correspond à l'état réel du clavier, tel qu'il est inscrit dans une variable interne. Si une divergence apparaît, le BIOS fixe automatiquement les diodes en conformité avec l'état du clavier.

Les trois programmes suivants écrits en BASIC, Pascal et en C se chargent de cette tâche, en vous offrant des routines qui permettent de fixer les différents modes de verrouillage et d'allumer ou d'éteindre les diodes correspondantes. Précisons à ce propos que sur des claviers de PC ou d'XT pouvus de diodes, ces programmes changent simplement les modes de verrouillage mais n'ont aucun effet sur les diodes. Cela est dû au fait que ces claviers, bien que semblables extérieurement aux claviers d'AT, ne comportent qu'un simple circuit 8048 incapable de commander des diodes. L'allumage et l'extinction des diodes lorsqu'on appuie sur les touches correspondantes n'a aucun rapport avec la programmation du BIOS. Il s'agit d'un mécanisme purement matériel car ces touches sont reliées à des commutateurs Flip Flop qui ouvrent et ferment alternativement l'alimentation des diodes.

Listing : LEDB.BAS

```
'*****************************************************************
'*                          L E D B                              *
'*---------------------------------------------------------------*
'*  Fonction  : Fixe les bits de l'indicateur d'état du clavier  *
'*              du BIOS en allumant ou éteignant les diodes      *
'*              électroluminescentes                             *
'*---------------------------------------------------------------*
'*  Auteur               : MICHAEL TISCHER                       *
'*  Développé le         : 08.06.1991                            *
'*  Dernière modification : 03.01.1992                           *
'*****************************************************************
'
DECLARE SUB SetFlag (Flag AS INTEGER)
DECLARE SUB Delay (Pause AS INTEGER)
DECLARE SUB ClrFlag (Flag AS INTEGER)

'$INCLUDE: 'QB.BI'          'Fichier d'inclusion pour les interruptions

CONST SCRL = 16                                      'Bit Scroll-Lock
CONST NUML = 32                                      'Bit Num-Lock
CONST CAPL = 64                                      'Bit Caps-Lock

DIM Compteur AS INTEGER                              'Compteur d'itérations

CLS                                                  'Efface l'écran
PRINT " LEDB  -  (c) 1988, 1992 by Michael Tischer"
PRINT
PRINT " Observez les LEDs de votre clavier "

FOR Compteur = 1 TO 10                               '10 itérations
  SetFlag (CAPL)                                     'Allume CAPS
  Delay (100)                                        'Attend 100 ms
  ClrFlag (CAPL)                                     'Eteint CAPS
  SetFlag (NUML)                                     'Allume NUM
  Delay (100)                                        'Attend 100 ms
  ClrFlag (NUML)                                     'Eteint NUML
  SetFlag (SCRL)                                     'Allume Scroll-Lock
  Delay (100)                                        'Attend 100 ms
  ClrFlag (SCRL)                                     'Eteint Scroll-Lock
NEXT

FOR Compteur = 1 TO 10                               '10 itérations
  SetFlag (CAPL OR SCRL OR NUML)                     'Allume les trois indicateurs
  Delay (500)  'Attend 200 ms
  ClrFlag (CAPL OR SCRL OR NUML)                     'Eteint les trois indicateurs
  Delay (500) 'Attend 200 ms
NEXT
END

'*****************************************************************
'* ClrFLAG : Met à 0 un bit de l'indicateur d'état du clavier    *
'* Entrée  : Bit à mettre à 0 (cf Constantes)                    *
'* Sortie  : néant                                               *
'*****************************************************************
'
SUB ClrFlag (Flag AS INTEGER)

DIM Register AS RegType                              'Registres du processseur

DEF SEG = &H40          'Segment de l'indicateur d'état du clavier du BIOS
POKE &H17, (PEEK(&H17) AND (NOT Flag))                          'Mise à 0
Register.ax = 1 * 256       'AH = Num. fonction : Caractère disponible ?
CALL INTERRUPT(&H16, Register, Register)       'Déclenche l'interruption

END SUB

'*****************************************************************
'* Delay   : Arrête l'exécution du programme pendant une certain temps *
'*           donné en millisecondes                              *
'* Entrée  : Pause = temps d'arrêt en ms                         *
'* Sortie  : néant                                               *
'*****************************************************************
'
SUB Delay (Pause AS INTEGER)

DIgister AS RegType     'Registres du processeur pour l'interruption
DIM Time AS LONG                                  'Mémorise le temps final

Register.ax = 0                  'Numéro de la fonction : lecture du timer
CALL INTERRUPT(&H1A, Register, Register) 'Déclenche interruption du BIOS
Time = Register.dx + (Register.cx * 32768)        'Calcule le temps final
Time = Time + (Pause * 18 + ((Pause * 2) / 10)) / 1000
DO
  CALL INTERRUPT(&H1A, Register, Register)  'Appel interruption du timer
LOOP WHILE (Register.dx + (Register.cx * 32768)) <= Time
        'Jusqu'à ce que Time soit atteint
END SUB

'*****************************************************************
'*  SETFLAG : Met à 1 un bit de l'indicateur d'état du clavier   *
'*  Entrée  : Bit à mettre à 1 (cf constantes)                   *
'*  Sortie  : néant                                              *
'*****************************************************************
'
SUB SetFlag (Flag AS INTEGER)

DIM Register AS RegType     'Registres du processeur pour l'interruption

DEF SEG = &H40          'Segment de l'indicateur d'état du clavier du BIOS
POKE &H17, (PEEK(&H17) OR Flag)                                 'Mise à 1
Register.ax = 1 * 256       'AH = num. fonction : Caractère disponible ?
CALL INTERRUPT(&H16, Register, Register) 'Déclenche interruption du BIOS

END SUB
```

Listing : LEDP.PAS

```
{*****************************************************************}
{*                          L E D P                              *}
{*---------------------------------------------------------------*}
{*  Fonction      : Fixe les bits de l'indicateur d'état du clavier *}
{*                  du BIOS en allumant ou éteignant les diodes   *}
{*                  électroluminescentes                          *}
{*---------------------------------------------------------------*}
{*  Auteur               : MICHAEL TISCHER                       *}
{*  Développé le         : 16.08.1988                            *}
{*  Dernière modification : 03.01.1992                           *}
{*****************************************************************}

program LEDP;

uses CRT,                                        { Inclut l'unité CRT }
     DOS;                                        { Inclut l'unité DOS }

const SCRL =  16;                                { Bit Scroll-Lock}
      NUML =  32;                                { Bit Num-Lock}
      CAPL =  64;                                { Bit Caps-Lock}
      INS  = 128;                                { Bit Insert}

{*****************************************************************}
{* SETFLAG : Met à 1 des bits de l'indicateur d'état du clavier  *}
{* Entrée  : Bits à mettre à 1 ( cf constantes )                 *}
{* Sortie  : néant                                               *}
{*****************************************************************}

procedure SetFlag(Flag : byte);

var BiosTSByte : byte absolute $0040:$0017;{ Indicateur état du clavier}
    Regs       : Registers;              { Registres pour l'interruption }

begin
  BiosTSByte := BiosTSByte or Flag;        { Met à 1 les bits indiqués }
  Regs.AH := 1;        { Numéro de la fonction : caractère disponible ? }
  intr($16, Regs);                          { Déclenche l'interruption}
end;

{*****************************************************************}
{* CLRFLAG : Met à 0 des bits de l'indicateur d'état du clavier  *}
{* Entrée  : Bits à mettre à 0 (cf constantes )                  *}
{* Sortie  : néant                                               *}
{*****************************************************************}

procedure ClrFlag(Flag : byte);
```

```
!
!var BiosTSByte : byte absolute $0040:$0017;{ Indicateur état du clavier}
!    Regs       : Registers;              { Registres pour l'interruption }
!
!begin
!  BiosTSByte := BiosTSByte and ( not Flag );        { Met à 0 les bits }
!  Regs.AH := 1;       { Numéro de la fonction : caractère disponible ? }
!  intr($16, Regs);                  { Déclenche l'interruption du BIOS }
!end;
!
!{*********************************************************************}
!{**                  PROGRAMME PRINCIPAL                            **}
!{*********************************************************************}
!
!var compteur : integer;
!
!begin
!  writeln('LEDP  -  (c) 1988, 92 by Michael Tischer');
!  writeln(#13,#10, 'Observez les LEDs de votre clavier !');
!
!  for compteur:=1 to 10 do                              { 10 itérations }
!    begin
!      SetFlag( CAPL);                                    { Allume CAPS }
!      Delay( 100 );                                    { Attend 100 ms }
!      ClrFlag( CAPL );                                   { Eteint CAPS }
!      SetFlag( NUML);                                    { Allume NUM  }
!      Delay( 100 );                                    { Attend 100 ms }
!      ClrFlag( NUML );                                     { Eteint NUM }
!      SetFlag( SCRL);                                { Allume SCROLL-LOCK }
!      Delay( 100 );                                    { Attend 100 ms }
!      ClrFlag( SCRL );                               { Eteint SCROLL-LOCK }
!    end;
!
!  for compteur:=1 to 10 do                              { 10 itérations }
!    begin
!      SetFlag(CAPL or SCRL or NUML);    { Allume les trois indicateurs }
!      Delay( 500 );                                    { Attend 200 ms }
!      ClrFlag(CAPL or SCRL or NUML);          { Eteint les indicateurs }
!      Delay( 500 );                                    { Attend 200 ms }
!    end;
!end.
```

Listing : LEDC.C

```
!/*********************************************************************/
!/*                            L E D C                                */
!/*-------------------------------------------------------------------*/
!/*   Fonction      : Fixe les bits de l'indicateur d'état du clavier */
!/*                   du BIOS en allumant ou éteignant les diodes     */
!/*                   électroluminescentes                            */
!/*-------------------------------------------------------------------*/
!/*   Auteur        : MICHAEL TISCHER                                 */
!/*   Développé le  : 22.08.1988                                      */
!/*   Dernière modification : 03.01.1992                              */
!/*-------------------------------------------------------------------*/
!/*   Modèle de mémoire : SMALL                                       */
!/*********************************************************************/
!
!/*-- Fichiers d'inclusion ---------------------------------------------*/
!
!#include <stdio.h>
!#include <dos.h>
!#include <bios.h>
!
!/*-- Macros -----------------------------------------------------------*/
!
!#ifndef MK_FP                          /* MK_FP est-il défini ? */
!  #define MK_FP(seg, ofs) \
!  ((void far *) ((unsigned long) (seg)<<16|(ofs)))
!#endif
!
!/*-- BIOS_KBF crée un pointeur sur l'indicateur du clavier ----------*/
!
!#define BIOS_KBF ((unsigned far *) MK_FP(0x40, 0x17))
!
!#define TICKS(ms) ((ms*10+549) / 550 )
!      /*Convertit les millisecondes en tops d'horloge */
!
!/*-- Constantes -------------------------------------------------------*/
!
!#define SCRL  16                                /* Bit Scroll Lock */
!#define NUML  32                                  /* Bit Num-Lock */
!#define CAPL  64                                 /* Bit Caps-Lock */
!#define INS  128                                    /* Bit Insert */
!
!#ifdef __TURBOC__                       /* Définitions pour TURBO C */
!   #define GetBiosTime(x)    ( x = biostime( 0, NULL ) )
!#else                  /* Definitions pour le compilateur Microsoft C*/
!   #define GetBiosTime(x)    ( _bios_timeofday( _TIME_GETCLOCK, &x) )
!#endif
!
!/*********************************************************************/
!/*  Delay  : Arrête l'exécution du programme pendant un certain      */
!/*           temps indépendamment de la vitesse du système           */
!/*  Entrée : PAUSE = temps d'arrêt décompté en tops d'horloge        */
!/*  Sortie : néant                                                   */
!/*  Info   : un top d'horloge = 1/18,2 secondes                      */
!/*********************************************************************/
!
!void delay( unsigned int pause )
!{
!  long temps,                                    /*Temps instantané */
!       tempslimite;                               /* Temps fixé */
!
!  if ( pause )                                        /* Pause # 0 ? */
!  {                                                        /* Non */
!    GetBiosTime( tempslimite );
!    tempslimite += (long) pause;          /* Calcule le temps limite */
!
!    do                    /* Boucle d'attente, lit le temps instantané */
!      GetBiosTime( temps );
!    while ( temps < tempslimite );          /* Temps limite atteint ? */
!  }                                                    /* Oui, terminé */
!}
!
!/*********************************************************************
!*          : S E T _ F L A G                                        *
!**-----------------------------------------------------------------**
!*  Fonction : Met à 1 des bits de  l'indicateur d'état du clavier   *
!*  Entrée   : FLAG = Bits à mettre à 0                              *
!*  Sortie   : néant                                                 *
!*********************************************************************/
!
!void set_flag( unsigned flag )
!{
!  union REGS regs;                  /* Mémorise le contenu des registres */
!
!  *BIOS_KBF |= flag;                          /* Met à 1 les bits indiqués */
!  regs.h.ah = 1;      /* Numéro de la fonction : caractère disponible ? */
!  int86(0x16, &regs, &regs);          /* Déclenche l'interruption du BIOS*/
!}
!
!/*********************************************************************
!*                  C L R _ F L A G                                  *
!**-----------------------------------------------------------------**
!*  Fonction : Met à 0 des bits de l'indicateur du clavier           *
!*  Entrée   : FLAG = Bits à mettre à 0.                             *
!*  Sortie   : néant                                                 *
!*********************************************************************/
!
!void clr_flag( unsigned flag )
!{
!  union REGS regs;                  /* Mémorise le contenu des registres */
!
!  *BIOS_KBF &= flag;                          /* Met à 0 les bits indiqués */
!  regs.h.ah = 1;      /* Numéro de la fonction : caractère disponible ? */
!  int86(0x16, &regs, &regs);          /* Déclenche l'interruption du BIOS */
!}
!
!/*********************************************************************/
!/**                      PROGRAMME PRINCIPAL                        **/
!/*********************************************************************/
!
!void main()
!{
!  unsigned i;                                  /* Compteur d'itérations */
!
!  printf( "LEDC  -  (c) 1988, 1992 by Michael Tischer\n\n");
!  printf( "Observez les LEDs de votre clavier !\n");
!
!  for (i=0; i<10; ++i)                              /* 10 itérations */
!  {
```

```
!   set_flag( CAPL );                   /* Allume CAPS */
!   delay( TICKS(100) );          /* Attend 100 millisecondes */
!   clr_flag( CAPL );                   /* Eteint CAPS */
!   set_flag( NUML);                    /* Allume NUM */
!   delay( TICKS(100) );           /* Attend 100 millisecodes */
!   clr_flag( NUML );                   /* Eteint NUM */
!   set_flag( SCRL);                /* Allume SCROLL-LOCK */
!   delay( TICKS(100) );          /* Attend 100 millisecondes */
!   clr_flag( SCRL );               /* Eteint SCROLL-LOCK */
! }
```

```
!
! for (i=0; i<10; ++i)                                  /* 10 itérations */
! {
!   set_flag(CAPL | SCRL | NUML);     /* Allume les trois indicateurs */
!   delay( TICKS(500) );                                /* Attend 200 ms */
!   clr_flag(CAPL | SCRL | NUML);     /* Eteint les trois indicateurs */
!   delay( TICKS(500) );                                /* Attend 200 ms */
! }
!}
```

6. Disquettes et disques durs

Ce chapitre est consacré aux disquettes et aux disques durs qui sont, dans le cadre de la programmation sur PC, au moins aussi importants que la carte vidéo ou le clavier. Mais il existe de grandes différences entre ces périphériques : alors que l'on n'a que très rarement recours aux fonctions du DOS lors de l'accès au clavier ou à l'écran et que l'on utilise plutôt les fonctions du BIOS, c'est exactement le contraire qui se produit dans le cas des disquettes et des disques durs.

En effet, le BIOS intervient principalement lors du formatage physique des disquettes et des disques durs ainsi que dans le cas d'accès individuels à certains secteurs. Ceci n'est toutefois pas recommandé dans le cas d'une programmation normale sous DOS. En effet, il s'agit le plus souvent de manipuler des fichiers, ce qui est impossible sans l'intervention du DOS en tant qu'entité de gestion.

Les seules exceptions sont les utilitaires pour disques, tels les programmes PC Tools ou les utilitaires Norton. Nous partons toutefois du principe que vous n'avez nullement l'intention de vous lancer dans le développement d'utilitaires de ce type ; le cas échéant, vous serez très certainement amené à évaluer l'ampleur du travail en homme-mois plutôt qu'en homme-journées.

Nous aborderons néanmoins dans ce chapitre, entre autres choses, l'accès à une disquette à l'aide des fonctions BIOS. Nous ne décrirons toutefois pas les procédures de programmation directes du contrôleur de disquette et de disques durs, et ce pour une raison très simple. L'accès direct à ces contrôleurs entraîne en général bon nombre de problèmes avec des contrôleurs incompatibles. De plus, la programmation directe ne présente aucun avantage du point de vue de la rapidité vu que la plus grande partie du temps est consacrée aux accès disques. Nous vous recommandons donc de ne pas tenter de programmer directement les contrôleurs de disquettes ou de disques durs.

Mises à part les fonctions BIOS, ce chapitre traitera d'autres sujets relatifs au disque dur. Nous étudierons notamment la manière dont les données sont enregistrées sur le disque dur ainsi que la fonction des différents contrôleurs de disques durs que l'on trouve sur les PC.

Vous trouverez pour finir un chapitre consacré au partitionnement du disque dur en plusieurs unités logiques. Mais jetons préalablement un regard sur les particularités communes aux disquettes et aux disques durs.

6.1. Structure des disquettes et des disques durs

Disquettes et disques durs ont en commun leur structure. Celle-ci est prise en compte par différentes fonctions du BIOS affectées aux accès aux lecteurs de disquettes et de disques durs. Etudions dans un premier temps la structure de base des disquettes et des

disques durs et commençons par celle des disquettes qui, du point de vue de leur structure, peuvent être assimilées à des disques durs réduits (à deux dimensions).

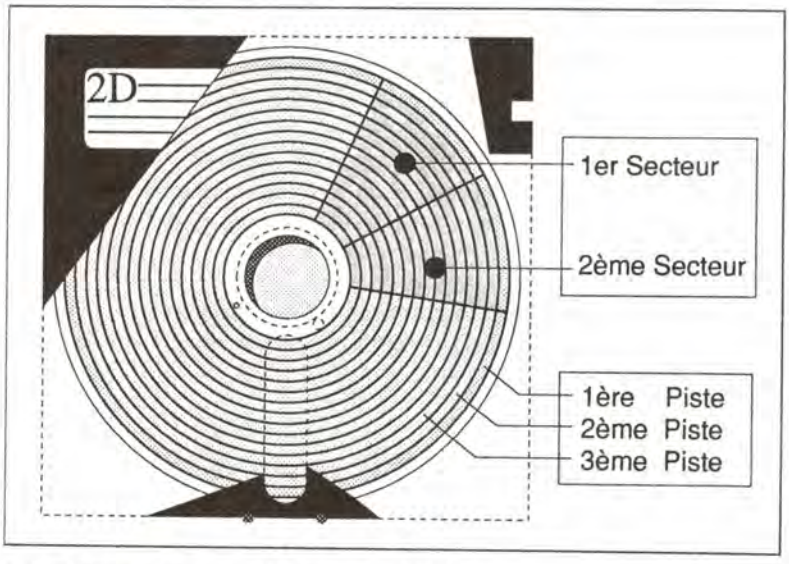
Structure d'une disquette

Les disquettes sont divisées en pistes, sorte de cercles concentriques répartis à intervalles réguliers sur leur surface magnétique. Ces pistes sont numérotées de 0 à N, N représentant le nombre total de secteurs moins 1 et pouvant varier selon le format de la disquette. La piste la plus extérieure porte systématiquement le numéro 0, la suivante le numéro 1 et ainsi de suite jusqu'à la piste située le plus à l'intérieur.

Chaque piste est divisé en un nombre constant de secteurs de taille égale. Le nombre de ces secteurs dépend du format de la disquette et du lecteur. A la différences des pistes, les secteurs ne sont pas numérotés à partir de 0, mais à partir de 1 jusqu'à N, N représentant le nombre de secteurs par piste.

Chaque secteur contient 512 octets et représente la plus petite unité d'allocation à laquelle les programmes ont accès. Il n'est donc pas possible de lire/écrire un seul octet d'une disquette ; seul un secteur entier peut être lu ou écrit.

Les informations sont inscrites dans un secteur selon la procédure FM ou MFM, également mises en oeuvre dans le cas du disque dur et décrites dans la cestion 6.6. Il n'est toutefois pas nécessaire d'en tenir compte en tant que programmeur, sachant que l'on n'aura affaire qu'aux données binaires qui doivent être lues ou écrites dans un secteur.



Structure d'une disquette, exemple disquette 5"1/4

En fonction du nombre de pistes et de secteurs qu'elle contient, la capacité d'une disquette s'évalue d'après la formule suivante :

```
Nombre de pistes * nombre de secteurs par piste * 512 octets par
secteur.
```

La valeur ainsi obtenue ne traduit toutefois que la capacité d'une seule face de la disquette et doit donc être multipliée par deux dans le cas d'un lecteur de disquettes équipé de deux têtes de lecture/écriture. Dans ce cas, la face supérieure et la face inférieure de la disquette seront toutes deux utilisées pour le stockage des données. Elles portent respectivement le numéro 0 et le numéro 1, ce qui permet de les différencier au cours de la programmation.

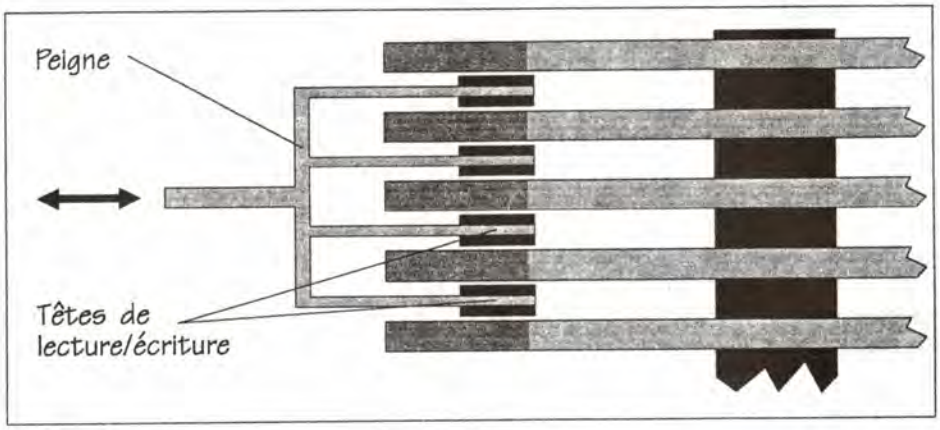
Le nombre de secteurs par piste détermine également le débit de transfert de données qui sert à mesurer la vitesse d'exécution du lecteur ainsi que celle du contrôleur qui y est associé. A raison d'une vitesse de rotation constante de 300 rotations par minute, plus le volume de bits qui transite par la tête de lecture/écriture est élevé, plus le nombre de secteurs inscrits sur une piste sera important.

Structure d'un disque dur

Le débit de transfert de données est approximativement dix fois plus élevé dans le cas d'un disque dur, la vitesse de rotation étant elle-même dix fois plus élevée. Mais ce débit de transfert peut être une nouvelle fois multiplié par dix car les disques durs récents sont aujourd'hui capables d'inscrire presque une centaine de secteurs par piste, et ce même dans le cas de disquettes 3"1/2.

Mais cela ne modifie en rien la structure fondamentale d'un disque dur qui se présente sous la forme de plateaux superposés. En effet, un lecteur de disque dur se compose tout simplement de plusieurs plateaux magnétiques superposés lesquels sont, tout comme les disquettes, divisés en pistes et en secteurs. Chaque plateau dispose de deux têtes de lecture/écriture capables de traiter les deux faces du plateau.

Elles sont rendues solidaires des autres têtes par l'intermédiaire du bras de lecture/écriture de telle manière qu'un changement de piste s'effectuera simultanément sur l'ensemble des plateaux. C'est d'ailleurs pour cette raison que, au cours de l'enregistrement de données sur le disque dur, il est préférable de ne pas passer de piste en piste mais de passer d'une tête à une autre, jusqu'à ce que la piste concernée soit complètement enregistrée pour chaque plateau. En effet, le passage d'une tête à une autre s'effectue beaucoup plus rapidement car il ne met pas en jeu de déplacements mécaniques, ce qui n'est pas le cas lors d'un changement de piste.



Structure d'un disque dur

Pour décrire ce mouvement de tête à tête d'une piste, on utilise le concept de "cylindre". Un cylindre englobe l'ensemble des pistes qui portent le même numéro mais qui se trouvent sur différents plateaux. Et il s'agit en fait de la seule différence d'importance par rapport à la structure d'une disquette.

6.2. Lecteurs et formats de disquettes

Lorsqu'on souhaite accéder à des disquettes à l'aide du BIOS, il est nécessaire (ce qui n'est pas le cas dans le cadre de la programmation DOS) de se familiariser tout d'abord avec les différents formats de disquettes qui existent sur PC. On différencie tout d'abord les lecteurs 5"1/4 des lecteurs 3"1/2 ainsi que les disquettes simple densité, double densité et haute densité.

Les lecteurs et les disquettes 5"1/4

Le règne des lecteurs de 8" prit fin avec l'avènement des PC. En raison de l'obsolescence de ce type de lecteurs, les PC furent donc d'emblée équipés de lecteurs 5"1/4 lesquels font, aujourd'hui encore, partie de l'agencement standard des micro-ordinateurs. Ces lecteurs supportent les disquettes "Haute densité". Cette désignation fut choisie afin d'établir une différence avec les disquettes "Simple densité" utilisées auparavant sur de nombreux micro-ordinateurs.

Les disquettes "simple densité" ne comportaient que quatre secteurs par pistes et 40 secteurs par face, ce qui correspond à une capacité de 80 Ko par face ou de 160 Ko dans le cas de lecteurs de disquettes équipés de deux têtes de lecture/écriture. Il en

résultait un débit de transfert de données (125 Ko/s) peu élevé, notamment si on le compare aux débits d'aujourd'hui.

Les disquettes "double densité" comptent huit secteurs par piste, soit le double des disquettes "simple densité", le nombre de pistes par face restant quant à lui inchangé. La capacité de ces disquettes est donc du double, passant à 160 Ko dans la cas d'un lecteur simple face et à 320 Ko pour les lecteurs double face. Ceci vaut également pour le débit de transfert de données qui, dans le cas de disquettes "double densité", passe à 250 Ko/s.

Mais ce format ne perdura guère ; en effet, à raison de 8 secteurs par piste, une piste d'une disquette double densité n'est pas complètement remplie. Il reste encore suffisamment de place pour un neuvième secteur. On obtient alors le fameux format de disquette de 360 Ko que le DOS utilise également.

A la suite des formats "simple" et "double densité", on vit arriver, avec l'avènement des PC/AT, le format "haute densité" qui permet une capacité de 1,2 Mo pour les lecteurs 5"1/4. Le nombre de secteurs par piste passe de huit à quinze et le nombre de pistes par face double, passant ainsi à 80. Tout comme pour les disquettes double densité, les deux faces de la disquette sont utilisées.

En théorie, on aurait pu obtenir jusqu'à 16 secteurs car les lecteurs des AT permettent un débit de transfert de données de 500 Ko/s. Les fabricants ont toutefois craint que la couche magnétique, qu'il eût alors fallu également doubler, ne supporte pas la vitesse de rotation du moteur des lecteurs de disquettes, celle-ci passant en effet de 300 à 360 rotations. Ceci a pour effet de réduire le nombre théorique de secteurs par piste de 16 à 15.

Format de disquette 5"1/4

Type	Lecteur	Secteurs par piste	Pistes par face	Capacité	Débit de transfert de données
Double densité	PC/XT	8	40	160/320 Ko	250 Ko/s
Double densité	PC/XT	9	40	180/360 Ko	250 Ko/s
Haute densité	AT	15	80	1,2 Mo	500 Ko/s

Du fait que les pistes des disquettes haute densité sont plus nombreuses (80 au lieu de 40) et donc plus denses, ce format ne peut pas être lu par les lecteurs des anciens PC et des XT mais uniquement par les nouveaux lecteurs MF des AT. MF signifie "Multifonction" ; en effet, ce type de lecteur est capable de traiter les disquettes au format double densité de 360 Ko. La vitesse de rotation peut donc, à l'instar du débit de transfert de données, s'adapter à ce format à raison de 300 rotations par minute.

L'augmentation du nombre de secteurs par piste empêche lui aussi le traitement de disquettes haute densité par les anciens lecteurs des PC/XT. Ceux-ci ne peuvent en effet atteindre le débit de transfert de données de 500 Ko/s requis, une incapacité structurelle qui ne peut être modifiée.

Toutefois, l'augmentation de la capacité d'enregistrement, passant de la double à la haute densité, na dépend pas uniquement des caractéristiques électroniques des lecteurs. Il s'agit avant tout d'une différence dans la "granulation" du matériau magnétique utilisé. Plus la taille des différentes particules magnétiques est réduite, plus elles pourront absorber d'informations. C'est d'ailleurs pour cette raison que les disquettes double densité ne peuvent être correctement formatées par les lecteurs des micro-ordinateurs de type AT ; leur granulation n'est pas assez fine.

Certes, le formatage de la plupart des disquettes est possible permettant d'obtenir une capacité de 600 Ko, voire davantage. Mais celles-ci ne pourront toutefois être lues que sur les PC sur lesquels elles ont été formatées. Les autres PC affichent en revanche un message d'erreur en lecture, rendant ainsi l'utilisation de la disquette quasi impossible. Il s'agit ici d'un phénomène dont on doit tenir compte, même dans le cas de formatages tout à fait orthodoxes, à savoir les différents positionnements des têtes de lecture/écriture sur les différents types de lecteurs de disquettes.

Il ne s'agit pourtant que de différences infimes, inférieure à un millimètre, mais qu'il importe néanmoins de neutraliser en augmentant la largeur des différentes pistes de façon à ce qu'elles soient plus larges que la tête de lecture/écriture, laquelle devra être positionnée au centre présumé de la piste concernée. Les informations pourront ainsi être lues correctement.

Cela ne fonctionne toutefois pas dans le cas de disquettes double densité "surformatées", car la position de la tête de lecture/écriture du lecteur affecté au formatage décide si une piste sera lue de manière aléatoire -et si elle est intacte- ou non. Tel ne sera pas le cas si la position de la tête de lecture/écriture d'un autre lecteur varie un tant soit peu de celle du lecteur initial ; le cas échéant, elle ratera la piste concernée d'un cheveux.

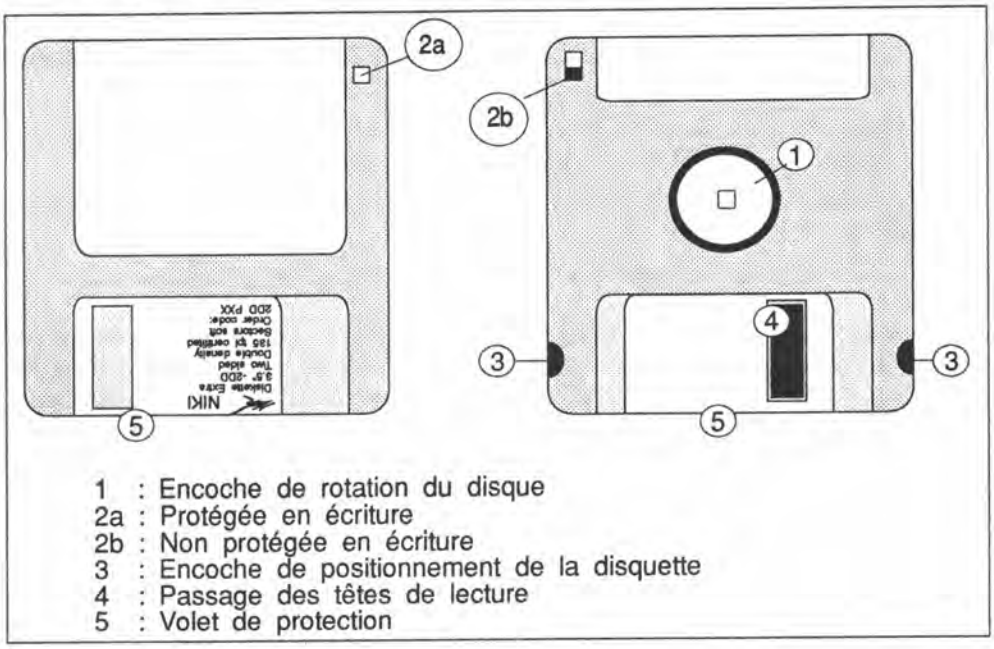
Un phénomène analogue se produit lorsqu'on entreprend de formater des disquettes haute densité à partir de lecteurs double densité. La procédure échouera dans la plupart des cas ou provoquera au minimum des erreurs de lecture.

Conclusion : ceux qui pensent améliorer les performances des lecteurs de disquettes de leur PC/XT en achetant des disquettes HD (Haute Densité), lesquelles présentent de plus l'inconvénient d'être chères, se trompent.

Les disquettes et les lecteurs 3"1/2

Bien que les lecteurs de disquettes 5"1/4 soient encore très répandus, les lecteurs de disquettes plus petits 3"1/2 ont de plus en plus tendance à les supplanter ; leur

introduction massive, au départ limitée au micro ordinateur portable, touche aujourd'hui toutes les catégories de micro- ordinateurs. Ces disquettes présentent tout d'abord l'avantage d'être de taille réduite. D'autre part, elles offrent une plus grande stabilité en raison de leur enveloppe rigide. De plus, la fenêtre d'accès recouverte d'un clapet amovible métallique ne nécessite aucune protection particulière.



1 : Encoche de rotation du disque
2a : Protégée en écriture
2b : Non protégée en écriture
3 : Encoche de positionnement de la disquette
4 : Passage des têtes de lecture
5 : Volet de protection

Structure d'une disquette 3"1/2

La différence entre Double Densité et Haute Densité existe également pour les lecteurs et les disquettes 3"1/2. La double densité correspond à une capacité de 720 Ko ; dans ce cas, chaque piste compte neuf secteurs et chaque face de disquette (les deux faces sont utilisées) comprend 80 pistes. La densité des pistes est donc ici beaucoup plus importante que dans le cas de disquette 5"1/4.

Les premiers lecteurs 3"1/2 utilisés dans le monde PC ne supportaient que ce format. Ils restaient de ce fait limités à une capacité de 720 Ko. Les disquettes Haute Densité permettent quant à elles une capacité double, soit 1,44 Mo, et les lecteurs correspondant au format 3"1/2 font aujourd'hui partie de l'équipement standard des micro- ordinateurs. Ces lecteurs n'utilisent également que 80 pistes par face de disquettes ; chaque piste comporte toutefois 18 secteurs. Le débit de transfert de données se trouve ainsi multiplié par deux. De ce fait, ces disquettes ne peuvent pas être utilisées dans les lecteurs 3"1/2 double densité des anciens PC/XT.

De la même manière que pour les lecteurs 5"1/4, les lecteurs 3"1/2 capables de traiter les disquettes au format haute densité peuvent s'adapter aux disquettes double densité, c'est-à-dire les lire, les formater et y inscrire des informations.

Format de disquette 3"1/2					
Type	Lecteur	Secteurs par piste	Pistes par face	Capacité	Taux de transfert de données
Double Densité	PC/XT	9	80	720 Ko	250 KBit/s
Haute Densité	AT	18	80	1,44 Mo	500 KBit/s
Extra haute densité	AT	36	80	2,88 Mo	1 MBit/s

La tendance actuelle concernant une utilisation de plus en plus répandue du format 3"1/2 est confirmée par l'arrivée des disquettes 3"1/2 de haute capacité pouvant contenir 2,88 Mo d'informations et qui portent le nom de disquettes "extra haute densité" (ED). Le nombre de secteurs, égal à 36, est une fois encore multiplié par deux. Ces disquettes ne peuvent donc être lues qu'à partir de lecteurs ED. A l'instar des lecteurs HD, ceux-ci ont une compatibilité ascendante de telle manière qu'ils sont capables de traiter tous les formats de disquettes précédents.

Les lecteurs qui acceptent différents formats de disquettes doivent tout d'abord communiquer le format de la disquette traité au BIOS. Dans le cas de disquettes 5"1/4, cette procédure n'est pas sans poser de problèmes car cette information ne peut être transmise que suite à un accès en lecture, lequel accès implique bien évidemment que la disquette soit préalablement formatée. Ce n'est pas le cas pour les disquettes 3"1/2 dont le format est d'emblée identifiable par un petit trou qui se trouve à la hauteur du clapet de protection.

Tout comme pour le trou de protection en écriture, l'existence de ce trou peut être aisément repérée à l'intérieur du lecteur à l'aide de la diode lumineuse et de la cellule photo-électrique dont il est équipé. De manière conventionnelle, ce trou n'existe pas sur les disquettes double densité alors qu'il est toujours présent sur les disquettes haute densité. On le trouve également sur les disquettes extra haute densité ; il est toutefois placé un peu plus bas afin de pouvoir différencier ces disquettes des disquettes haute densité.

Les lecteurs de disquettes et leur contrôleur

Un lecteur de disquette se compose d'une partie mécanique qui permet la rotation de la disquette à raison de 300 rotations par minute (360 pour les disquettes 5"1/4" HD) ainsi que le déplacement de la tête de lecture/écriture. De plus, le lecteur de disquettes

héberge une partie électronique sous la forme d'un séparateur de données qui transforme les variations du signal électrique en un flux de données binaires.

La gestion proprement dites du lecteur s'effectue toutefois à l'aide d'un contrôleur de disquettes à part qui se présente, en règle générale, sous la forme d'une carte d'extension placée dans l'un des slots de l'ordinateur. Alors que les contrôleurs de disquettes sur les PC/XT gèrent uniquement l'accès aux différents lecteurs de disquettes, au nombre maximum de 4, les contrôleurs des disquettes des PC/AT sont associés aux contrôleurs des disques durs sur une même carte. On parle alors de contrôleurs mixtes capables de gérer deux disques durs et seulement deux lecteurs de disquettes. Au coeur d'un tel contrôleur de disquettes, on trouve un micro-processeur de type NEC PD765 ou un micro-processeur équivalent d'un autre fabricant et compatible avec celui précédemment nommé. En effet, la compatibilité d'un registre avec le micro-processeur NEC, utilisé par IBM pour la fabrication de ces lecteurs de disquettes, est indispensable pour une parfaite collaboration avec le BIOS en ROM. C'est en effet ce dernier qui gère l'accès au lecteur de disquettes à partir de ce micro-processeur.

Il est certes tout à fait envisageable d'affecter le BIOS en ROM à un autre contrôleur de disquettes. Le micro-processeur de NEC de type NEC PD765 s'est toutefois établi depuis longtemps comme standard.

Et c'est justement de là que viennent les problèmes avec les nouveaux lecteurs de disquettes ED dont la taille des pistes (36 au lieu de 18 secteurs, soit le double) ainsi que le débit de transfert de données, également multiplié par deux, ne sont pas connus du BIOS. Il est ici nécessaire d'associer à la carte contrôleur une extension ROM qui s'intégrera lors de chaque lancement du système au BIOS en ROM et gérera elle même l'accès à ces lecteurs. Ceux-ci ne fonctionnent toutefois que temps que l'ordinateur fonctionne en mode protégé. Le BIOS n'est dans ce cas pas opérationnel vu qu'il est conçu exclusivement pour le mode réel.

Un système d'exploitation du type UNIX ou OS/2 devra à ce moment reprendre la programmation du contrôleur de disquettes. La procédure de gestion des derniers nés des lecteurs de disquettes échouera alors, à moins que des drivers adaptés soient disponibles. Il ne nous reste donc plus qu'à souhaiter qu'un standard BIOS voit le jour le plus rapidement possible concernant l'accès au lecteur de disquettes ED afin de ne plus être contraint d'associer à chaque contrôleur des extensions ROM. Ceci vaut d'autant plus que WINDOWS se chargera à l'avenir d'exécuter de plus en plus de tâches incombant habituellement au BIOS. Le problème des lecteurs ED, destinés à être de plus en plus répandus, se posera donc pour de nombreux utilisateurs PC.

6.3. Accès aux disquettes avec le BIOS

Le BIOS dispose de toute une série de fonctions concernant l'accès aux disquettes qui peuvent être appelées par l'interruption 13h. Cette interruption joue en plus le rôle d'interface pour les fonctions disque dur du BIOS. En règle générale, les fonctions

disquettes et disques durs de même nature portent un numéro de fonction identique. La différenciation s'effectue dans ce cas par l'indication du lecteur qui doit être chargé avant le lancement de la fonction dans le registre DL.

Concernant le lecteur de disquette, on choisira entre la valeur 0 (lecteur A) ou 1 (lecteur B). Certains contrôleurs de disquettes capables de gérer 4 lecteurs de disquettes proposent une extension du BIOS qui accepte les valeurs 2 et 3 pour les deux autres lecteurs de disquettes. En revanche, les disques durs sont représentés par les valeurs 80h et 81h.

Il est important de différencier les fonctions du BIOS PC/XT de celles des PC/AT. En effet, avec l'arrivée des lecteurs de disquettes MF, certaines fonctions BIOS supplémentaires ont du être introduites, permettant de gérer les nouvelles fonctionnalités de ce type de lecteurs.

Le tableau suivant récapitule de manière claire les fonctions disquettes de l'interruption 13h du BIOS.

Fonctions disquette de l'interruption 13h du BIOS

Nº	Fonction	PC/XT	AT
00h	Réinitialisation	Oui	Oui
01h	Lecture de l'état	Oui	Oui
02h	Lecture	Oui	Oui
03h	Ecriture	Oui	Oui
04h	Vérification	Oui	Oui
05h	Formatage	Oui	Oui
08h	Lecture des paramètres disque	Oui	Oui
15h	Lecture du type de disque	Non	Oui
16h	Détection ouverture lecteur	Non	Oui
17h	Détection format disquette	Non	Oui
18h	Détection format disquette	Non	Oui

D'ailleurs, rien ne s'oppose à implémenter les fonctions AT sur les XT à condition que ceux-ci soient équipés de lecteurs MF. Il est toutefois difficile de prévoir si l'un des nombreux fabricants de PC mettra un jour cette théorie en pratique.

Si vous regardez bien le tableau précédent, vous constaterez que les fonctions 17h et 18h sont affectées à la même tâche. Il ne s'agit en aucun cas d'une faute de frappe. Cet état de fait rend compte des difficultés rencontrées par l'introduction des disquettes haute densité 3"1/2 dans l'univers du BIOS en ROM. La fonction 17h n'était en effet pas du tout conçue pour faire face à ces nouvelles disquettes et à donc dû être remplacée par

une nouvelle fonction. Nous étudierons ce point plus en détail lorsque nous aborderons le formatage des disquettes.

L'état du lecteur

Parallèlement à l'indication du numéro du lecteur dans le registre DL, la communication d'un code d'état ou d'erreur dans le registre AH constitue un autre point commun entre les différentes fonctions. Une erreur est alors différenciée par une valeur différente de zéro et est accompagnée d'un rapport d'erreur (Carry-Flag).

Codes d'état et d'erreur des fonctions disquette du BIOS

Code	Signification
00h	Pas d'erreur
01h	Appel de fonction invalide
02h	Marque d'adresse non trouvé
03h	Tentative d'écriture sur disquette protégée
04h	Secteur non trouvé
06h	La disquette a été changée
08h	Débordement DMA
09h	Dépassement de la limite de segment lors de transfert de données
10h	Erreur de lecture
20h	Erreur du contrôleur de disquette
40h	Piste non trouvée
80h	Erreur de Time-Out, le lecteur ne réagit pas

On peut à tout instant connaître l'état de la disquette à l'aide de la fonction 01h qui n'a d'ailleurs d'autre objet que de transmettre cette information. Avant le lancement de la fonction, placez le numéro de la fonction 01h dans le registre AH et le numéro d'identification du lecteur dans le registre DL. Après avoir lancé la fonction, vous obtenez comme à l'accoutumé l'état dans le registre AH.

Réinitialisation du lecteur de disquettes

Si, après le lancement d'une fonction disquette, vous constatez, à l'aide du code d'état ou d'erreur, qu'une erreur s'est produite, il est alors préférable de réinitialiser le lecteur de disquettes à l'aide de la fonction 00h. Celle-ci n'attend rien d'autre que le numéro

de la fonction (AH) et l'indicateur du lecteur (DL). Mais elle communiquera de son côté l'état actuel du lecteur dans le registre AH.

Que vous indiquiez 0 ou 1 pour identifier le lecteur, ceci n'a aucune importance, car une réinitialisation s'effectue systématiquement sur l'ensemble des lecteurs de disquettes. La valeur indiquée n'est toutefois pas sans incidence sur le registre DL ; en effet, si vous saisissez une valeur supérieure à 80h, la réinitialisation affectera les disques durs connectés au lieu des lecteurs de disquettes.

Détection du type de disque

Un programme ne sait pas toujours d'emblée à quel type de disque il à affaire, pas plus d'ailleurs que le format de disquettes supporté par celui-ci. Les fonctions 08h et 15h permettent toutefois de communiquer ces informations. La première de ces deux fonctions existe déjà dans le BIOS des PC/XT et sert à différencier les différents formats de lecteurs et de disquettes. Lors de son lancement, elle attend uniquement le numéro de la fonction dans le registre AH et le numéro du lecteur dans le registre DL. Mais elle retransmet toute une série d'informations telle que nous le montre l'illustration suivante.

Comme d'habitude, le sémaphore est positionné suite au lancement de la fonction et un code d'erreur est alors inscrit dans le registre AH pour le cas ou le lancement de la fonction aurait échoué. Il est ainsi possible de constater si un lecteur particulier est installé, ainsi que la présence d'un deuxième, troisième, voire d'un quatrième lecteur.

Informations transmises par la fonction 08h

Registre	Informations
BL	Type de lecteur 01h = 5"25, 360 Ko 02h = 5"25, 1,2 Mo 03h = 3"5, 720 Ko 04h = 3"5, 1,44 Mo
DH	Plus grand numéro de face (toujours = 1)
CH	Plus grand numéro de piste
CL	Plus grand numéro de secteur
ES:DI	Pointe sur DDPT (Table des Paramètres Disque)

La valeur la plus intéressante est certainement contenue dans le registre DL qui indique non seulement le format du lecteur (3"1/2 ou 5"1/4) mais également le format des disquettes (DD ou HD). Toutefois, la description ne s'applique pas nécessairement au format de la disquette insérée mais indique le plus grand format possible.

Cette information provient de la mémoire CMOS dans laquelles ces informations ont été enregistrées lors de la configuration à partir du Setup de l'ordinateur. L'attribution des numéros de code est standardisée mais ne comprend toutefois pas encore les lecteurs ED 3"1/2. Il est toutefois fort probable que ces lecteurs porteront à l'avenir le numéro de code 05h, ce qui paraît le plus logique.

Le code des lecteurs permet de déduire automatiquement le nombre de secteurs, de pistes et de têtes ; ces informations sont toutefois indiquées explicitement une nouvelle fois dans les registres CH/CL, DH/DL.

La Table des Paramètres Disque (DDPT), référencée par le pointeur dans les deux registres ES:DI, contient les paramètres nécessaires au BIOS pour la programmation du contrôleur de disquettes. Vous trouverez une description de cette table au cours de ce chapitre.

La fonction 15h n'est supportée que par les PC/AT et leur lecteur FM. En effet, à l'inverse des PC/XT, ceux-ci sont capables d'identifier un changement de disquette. Ceci est particulièrement important pour de nombreux programmes car il est impératif d'empêcher que l'utilisateur change de disquette par inadvertance.

Ceci vaut plus particulièrement pour le DOS qui effectue une lecture de la table d'allocation des fichiers (FAT) lors du premier accès à une disquette, afin de voir quelles sont les secteurs des disquettes occupées par des fichiers et ceux encore disponibles. Si la disquette a été subrepticement changée, le DOS continue de travailler avec l'ancienne table d'allocation des fichiers (FAT) et risque alors d'accéder à des secteurs déjà occupés de la nouvelle disquette. D'autre part, il peut naturellement se produire que la même disquette soit insérée une nouvelle fois sans rendre toutefois nécessaire une nouvelle lecture de la FAT. Comme vous pouvez le voir, la question est plus complexe qu'elle n'apparaît au premier abord. Ce problème ne concerne toutefois que plus particulièrement le niveau du DOS et revêt une importance secondaire pour ce qui nous concerne ici.

En fait, le BIOS met à disposition du DOS des moyens permettant à ce dernier de détecter un changement de disquette dans le lecteur. Il suffit donc de communiquer à la fonction 15h, de la même manière que pour la fonction 8h, le numéro de fonction du registre AH et l'identification du lecteur du registre DL.

Cette fonction renvoit un code correspondant à la table suivante. Ce code indique, à côté de l'état, si le lecteur indiqué existe et s'il s'agit d'un disque dur.

Fonction 15h, lecture du type de disque	
Code	**Signification**
AH = 00h	Pas de disque
AH = 01h	Disque souple ne pouvant détecter le changement de disque
AH = 02h	Disque souple pouvant détecter le changement de disque
AH = 03h	Disque dur

Lecture des secteurs de disquette

L'une des fonctions de base que le BIOS est chargé de mettre à disposition consiste à effectuer une lecture des différents secteurs des disquettes. La fonction 02h permet d'effectuer ce travail. Il est nécessaire de lui indiquer les secteurs qui doivent être lus ; en effet, plusieurs secteurs peuvent être lus simultanément à condition qu'ils fassent partie d'une même piste et qu'ils soient contigus.

Les registres suivants doivent être chargés avant le lancement de la fonction avec l'adresse du premier secteur qui doit être lu.

Registres lors du lancement de la fonction 02h	
Code	**Signification**
DL	Numéro de disque
DH	Numéro de tête (0 = haut
CL	Numéro de secteur (1 à N)
CH	Numéro de piste (0 à N-1)
AL	Nombre de secteurs à lire (doit être >0)

Sachant que les données ne peuvent être transférées automatiquement dans une zone de mémoire fixe, l'adresse d'un tampon doit être indiquée dans le couple de registres ES:DX. Comme d'habitude, ES retient l'adresse segment du tampon et DX l'adresse offset. Ces paramètres une fois chargés correctement, on a encore besoin pour le lancement de la fonction de l'identificateur de lecteur dans le registre DL et du numéro de fonction dans le registre AH.

Après l'accès en lecture, la fonction indique l'état d'erreur dans le registre AH et le nombre de secteurs lus dans le registre AL. Le positionnement d'un sémaphore de report signale en outre l'apparition d'une erreur.

Il est par ailleurs possible de "dévoyer" la lecture de certains secteurs afin de connaître le format d'une disquette dans le cas d'un lecteur MF sachant que celui-ci contiendra

soit des disquettes DD, soit des disquettes HD. Un accès en lecture sur un secteur ayant un numéro supérieur à 9 permet de faire toute la lumière ; en effet, dans le cas de disquettes DD, un tel secteur n'existe pas et la fonction indiquera alors une erreur.

Même si, dans ce cas, une erreur est souhaitable, il est recommandé de ne pas interrompre aussitôt une opération dans une telle situation. En effet, il est recommandé de répéter au moins trois fois chaque tentative de lecture, d'écriture et de formatage avant de se considérer vaincu et de supposer une erreur. Il arrive en effet très fréquemment qu'une opération échoue lors de la première tentative, mais réussisse lors d'une seconde voire, d'une troisième fois. Cela provient du fait que la tête de lecture/écriture n'était pas correctement positionnée lors de la première tentative et que le lecteur n'était pas correctement synchronisé avec les variations de signal électrique.

Il n'y a aucun danger qu'une erreur soit inscrite dans les données car le lecteur enregistre pour chaque secteur un "checksum" (vérification de la somme) permettant de contrôler la cohérence des informations lues.

Ecrire des secteurs de disquettes

La fonction O3h est mise oeuvre pour la description des différents secteurs ; les entrées de registres de cette fonction ressemble à celle de la fonction O2h.

Registres lors du lancement de la fonction 03h	
Code	**Signification**
AL	Nombre de secteurs à écrire
DL	Numéro de disque
DH	Numéro de tête (0 = haut, 1 = bas)
CL	Numéro de secteur (1 à N)
CH	Numéro de piste (0 à N-1)
AL	Nombre de secteurs à lire
ES:BX	Adresse de la zone mémoire qui contient les secteurs à écrire

La seule différence avec la fonction 02h est que la zone mémoire doit être remplie avant le lancement de la fonction, et ce par le programme à lancer et par les données à écrire.

Vérifier les secteurs de la disquette

Si vous souhaitez vérifier que les données ont été correctement transférées sur la disquette, vous pouvez utiliser la fonction 04h. Celle-ci ne procédera pas, contrairement à l'idée communément reçue et répandue, à une comparaison des données en mémoire avec celle de la disquette. Seule sera vérifiée la correction du transfert de données à l'aide d'une valeur CRC. CRC signifie "Cyclical Redundancy Check" et représente une procédure de test très fiable au cours de laquelle les valeurs de chaque octet à l'intérieur d'un secteur vont être associées à une somme par le biais d'une formule mathématique compliquée.

Vu que la plupart des lecteurs de disquettes sont très fiables et travaillent avec précision, cette routine est considérée par la plupart des programmeurs comme superflue. Le DOS lui-même ne fait appel à cette fonction lors de l'écriture de données que lorsque la commande DOS VERIFY ON a été préalablement saisie.

Les entrées de registres sont semblables à celles des fonctions 02h et 03h avec toutefois une différence : aucune adresse de zone mémoire ne sera communiquée.

Le formatage des pistes de disquettes

Il est bien sûr possible de formater des disquettes à l'aide du BIOS, mais un problème se pose, lié à l'introduction des lecteurs MF et de l'utilisation du format HD dans l'univers PC qui en découle. Il s'agit du paramétrage du format de disquettes. En effet, le BIOS doit connaître, pour le formatage d'une disquette, le nombre de secteurs par piste et le nombre de pistes qui doivent être tracées. Il s'agit donc du format de la disquette que l'on peut obtenir très simplement en faisant un accès en lecture sur le dixième, onzième ou douzième secteur d'une piste tel que nous l'avons vu précédemment. Cette astuce ne fonctionne évidemment pas dans le cas du formatage vu que les secteurs qui doivent être lus n'existent pas encore à ce moment.

C'est d'ailleurs pourquoi l'utilisateur doit taper le paramètre /4 lors du lancement de la commande FORMAT lorsqu'il souhaite formater une disquette DD à partir d'un lecteur 5"1/4 sous DOS. Le DOS a en effet besoin d'une information pour paramétrer le format souhaité. Le programme FORMAT a alors recours à une fonction BIOS qui doit être lancée systématiquement avant le premier formatage d'une piste : il s'agit de la fonction 18h.

Cette fonction a été introduite avec la PC/AT et remplace l'ancienne fonction 17h. Parallèlement au numéro de fonction dans le registre AH, la fonction 18h attend comme paramètres le nombre souhaité de pistes dans le registre CH, le nombre de secteurs par piste dans le registre CL avec, bien sûr, le numéro du lecteur dans le registre DL.

Si le format indiqué est supporté par le lecteur concerné, la fonction est renvoyée avec un sémaphore de report effacé et inscrit dans le couple de registres ES:DI un pointeur sur la Table des Paramètres du Lecteur de Disquettes nécessaire pour le formatage sous le format indiqué.

Nous décrirons dans le passage suivant le rôle de la table des paramètres disques (DDPT). Notons simplement à cet endroit que le pointeur doit être inscrit dans le vecteur d'interruption 1Eh dans lequel BIOS fixe respectivement un pointeur sur la table des paramètres disques courante.

Lorsque le format souhaité est paramétré et que la table des paramètres disques est installée, la procédure de formatage proprement dite peut alors commencer.

A cet effet, on fait appel à la fonction 05h qui permet le formatage d'une piste complète. Chaque secteur peut être formaté à raison de 128, 256, 512 ou même 1024 octets par secteur ; il est toutefois impératif de sélectionner le format 512 octets si la disquette doit ensuite être utilisée sous DOS. Le DOS tolère en effet exclusivement cette taille de secteur.

Lors du lancement de la fonction 05h, le registre AL doit contenir le nombre des secteurs de la piste à formater. Le registre CH attend quant à lui le numéro de la piste à formater qui se situera entre 0 et 39 ou entre 0 et 79. Le numéro du lecteur de disquettes sera inscrit dans le registre DL et la phase de la disquette dans le registre DH.

Registres lors du lancement de la fonction 05h	
Code	**Signification**
AL	Nombre de secteurs sur piste (doit être >0)
DL	Numéro de disque
CH	Numéro de piste
DH	Numéro de tête
ES:BX	Adresse de la zone mémoire qui contient les paramètres de formatage

Parallèlement à ces informations, également attendues pour les fonctions de lecture, d'enregistrement et de vérification d'un secteur, la fonction de formatage doit disposer également d'un champ contenant les caractéristiques de formatage. L'adresse de ce champ est attendue dans le couple de registres ES:BX en tant que pointeur FAR.

La table présentée ci-après peut contenir une entrée pour chaque secteur à formater ; chaque entrée se compose de 4 octets :

Valeur	Signification
0	Piste à formater
1	Face de disquette (toujours 0 pour les disquettes simple face): 0 = dessus 1 = dessous
3	Nombre d'octets dans ce secteur 0 = 128 octets 1 = 256 octets 2 = 512 octets 3 = 1024 octets

Bien que le numéro de la piste à formater ainsi que la phase de la disquette ai été déjà indiqués lors du lancement de la fonction 05h, ils doivent être ici répétés. Les secteurs vont être rangés physiquement dans cette table suivant l'ordre des entrées, ce qui permet d'attribuer au premier secteur le numéro 1 et au deuxième le numéro 7. En effet, le numéro de secteur logique va être inscrit sur la disquette au début de chaque secteur afin que le lecteur soit capable d'identifier un secteur recherché.

Sachant que le BIOS ne fixe pas lui-même le numéro de secteur logique, ceux-ci peuvent donc être décalés par rapport aux secteurs physiques provoquant ainsi ce que l'on appelle "l'entrelacement". En règle générale, seuls les disques durs y ont recours, tel que nous le montre la section 6.7.1. Lors du formatage de disquette, cette procédure n'est d'aucune aide ; nous vous recommandons donc d'utiliser une numérotation ascendante des différents secteurs lors de la création de la table de formats.

Le nombre d'octets affecté à un secteur n'a pas lieu d'être identique pour chacun d'eux vu que ce nombre est défini individuellement pour chaque secteur par le biais de la table. Ceci permet, en liaison avec les autres paramètres de cette table, d'élaborer une protection en copie, tel que nous le verrons un peu plus loin.

Un exemple de programme, que vous trouverez à la fin de ce chapitre, vous montre comment formater des disquettes à l'aide des fonction 18h et 05h.

Table des Paramètres du Lecteur de Disquettes

Mises à part les indications concernant le format physique de la disquette, le BIOS à besoin, pour la programmation du contrôleur de disquettes, de toute une série d'informations supplémentaires. Celles-ci se trouvent dans la table des paramètres disques (abréviation : DDPT), que nous avons déjà mentionnée plusieurs fois au cours de ce chapitre.

Une table de ce type existe dans le BIOS en ROM pour chaque lecteur et chaque format de disquettes supporté. Il est toutefois possible d'installer une DDPT particulière car le

BIOS repère la table des paramètres disques courante au moyen d'un pointeur FAR fixé dans une zone mémoire affectée autrement à l'interruption 1Eh. L'utilisation de cet espace mémoire devient possible vu que l'interruption 1Eh n'est utilisée ni par l'électronique du PC, ni par le DOS et qu'elle ne sert pas d'interface pour certaines fonctions BIOS.

La possibilité qui consiste à installer une DDPT particulière est utilisée par le DOS qui modifie certaines valeurs de cette table en fonction du BIOS. L'idée consiste à pouvoir accélérer l'accès aux disquettes, ce qui est effectivement possible vu que certaines valeurs de cette table ont été définies avec largesse. La question qui se pose maintenant est de savoir quel est le contenu exact de cette table.

La table en elle-même se compose de 11 octets comme nous le montre le tableau suivant. Il n'est pas possible de modifier tous les paramètres ; on dispose toutefois d'une certaine marge de manoeuvre pour ceux qui apparaissent en italique.

Structure de la Table des Paramètres du Lecteur de Disquettes

Valeur	Signification	Type
00h	*Vitesse de progression et Head-Unload-Time*	*1 OCTET*
01h	*Head-Load-Time*	*1 OCTET*
02h	*Laps de temps du fonctionnement du moteur du lecteur après opération disquette*	*1 OCTET*
03h	Taille du secteur	1 OCTET
04h	Secteurs par piste	1 OCTET
05h	Taille du GAP3 en lecture/écriture	1 OCTET
06h	Longueur transfert de données	1 OCTET
07h	Taille du GAP3 lors du formatage	1 OCTET
08h	*Code ASCII pour formatage*	*1 OCTET*
09h	*Temps de repos*	*1 OCTET*
0Ah	*Vitesse du moteur du lecteur*	*1 OCTET*

Le premier octet est en deux parties et contient dans chacun des deux mots respectivement la vitesse de progression (BIT 4 à 7) ainsi que le temps de remontée de la tête (BIT 0 à 3). La vitesse de progression fixe le laps de temps maximal pouvant s'écouler lors du déplacement de la tête de lecture/écriture d'une piste à l'autre. Elle est exprimée en millisecondes, la valeur 0Fh représentant 1 ms, 0Eh 2 ms, 0Dh 3 ms, etc.. Le temps de remontée de la tête fixe le laps de temps pouvant s'écouler lors de la remontée de la tête de lecture/écriture, par exemple lors du changement de piste. Elle est exprimée en multiple de 16 ms. La valeur par défaut 0Fh (240 ms) laisse une bonne marge de manoeuvre. Il est donc possible en règle générale de réduire cette valeur.

Dans le second octet, on trouve le temps de descente de la tête de lecture/écriture sur une piste. Cet octet occupe les BIT 1 à 7 et est exprimé en multiple de 2 ms. Sachant que, lors d'un accès aux disquettes, il faut attendre que le moteur du lecteur de disquettes ait atteint sa vitesse de croisière, on indique en règle générale un temps de descente peu élevé (1 ou 2).

Dans cet octet, le BIT 0 est occupé par le flag DMA qui décide de l'utilisation du canal DMA par le lecteur de disquettes. Ce BIT doit avoir la valeur 0.

Le troisième octet définit le laps de temps pendant lequel le moteur du lecteur de disquettes doit continuer à tourner après une opération disquette. Il s'agit en fait du laps de temps qui s'écoule jusqu'à ce que le moteur du lecteur soit arrêté. Cette durée est en principe de deux secondes car on considère qu'une opération disquette est généralement suivie d'un autre accès à la disquette. Or, il est bien sûr préférable, lors d'accès successifs, que le moteur de la disquette est déjà atteint sa vitesse d'exploitation et qu'il ne soit donc pas nécessaire de le relancer. La valeur figurant dans cette cellule de mémoire est basée sur une fréquence système de 18 unités par seconde (1 unité correspond à 55 ms) une valeur de 18 représentera donc une durée de rotation d'une seconde environ. La valeur par défaut est de 25h, ce qui correspond à deux secondes.

La valeur de l'octet suivant fixe le nombre d'octets par secteur avec lequel on doit travailler pour les opérations de lecture ou d'écriture. Cette valeur doit bien sûr coïncider avec celle utilisée pour le formatage d'un secteur. Elle sera en règle générale de 3 pour 512 octets par secteur. Si l'on veut lire ou écrire des secteurs affectés d'autres tailles de secteurs, il convient auparavant d'inscrire la valeur appropriée dans cette cellule de mémoire.

L'octet 4 indique le nombre maximal de secteurs par piste, lequel dépend du format de disquette choisi.

Les trois octets suivants se rapportent au codage et au décodage des informations secteurs qui sont stockées sur la disquette en plus des données proprement dites. Il est fortement déconseillé de modifier ces valeurs !

L'octet 8 est en revanche fort intéressant. Il reçoit le code ASCII du caractère qui doit être utilisé pour remplir un secteur lors de son formatage. En effet, au cours du formatage, les secteurs ne sont pas seulement créés mais également remplis par un contenu constant. Par défaut, il s'agira du caractère de division (Code ASCII 246).

Après être passé d'une piste à une autre, la tête de lecture/écriture a besoin d'un bref temps de repos pour que les vibrations liées à ce déplacement puissent s'estomper. Ce n'est qu'après cette pause que la tête sera en mesure d'effectuer correctement les prochains accès à la disquette. Le temps qu'on lui accorde est fixé par l'octet 9 de la table. Il est exprimé en unités d'une millisecondes. La valeur par défaut est de 25 millisecondes.

La dernière entrées dans la table, l'octet 10, fixe le laps de temps nécessaire pour que le moteur du lecteur de disquettes atteigne sa vitesse de croisière. La valeur de cette cellule de mémoire est exprimée en multiple de 1/8 de seconde. Ici encore, le DOS remet en cause les avantages consentis au moteur disquettes par le BIOS. Il ne prévoit en effet qu'un délai de 1/4 de seconde au lieu de 1/2 secondes prévu par le DOS.

Exemple de programmes

Il ne faut pas attendre de miracles de la modification des différentes valeurs de la Table des Paramètres du Lecteur de Disquettes. Mais je suis certain que certains lecteurs de cet ouvrage doivent avoir toutefois envie de faire quelques essais. C'est pourquoi j'ai développé quelques petits programmes en PASCAL et en C qui permettent de modifier les différents paramètres de la Table des Paramètres du Lecteur de Disquettes. Tous ne seront toutefois pas modifiés car, comme nous l'avons déjà évoqué, la modification de certains d'entre-eux pourrait avoir de fâcheuses conséquences.

Ces deux programmes s'appellent DDPTP.PAS et DDPTC.C et fonctionnent de la même manière. On les lance à partir de la ligne de commande du DOS, en n'indiquant tout d'abord aucun paramètre. Dans ce cas, les programmes affichent uniquement les contenus de la Table des Paramètres du Lecteur de Disquettes. Il s'agit de la dernière table appelée. Vous pouvez, avant le programme Table des Paramètres du Lecteur de Disquettes, lancer par exemple la commande DIR si vous souhaitez appliquer ce programme à un lecteur de disquettes particulier.

```
DDPTP (c) 1991, 1992 by Michael Tischer
Optimiser les accès à la disquette


Contenu DDPT:
Steprate                (SR): $0D

Head-Unload-Time        (HU): $0F
Head-Load-Time          (HL): $01
Head-Settle-Time        (HS): $0F

Temps de rotation du moteur après (MN): $25
Temps de rotation du moteur avant (MA): $08
```

Les différents paramètres DDPT peuvent être modifiés ; il suffit d'indiquer lors du lancement du programme l'abréviation souhaitée qui apparaît lors de l'affichage des paramètres à l'écran. Les deux caractères doivent être immédiatement précédés de deux points auxquels viendront s'ajouter les deux chiffres hexadécimaux qui fixent la nouvelle valeur du paramètre concerné.

La commande :

```
DDPTP MA:04 SR:08
```

vous permet par exemple de réduire le temps de démarrage du lecteur de disquettes à 1/2 seconde ainsi que la vitesse de progression à 8 millisecondes.

La modification des différents paramètres n'est toutefois possible que si la Table des Paramètres du Lecteur de Disquettes ne se trouve pas dans le BIOS en ROM et ne pourra donc être écrasée. Le programme vous en informe dans tous les cas :

Listing : DDPTP.PAS

```
{*********************************************************************}
{*                          D D P T P                                *}
{*-------------------------------------------------------------------*}
{*   Fonction       : Permet d'optimiser les accès à la disquette    *}
{*                    en modifiant les valeurs de la Disk-Drive-     *}
{*                    Parameter-Table.                               *}
{*-------------------------------------------------------------------*}
{*   Auteur              : Michael Tischer                           *}
{*   Développé le        : 22.08.1991                                *}
{*   Dernière modification : 27.01.1992                              *}
{*********************************************************************}

program DDPT;

Uses Crt, Dos;                          { Intégrer les unités CRT et DOS }

type DDPT_T   = array[ 0..10 ] of byte;      { Structure pour la DDPT }
     DDPT_PTR = ^DDPT_T;                     { Pointeur sur la DDPT }

var  DDPT : DDPT_PTR;                        { Pointeur sur la DDPT }

{*********************************************************************}
{* byte_hex : Convertit un octet en un nombre hexadécimal            *}
{* Entrée   : Nombre à convertir                                     *}
{* Sortie   : Nombre sous forme de chaîne hexadécimal                *}
{*********************************************************************}

function byte_hex( Nombre : byte ) : string;

{-- Convertit un chiffre 0 - 15 en hexa 0 - F ------------------------}

function h_Chiffre( Chiffre : byte ) : char;

begin
  if ( Chiffre >= 10 ) then                  { Chiffre >= 10 soit A - F }
    h_Chiffre := chr( 55 + Chiffre )
  else                                  { Non chiffre comme chiffre décimal }
    h_Chiffre := chr( 48 + Chiffre );
end;

begin
  byte_hex := '$' + h_Chiffre( Nombre div 16 )
                  + h_Chiffre( Nombre mod 16 );
end;

{*********************************************************************}
{* hex_byte : Convertit une chaîne hexa en un octet                  *}
{* Entrée   : Chaîne hexa à convertir                                *}
{* Sortie   : Nombre                                                 *}
{*********************************************************************}

function hex_byte( hex : string ) : byte;

{-- Convertir un chiffre hexa 0 - F en  0 - 15 -----------------------}

function d_Chiffre( Chiffre : char ) : byte;

begin
  if ( Chiffre >= 'A' ) and ( Chiffre <= 'F' ) then
    d_Chiffre := ord( Chiffre ) - 55
  else                                  { Non chiffre comme chiffre décimal }
    d_Chiffre := ord( Chiffre ) - 48;
end;

begin
  if ( hex[ 1 ] = '$' ) then delete( hex, 1, 1 );
  if length( hex ) = 1 then hex := '0' + hex;
  hex_byte := d_Chiffre( hex[ 1 ] ) * 16 + d_Chiffre( hex[ 2 ] );
end;

{*********************************************************************}
{* RAM_DDPT : Tester si DDPT se trouve dans la RAM ou la ROM         *}
{* Entrée   : Aucune                                                 *}
{* Sortie   : true si DDPT dans la RAM                               *}
{* Info     : La fonction inscrit une valeur dans la DDPT, la relit  *}
{*            puis compare si la valeur a été inscrite, la DDPT      *}
{*            se trouve donc dans la RAM                             *}
{*********************************************************************}

function RAM_DDPT : boolean;

var buffer : byte;          { Mémoire pour la valeur actuelle de la DDPT }

begin
  buffer := DDPT^[ 0 ];                  { Sauvegarder la valeur DDPT }
  DDPT^[ 0 ] := not buffer;                    { Inverser la valeur }
  RAM_DDPT := ( DDPT^[ 0 ] = not buffer );{ Evaluer le test d'écriture }
  DDPT^[ 0 ] := buffer                   { Restaurer l'ancienne valeur }
end;

{*********************************************************************}
{* AfficherValeur : Afficher la valeur DDPT                          *}
{* Entrée         : Aucune                                           *}
{* Sortie         : Aucune                                           *}
{* Infos          : La procédure affiche la valeur actuelle de la    *}
{*                  DDPT sur l'écran.                                *}
{*********************************************************************}

procedure AfficherValeur;

begin
  writeln( 'Steprate                (SR): ',
           byte_hex( DDPT^[ 0 ] shr 4 ) );
  writeln( #13#10'Head-Unload-Time        (HU): ',
           byte_hex( DDPT^[ 0 ] and $F ) );
  writeln( 'Head-Load-Time          (HL): ',
           byte_hex( DDPT^[ 1 ] shr 1 ) );
  writeln( 'Head-Settle-Time        (HS): ',
           byte_hex( DDPT^[ 9 ] ) );
  writeln( #13#10'Temps de rotation du moteur après (MN): ',
           byte_hex( DDPT^[ 2 ] ) );
  writeln( 'Temps de rotation du moteur avant (MA): ',
           byte_hex( DDPT^[ 10 ] ) );
end;

{*********************************************************************}
{* ValeursNouv : Définir les nouvelles valeurs de la DDPT            *}
{* Entrée      : Aucune                                              *}
{* Sortie      : Aucune                                              *}
{*********************************************************************}

procedure ValeursNouv;

var i,j     : byte;                          { Compteur de boucle }
    Art     : string[ 2 ];                   { Paramètre à modifier}
    Valeur  : byte;                  { Nouvelle valeur à définir }
```

```
  ValSecours : byte;                    { Valeur de secours à sauvegarder }
  Commande   : string[ 6 ];   { Paramètre de la ligne d'instruction }

begin
  {-- Boucle: Examiner tous les paramètres ----------------------------}

  for i := 1 to Paramcount do
    begin                                                { Lire le paramètre }
      Commande := paramstr( i );                { Commande en majuscule }
      for j := 1 to length( Commande ) do
        Commande[ j ] := upcase( Commande[ j ] );
      Art := copy( Commande, 1, 2 );                  { Valeur à modifier }
      delete( Commande, 1, 3 );                { Déterminer nouvelle valeur }
      Valeur := hex_byte( Commande );                          { Step rate? }
      if ( Art = 'SR' ) then
        begin
          Valeur := Valeur shl 4;   { Valeur dans le quartet supérieur }
          ValSecours := DDPT^[ 0 ] and $0F;          { Quartet inférieur }
          DDPT^[ 0 ] := Valeur or ValSecours;  { Sauvegarder la valeur }
        end                                                { Head-Unload-Time? }
      else if ( Art = 'HU' ) then
        begin
          Valeur := Valeur and $0F; { Valeur dans le quartet inférieur }
          ValSecours := DDPT^[ 0 ] and $F0;          { Quartet supérieur }
          DDPT^[ 0 ] := Valeur or ValSecours;  { Sauvegarder la valeur }
        end                                                { Head-Load-Time? }
      else if ( Art = 'HL' ) then
        DDPT^[ 1 ] := Valeur shl 1   { Sauver valeur dans les bits 1-7 }
      else if ( Art = 'HS' ) then                  { Head-Settle-Time? }
        DDPT^[ 9 ] := Valeur                       { Sauvegarder la valeur }
      else if ( Art = 'MN' ) then { Temps de rotation du moteur après? }
        DDPT^[ 2 ] := Valeur                       { Sauvegarder la valeur }
      else if ( Art = 'MA' ) then { Temps de rotation du moteur avant? }
        DDPT^[ 10 ] := Valeur;                     { Sauvegarder la valeur }
    end;
end;
{*****************************************************************}
{*                                                               *}
{*                        PROGRAMME PRINCIPAL                    *}
{*                                                               *}
{*****************************************************************}

begin                                                { Effacer l'écran }
  ClrScr;
  writeln( 'DDPTP (c) 1991, 1992 by Michael Tischer');
  writeln( 'Optimiser les accès à la disquette' );

  GetIntVec( $1E, pointer( DDPT ) );     { Lire le pointeur sur la DDPT }

  if ( RAM_DDPT ) then                   { DDPT dans la RAM donc modifiable? }
    begin
      if ( Paramcount > 0 ) then          { Faut-il définir des valeurs? }
        begin
          ValeursNouv;                    { Oui, définir de nouvelles valeurs }
          writeln( #13#10#10'Nouvelles valeurs DDPT:');
          AfficherValeur;                 { Afficher nouvelle valeur DDPT }
        end;
    end                { DDPT se trouve dans la ROM, impossible de modifier }
  else
    writeln( 'Il est impossible de modifier la ' +
             'Disk-Drive-Paramter-Table car elle se trouve dans la ROM ' );

  writeln( #13#10#10'Contenu DDPT:');
  AfficherValeur;                        { Afficher anciennes valeurs DDPT }
end.
```

Listing : DDPTC.C

```
/*****************************************************************/
/*                                                               */
/*                            D D P T C                          */
/*                                                               */
/*-----------------------------------------------------------------*/
/*  Fonction       : Permet d'optimiser les accès à la disquette   */
/*                   en modifiant les valeurs de la Disk-Drive-    */
/*                   Parameter-Table.                            */
/*                                                               */
/*-----------------------------------------------------------------*/
/*  Auteur                 : Michael Tischer                     */
/*  Développé le           : 22.09.1991                          */
/*  Dernière modification  : 22.09.1991                          */
/*****************************************************************/

/*-- Intégrer les fichiers Include ---------------------------------*/

#include <dos.h>
#include <stdio.h>
#include <io.h>
#include <string.h>

/*-- Macros --------------------------------------------------------*/

#ifdef MK_FP                          /* Macro MK_FP déjà définie? */
  #undef MK_FP                        /* Si oui, effacer la macro */
#endif

#define MK_FP(seg,ofs) ((void far *) ((unsigned long) (seg)<<16|( ofs)))

/*-- Typedefs ------------------------------------------------------*/

typedef unsigned char byte;           /* Octet de type de données */
typedef byte DDPT_Typ[ 11 ];            /* Champ pour une DDPT */

/*****************************************************************/
/* upcase       : Convertit une lettre en majuscule              */
/* Entrée       : Lettre                                         */
/* Sortie       : Majuscule                                      */
/*****************************************************************/

byte upcase( letter )
byte letter;

{
  if ( ( letter > 0x60 ) && ( letter < 0x7B ) )   /* Convertir? */
    return( letter & 0xDF );                      /* Oui, masquer le bit */
  else                                            /* Non, retourner tel quel */
    return( letter);
}

/*****************************************************************/
/* D_Chiffre : Convertit un chiffre hexadécimal en une valeur    */
/*             décimale                                          */
/* Entrée    : Chiffre hexadécimal                               */
/* Sortie    : Nombre                                            */
/*****************************************************************/

byte D_Chiffre( Hex )
char Hex;

{
  if ( ( Hex >= 0x41 ) && ( Hex <= 0x46 ) )          /* A - F? */
    return( Hex - 55 );                              /* Oui */
  else                                               /* Non, donc 0..9 */
    return( Hex ) - 48;
}

/*****************************************************************/
/* hex_byte : Convertit une chaîne hexa en un octet              */
/* Entrée   : Voir plus bas                                      */
/* Sortie   : Nombre                                             */
/*****************************************************************/

byte hex_byte( hex )                  /* Chaîne hexa à convertir */
char *hex;

{
  if ( hex[ 1 ] == 0x58 )             /* Tester si X donc Nombre 0x.. */
    hex += 2;                         /* Pointeur sur 1er chiffre */
  return( ( D_Chiffre( *hex ) << 4 ) | D_Chiffre( hex[ 1 ] ) );
}

/*****************************************************************/
/* GetIntVec    : Lit un vecteur d'interruption                  */
/* Entrée       : Numéro d'interruption                          */
/* Sortie       : Vecteur d'interruption                         */
/*****************************************************************/

void far *GetIntVec( Numero )
int Numero;

{
  return( * ( void far * far * ) ( MK_FP( 0, Numero * 4 ) ) );
}

/*****************************************************************/
/* RAM_DDPT : Tester si DDPT se trouve dans la RAM ou la ROM     */
/* Entrée   : Voir plus bas                                      */
/* Sortie   : true si DDPT dans la RAM                           */
/* Infos    : La fonction inscrit une valeur dans la DDPT, la relit */
```

```
/*
/*          puis compare si la valeur a été inscrite, la DDPT se    */
/*          trouve donc dans la RAM                                 */
/************************************************************************/

int RAM_DDPT( DDPT )
DDPT_Typ far *DDPT;
                                        /* Pointeur sur DDPT */
{
 byte buffer;           /* Mémoire pour la valeur actuelle de la DDPT */
 int  Flag;                      /* Mémoire pour la valeur de retour */

 buffer = *DDPT[ 0 ];                   /* Sauvegarder la valeur DDPT */
 *DDPT[ 0 ] = buffer ^ 0xFF;                    /* Inverser la valeur */
 Flag = ( *DDPT[ 0 ] == ( buffer ^0xFF ) );
 *DDPT[ 0 ] = buffer;                   /* Restaurer l'ancienne valeur */
 return( Flag );                                /* Valeur de retour */
}

/************************************************************************/
/* AfficherValeur : Afficher la valeur DDPT                             */
/* Entrée         : Voir plus bas.                                      */
/* Sortie         : Aucune                                              */
/* Infos          : La procédure affiche la valeur actuelle de la       */
/*                  DDPT sur l'écran.                                   */
/************************************************************************/

void AfficherValeur( DDPT )
DDPT_Typ far *DDPT;
                                        /* Pointeur sur DDPT */
{
  printf( "Steprate                 (SR): 0x%02x\n\n",
          ( *DDPT )[ 0 ] >> 4 );
  printf( "Head-Unload-Time         (HU): 0x%02x\n",
          ( *DDPT )[ 0 ] & 0x0f );
  printf( "Head-Load-Time           (HL): 0x%02x\n",
          ( *DDPT )[ 1 ] >> 1 );
  printf( "Head-Settle-Time         (HS): 0x%02x\n\n", ( *DDPT )[ 9 ] );
  printf( "Temps de rotation du moteur après (MN): 0x%02x\n",
                          ( *DDPT )[ 2 ] );
  printf( "Temps de rotation du moteur avant   (MA): 0x%02x\n",
                          ( *DDPT )[ 10 ] );
}

/************************************************************************/
/* ValeursNouv : Définir les nouvelles valeurs DDPT                     */
/* Entrée      : Voir plus bas                                          */
/* Sortie      : Aucune                                                 */
/************************************************************************/

void ValeursNouv( Nombre, Valeurs, DDPT )
int Nombre;
char *Valeurs[];                        /* Nombre de commandes */
                                        /* Champ avec commandes */
DDPT_Typ far *DDPT;                     /* Pointeur sur DDPT */
{
 int i,j;
 char Art[ 4 ],                         /* Compteur de boucle */
      Commande[ 8 ];                    /* Paramètre à modifier */
 byte Valeur,                   /* Paramètres de la ligne d'instruction */
      ValSecours;               /* Nouvelle valeur à définir */
                                /* Valeur de secours à sauvegarder */
 /*-- Boucle: Examiner tous les paramètres ----------------------------*/

 for ( i = 1; i < Nombre; i++ )
 {
  strcpy( Commande, Valeurs[ i ] );             /* Lire le paramètre */
  j = 0;
  while ( Commande[ j ] != 0 )
   Commande[ j++ ] = upcase( Commande[ j ] );

  Art[ 0 ] = Commande[ 0 ];
  Art[ 1 ] = Commande[ 1 ];                     /* Paramètre à définir */
  Art[ 2 ] = 0;
  Valeur = hex_byte( &Commande[ 3 ] );          /* Valeur à définir */
  if ( !strcmp( Art, "SR" ) )                           /* Step rate? */
  {
   Valeur = Valeur << 4;        /* Valeur dans le quartet supérieur */
   ValSecours = ( *DDPT )[ 0 ] & 0x0F;  /* Lire le quartet inférieur */
   ( *DDPT )[ 0 ] = Valeur | ValSecours;        /* Ecrire la valeur */
  }
  else if ( !strcmp( Art, "HU" ) )              /* Head-Unload-Time? */
  {
   Valeur = Valeur & 0x0F;      /* Valeur dans le quartet inférieur */
   ValSecours = ( *DDPT )[ 0 ] & 0xF0;  /* Lire le quartet supérieur */
   ( *DDPT )[ 0 ] = Valeur | ValSecours;        /* Ecrire la valeur */
  }
  else if ( !strcmp( Art, "HL" ) )              /* Head-Load-Time? */
   ( *DDPT )[ 1 ] = Valeur << 1; /* Sauve la valeur dans les bits 1-7 */
  else if ( !strcmp( Art, "HS" ) )              /* Head-Settle-Time? */
   ( *DDPT )[ 9 ] = Valeur;                     /* Sauvegarder la valeur */
  else if ( !strcmp( Art, "MN" ))/* Temps de rotation du moteur avant */
   ( *DDPT )[ 2 ] = Valeur;                     /* Sauvegarder la valeur */
  else if ( !strcmp( Art, "MA" ))/* Temps de rotation du moteur après */
   ( *DDPT )[ 10 ] = Valeur;                    /* Sauvegarder la valeur */
 }
}

/************************************************************************/
/*                          PROGRAMME PRINCIPAL                         */
/************************************************************************/

void main( argc, argv )
int argc;
char *argv[];

{
 DDPT_Typ far *DDPT;                    /* Pointeur sur la DDPT actuelle */

 printf( "DPPTC (c) 1991, 1992 by Michael Tischer\n" );
 printf( "Optimiser les accès à la disquette\n" );

 DDPT = GetIntVec( 0x1E );              /* Lire le pointeur sur la DDPT */

 if ( RAM_DDPT )                        /* DDPT dans la RAM, donc modifiable? */
 {
  if ( argc > 1 )                       /* Faut-il définir les valeurs? */
  {
   ValeursNouv( argc, argv, DDPT );/* Définir les nouvelles valeurs */
   printf( "\n\nNouvelles valeurs de la DDPT:\n" );
   AfficherValeur( DDPT );/* Afficher les nouvelles valeurs de DDPT */
  }
 }
 else                   /* DDPT se trouve dans la ROM, impossible de modifier */
  printf( "%s %s ", "Il est impossible de modifier la ",
          "Disk-Drive-Parameter-Table car elle se trouve dans la ROM" );
 printf( "\nContenu DDPT:\n" );
 AfficherValeur( DDPT );/* Afficher les anciennes valeurs de la DDPT */
}
```

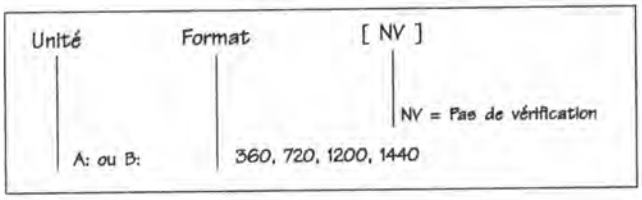
Formatage direct

En tant que programmeur, vous serez rarement amené à écrire des données sur une disquette ou à lire des données à partir d'une disquette directement à partir du BIOS ; en règle générale, vos programmes utiliseront des fichiers et il est alors préférable d'utiliser des fonctions du DOS.

Il existe toutefois une tâche qui requiert les différentes fonctions du BIOS de manière incontournable : le formatage de disquettes.

Nous allons donc vous proposer deux programmes écrits en PASCAL et en C capables d'effectuer ces deux tâches et qui représentent donc un succédanée d'années à la commande FORMAT du DOS. A l'instar de cette commande de DOS, ces deux programmes ne se contentent pas d'effectuer un formatage physique de la disquette mais inscrivent également sur celle-ci les différentes structures de données requises par le DOS. Cela concerne notamment le secteur d'amorçage, le répertoire principal de la disquette, vide dans un premier temps, ainsi que la FAT. Vous trouverez au chapitre 2 de plus amples informations sur les structures des données et sur la structure générale des mémoires de masse sous DOS.

Ces deux programmes s'appellent DFP.PAS et DFC.C et sont capables de traiter tous les formats connus par le DOS (360/1200 sur disquettes 5"25 ET 720/1440 sur disquette 3"5). On les lance suivant le schéma suivant :



Vu que ces deux programmes sont construits selon un schéma unique, et qu'ils utilisent des types et des constantes homonymiques, la description qui suit vaudra pour tous les deux.

Le programme principal interprète tout d'abord le premier argument de la ligne de commande du DOS en supposant qu'il s'agit de l'indication du lecteur. Cet argument est alors transformé en numéro de lecteur (0 ou 1) et le format du lecteur connecté est ensuite identifié à l'aide de la procédure GetDriveType ; celle-ci est basée sur la fonction 08h de l'interruption disque du BIOS qui renvoit un code type compris entre 0 et 4. Il est représenté à l'intérieur du programme par les constantes NO_DIRVE, DD_525, HD_525, DD_35 et HD_35.

La fonction 08h n'existe pas sur les PC/XT, ce qu'indique le sémaphore de report qui apparaît après le lancement de la fonction. Dans ce cas, on supposera la présence d'un lecteur 5"1/4 DD supportant uniquement le format 360 Ko.

Si la procédure GetDriveType a permis de constater que le lecteur indiqué existe, l'étape suivante consistera à transmettre les paramètres logiques et physiques nécessaires au formatage. Pour se faire, la fonction GetFormatParameter est lancée. Le format indiqué à partir de la ligne de commande lui est alors transmis sous forme de chaîne avec, en plus, le code type précédemment communiqué ainsi que deux variable du type PhysDataType et LogDataType. Vous pouvez voir ci-après les deux structures de données dans la version PASCAL.

```
type DdptType = array[ 0..10 ] of byte;                    { Structure DDPT }
     DdptPtr = ^DdptType;                                  { Pointeur sur DDPT }
     PhysDataType = record                                 { Paramètre Format Physique }
       Faces,                                  { nombre souhaité de faces de disquette }
       Pistes,                                             { Nombre de pistes par face }
       Secteurs : octet                                  { Nombre de secteurs par piste }
       DDPT     : DdptPtr;                  { Pointeur sur Table des Paramètres du
                                                          Lecteur de Disquettes }
     end;
     LogDataType = record                                  { Paramètre Format DOS }
       Media,                                                      { Media-Octet }
       Cluster,                                  { Nombre de secteurs par Cluster }
       FAT,                                      { Nombre de secteurs pour la FAT }
       RootSize : octet;                  { Entrées dans le répertoire principal }
    end;
    SpurBufType = array[ 1..18, 1..512 ] of byte;       { Zone mémoire par
                                                                        piste }
```

PhysDataType représente les paramètres physiques nécessaires au formatage. Il s'agit du nombre de faces, de pistes par face, ainsi que du nombre de secteurs par piste. De plus, un pointeur sur la DDPT est fixé car les deux programmes travaillent avec leur propre table afin d'accélérer le formatage.

Alors que les informations de PhysDataType sont requises pour le formatage physique, les informations contenues dans LogDataType sont destinées au "formatage logique", c'est à dire au transfert des différentes structures de données DOS. C'est pourquoi sont fixées ici le média DOS ID, le nombre de secteurs par clusters, la taille de la FAT en secteurs ainsi que le nombre des entrées du répertoire principal.

Les deux variables transmises du type PhysDataType et LogDataType sont initialisées à l'intérieur de GetFormatParameter à l'aide de toute une série de constantes typées (ou variables STATIC en C) qui contiennent toutes les informations nécessaires relatives au format supporté. Elles portent des noms du type DDPT_360, LOG_1200 ou PHYS_720 et sont donc facilement reconnaissables.

Avant que la procédure ne copie les différents paramètres dans les variables transmises, elle commence par vérifier si le format souhaité peut être accepté en liaison avec le lecteur concerné. Ce doit être le cas afin que l'exécution du programme puisse se poursuivre dans le programme principal avec le lancement de DiskPrepare, une procédure qui repose sur la fonction disque 18h du BIOS. Lors du paramétrage du format 360 Ko sur un lecteur 5"1/4, le lancement de cette fonction se soldera par un échec car la fonction 18h n'y est pas implémentée. Ceci est toutefois sans importance car ce format est fixé en raison du manque d'alternative.

Vous vous en souvenez peut-être la fonction 18h renvoit un pointeur sur la Table des Paramètres du Lecteur de Disquettes, qui appartient au format choisi. Cette information n'est toutefois pas utilisée à l'intérieur des deux programmes car une table personnelle est livrée par FormatGetParameter.

Cette table est activée après le lancement réussi de DiskPrepare ; son adresse est alors chargée dans le vecteur d'interruption qui représente l'interruption AEh. Le contenu de ce vecteur est cependant préalablement sauvegardé afin que l'ancienne table puisse être réactivée le formatage une fois réalisé.

Le formatage proprement dit s'exécute ensuite à l'aide de la fonction PhysicalFormat. Le troisième paramètre de la ligne de commande du DOS détermine si une vérification (vérify) des pistes doit avoir lieu au cours du formatage.

PhysicalFormat a recours, au cours du formatage, à la procédure FormatTrack qui correspond à la fonction disque 05h du BIOS afin de formater respectivement une piste. Elle est exécutée pour les pistes de 0 à N, chaque piste est tout d'abord formatée sur la face 0 puis ensuite sur la face 1. Il serait bien sûr possible de formater tout d'abord complètement la première face pour ensuite passer à la deuxième. Mais la procédure durerait alors plus longtemps car la tête de lecture/écriture devrait alors balayer deux fois la surface totale de la disquette. Certes, cette méthode requiert de basculer incessamment de la tête 0 à la tête 1, mais elle reste toutefois plus rapide que de déplacer la totalité du bras de lecture/écriture de piste en piste.

Après le formatage d'une piste à l'aide de FormatTrack, PhysicalFormat appelle la procédure VerifyTrack afin de contrôler la concordance entre le contenu d'un secteur et la "checksum" CRC enregistrée.

VerifyTrack, tout comme FormatTrack ou WriteTrack que nous n'avons pas encore décrits, n'est rien d'autre qu'une simple transposition de la fonction BIOS correspondante. Le lancement de la fonction est systématiquement répété plusieurs fois pour le cas où le BIOS annoncerait un échec de l'opération. Le nombre maximum de tentatives est fixé par la constante EssaisMax définie au début du listing. Si, à titre de test, vous placez cette constante sur A, les deux programmes de formatage risquent d'annoncer fréquemment un échec du formatage ; de fait, des erreurs de ce type apparaissent beaucoup plus souvent qu'on ne le penserait.

Mais retournons au programme principal. Lorsque la disquette a pu être formatée avec PhysicalFormat la dernière phase de traitement commence, à savoir le formatage logique du disque à l'aide de LogicalFormat. Comme nous l'avons déjà évoqué, les différentes structures de données vont alors être copiées sur la disquette, structures qui permettent au DOS de férer les fichiers.

Le secteur d'amorçage est toutefois intéressant. Il est inscrit sur la disquette pour le cas ou celle-ci se trouverait dans le lecteur au moment de l'amorçage. Son contenu est défini au début du programme dans la variable Masque Boot. La signification des données ainsi que du petit programme langage machine ne seront pas abordés ici. Ce que vous devez savoir cependant c'est la signification de la variable BootMes qui s'associe immédiatement au secteur d'amorçage. Elle contient la chaîne qui apparaît à l'écran lors de l'amorçage du système et est inscrite, de même que le secteur d'amorçage, sur la disquette.

Vous pouvez modifier cette chaîne à votre gré si vous souhaitez, par exemple, que votre propre nom ou celui de votre entreprise apparaisse à l'écran lors de l'amorçage de l'ordinateur à partir de la disquette. Notez cependant que le secteur se termine systématiquement par un octet ayant la valeur 00h qui marque la fin.

Lorsque LogicalFormat est terminée, l'exécution du programme est en soit terminé car, en dehors de l'affichage d'un message d'état, il ne se passe plus grand chose dans le programme principal. Voici donc les listings de DFP.PAS et DFC.C.

Listing : DFP.PAS

```
{*********************************************************************}
{*                          D F P . P A S                            *}
{*-------------------------------------------------------------------*}
{*  SUJET      : Formate disquettes 3,5" et 5,25"                    *}
{*-------------------------------------------------------------------*}
{*  Auteur              : Michael Tischer                            *}
{*  Développé le        : 23.08.1991                                 *}
{*  Dernière modification : 26.01.1992                               *}
{*********************************************************************}

program DFP;

Uses Dos;                                   { Intègre les unités Crt et DOS }

{-- Constantes ------------------------------------------------------}

const NO_DRIVE    = 0;                        { Lecteur introuvable }
      DD_525      = 1;                        { Lecteur: 5,25" DD   }
      HD_525      = 2;                        { Lecteur: 5,25" HD   }
      DD_35       = 3;                        { Lecteur: 3,5" DD    }
      HD_35       = 4;                        { Lecteur: 3,5" HD    }
      EssaisMax   = 5;                      { Nombre essais maximal }

{-- Déclarations de types -------------------------------------------}

type DdptType = array[ 0..10 ] of byte;       { Structure pour DDPT }
     DdptPtr = ^DdptType;                     { Pointeur sur DDPT   }

     PhysDataType = record          { Paramètres physiques de formatage }
       Faces,                                 { Nombre de faces demandé }
       Pistes,                                { Nombre Pistes par Face  }
       Secteurs : byte;                       { Nombre Secteurs par Piste }
       DDPT     : DdptPtr; { Ptr sur table paramètres du lect. de disk.}
     end;

     LogDataType = record                 { Paramètres de formatage DOS }
       Media,                                 { Octet de support }
       Cluster,                             { Nombre Secteurs par Cluster }
       FAT,                                 { Nombre Secteurs pour la FAT }
       RootSize : byte;                { Entrées dans le répertoire racine }
     end;

     PisteBufType = array[ 1..18, 1..512 ] of byte; { Tampon pour piste }

{-- Variables globales initialisées ---------------------------------}

const                           {-- Tables prédéfinies pour chaque format --}

  {-- Données par défaut pour secteur BOOT avec programme chargement --}

      Masqueboot  : array[ 1..102 ] of byte =
                  ( $EB, $35,               { 0000  JMP 0037          }
                    $90,                    { 0002  NOP               }
                  {-- Données des BPB ------------------------------}

                    $50, $43, $49, $4E, $54, $45, $52, $4E,
                    $00, $00, $00, $01, $00, $00, $00, $00,
                    $00, $00, $00, $00, $00, $00, $00, $00,
                    $00, $00, $00, $00, $00, $00, $00, $00,
                    $00, $00, $00, $00, $00, $00, $00, $00,
                    $00, $00, $00, $00, $00, $00, $00, $00,
                    $00, $00, $00, $00,

                    {-- Programme de chargement ----------------------}

                    $FA,                    { 0037  CLI               }
                    $8B, $30, $00,          { 0038  MOV   AX,0030     }
                    $8E, $D0,               { 003B  MOV   SS,AX       }
                    $BC, $FC, $00,          { 003D  MOV   SP,00FC     }
                    $FB,                    { 0040  STI               }
                    $0E,                    { 0041  PUSH  CS          }
                    $1F,                    { 0042  POP   DS          }
                    $BE, $66, $7C,          { 0043  MOV   SI,7C66     }
                    $B4, $0E,               { 0046  MOV   AH,0E       }
                    $FC,                    { 0048  CLD               }
                    $AC,                    { 0049  LODSB             }
                    $0A, $C0,               { 004A  OR    AL,AL       }
                    $74, $04,               { 004C  JZ    0052        }
                    $CD, $10,               { 004E  INT   10          }
                    $EB, $F7,               { 0050  JMP   0049        }
                    $B4, $01,               { 0052  MOV   AH,01       }
                    $CD, $16,               { 0054  INT   16          }
                    $74, $06,               { 0056  JZ    005E        }
                    $B4, $00,               { 0058  MOV   AH,00       }
                    $CD, $16,               { 005A  INT   16          }
                    $EB, $F4,               { 005C  JMP   0052        }
                    $B4, $00,               { 005E  MOV   AH,00       }
                    $CD, $16,               { 0060  INT   16          }
                    $33, $D2,               { 0062  XOR   DX,DX       }
                    $CD, $19 );             { 0064  INT   19          }

      BootMes : string =
       #13#10'DFP  -  (C) 1992 by Michael Tischer'+ #13#10 +
       #13#10'Disquette non système ou défectueuse!'#13#10 +
       'Veuillez changer de disquette et taper une touche' +
       #13#10;

{-- Variables globales non initialisées -----------------------------}

var AktDrive     : byte;            { Numéro du lecteur à formater 0, 1 }
    AktDriveType : byte;        { Type du lecteur de disquettes courant }
    PData        : PhysDataType; { Informations physiques de formatage }
    LData        : LogDataType;   { Informations logiques de formatage }
    AncDDPT      : pointer;              { Pointeur sur ancien DDPT }
    OK           : boolean;       { Drapeau pour exécution du programme }
    EndCode      : word;           { Valeur retournée au process appelé }
    Param        : string;   { pour évaluation de la ligne de commande }

{*********************************************************************}
{* GetDriveType : REtourne le type d'un lecteur de disquettes        *}
{* Entrée       : DRIVE = Numéro de lecteur (0, 1 etc.)              *}
{* Sortie       : Code lecteur comme constante (DD_525, HD_525 etc.) *}
{*********************************************************************}

function GetDriveType( Drive : byte ) : byte;

var Regs     : Registers;{ Registre processeur pour appel interruption }

begin
  Regs.ah := $08;               { Fonction: retourner le type de lecteur }
  Regs.dl := Drive;                                 { Numéro de lecteur }
  intr( $13, Regs );                { Appel d'une interruption du BIOS }
  if ( Regs.flags and fcarry = 0 ) then  { Appel terminé sans erreur? }
    GetDriveType := Regs.bl                          { Type de lecteur }
  else
    GetDriveType := DD_525;            { Fonction $08 de l'interruption }
end;                                  { introuvable => type ordinateur = XT }

{*********************************************************************}
{* ResetDisk  : Reset disque sur tous les lecteurs                   *}
{* Entrée     : aucune                                               *}
{* Sortie     : aucune                                               *}
{* Info       : Le Reset est effectué sur tous les lecteurs indépen- *}
{*              demment du numéro de lecteur chargé dans DL          *}
{*********************************************************************}
```

```
{
procedure DiskReset;

var Regs : Registers;    { Registre processeur pour appel interruption }

begin
  with Regs do
    begin
      ah := $00;          { Numéro de fonction pour appel interruption }
      dl := 0;                                  { Lecteur a: (voir Info) }
    end;
  intr( $13, Regs );                            { appel interruption }
end;

{*****************************************************************}
{* GetFormatParamter: Retourne les paramètres physiques et logiques  *}
{*                    requis par le formatage                    *}
{* Entrée           : FORMSTRING = Capacité demandée sous forme de  *}
{*                    chaine "360", "1200", "720", "1440"          *}
{*                    DRIVETYPE  = Code lecteur tel qu'il est      *}
{*                    retourné par GetDriveType                    *}
{*                    PDATA      = Chargé avec les données de base du *}
{*                    format physique                              *}
{*                    LDATA      = comme PDATA, mais données DOS   *}
{* Sortie           : TRUE si format possible sinon FALSE          *}
{* Info             : Vous pouvez ajouter de nouveaux formats en   *}
{*                    étendant cette procédure                     *}
{*****************************************************************}

function GetFormatParameter(     FormString : string;
                                 DriveType  : byte;
                           var PData        : PhysDataType;
                           var LData        : LogDataType ) : boolean;

const DDPT_360  : DdptType = (~$DF, $02, $25, $02, $09, $2A,
                              $FF, $50, $F6, $0F, $08 );
      DDPT_1200 : DdptType = ( $DF, $02, $25, $02, $0F, $1B,
                              $FF, $54, $F6, $0F, $08 );
      DDPT_720  : DdptType = ( $DF, $02, $25, $02, $09, $2A,
                              $FF, $50, $F6, $0F, $08 );
      DDPT_1440 : DdptType = ( $DF, $02, $25, $02, $12, $1B,
                              $FF, $6C, $F6, $0F, $08 );

      LOG_360  : LogDataType = ( Media : $FD; Cluster  : 2;
                                 FAT   : 2;   RootSize : $70 );
      LOG_1200 : LogDataType = ( Media : $F9; Cluster  : 1;
                                 FAT   : 7;   RootSize : $ED );
      LOG_720  : LogDataType = ( Media : $F9; Cluster  : 2;
                                 FAT   : 3;   RootSize : $70 );
      LOG_1440 : LogDataType = ( Media : $F0; Cluster  : 1;
                                 FAT   : 9;   RootSize : $ED );

      PHYS_360  : PhysDataType = ( Faces   :  2; Pistes : 40;
                                   Secteurs :  9; DDPT   : @DDPT_360 );
      PHYS_1200 : PhysDataType = ( Faces   :  2; Pistes : 80;
                                   Secteurs : 15; DDPT   : @DDPT_1200 );
      PHYS_1440 : PhysDataType = ( Faces   :  2; Pistes : 80;
                                   Secteurs : 18; DDPT   : @DDPT_1440 );
      PHYS_720  : PhysDataType = ( Faces   :  2; Pistes : 80;
                                   Secteurs :  9; DDPT   : @DDPT_720 );

begin
  if ( FormString = '1200' ) then                ( 1,2 MB sur 5,25" ? )
    if ( DriveType = HD_525 ) then   { Lecteur et format compatibles ? }
      begin                                  { Oui, renseigner paramètres }
        PData := PHYS_1200;
        LData := LOG_1200;
        GetFormatParameter := true;                  { Fin sans erreur }
      end
    else
      GetFormatParameter := false    { Lecteur et format incompatibles }
  else if ( FormString = '360' ) then                        { 360 Ko? }
    if ( DriveType = HD_525 ) or ( DriveType = DD_525 ) then
      begin     { Lecteur et format compatibles, renseigner paramètres }
        PData := PHYS_360;
        LData := LOG_360;
        GetFormatParameter := true;                  { Fin sans erreur }
      end
    else
      GetFormatParameter := false    { Lecteur et format incompatibles }
  else if ( FormString = '1440' ) then             { 1,44 MB auf 3,5"? }
    if ( DriveType = HD_35 ) then     { Lecteur et format compatibles? }
      begin                                  { Oui, renseigner paramètres }
        PData := PHYS_1440;
        LData := LOG_1440;
        GetFormatParameter := true;                  { Fin sans erreur }
      end
    else
      GetFormatParameter := false    { Lecteur et format incompatibles }
  else if ( FormString = '720' ) then              { 720 Ko sur 3,5"? }
    if ( DriveType = HD_35 ) or ( DriveType = DD_35 ) then
      begin     { Lecteur et format compatibles, renseigner paramètres }
        PData := PHYS_720;
        LData := LOG_720;
        GetFormatParameter := true;                  { Fin sans erreur }
      end
    else
      GetFormatParameter := false    { Lecteur et format incompatibles }
  else
    GetFormatParameter := false;     { Le format demandé est invalide }
end;

{*****************************************************************}
{* DiskPrepare  : Préparer le lecteur, paramétrer la vitesse de  *}
{*                transfert des données                          *}
{* Entrée       : DRIVE = Numéro de lecteur                      *}
{*                PDATA = Paramètres physiques                   *}
{* Sortie       : aucune                                         *}
{*****************************************************************}

procedure DiskPrepare( Drive : byte; PData : PhysDataType );

var Regs : Registers;    { Registre processeur pour appel interruption }

begin
  {-- Définir le type de support pour appeler formatage ----------}

  with Regs do
    begin
      ah := $18;          { Numéro de fonction pour appel interruption }
      ch := PData.Pistes - 1;                { Nombre Pistes par Face }
      cl := PData.Secteurs;                  { Nombre Secteurs par Piste }
      dl := Drive;                                  { Numéro de lecteur }
    end;
  intr( $13, Regs );                            { appel interruption }
end;

{*****************************************************************}
{* FormatTrack  : Formate une piste                              *}
{* Entrée       : voir ci-dessous                                *}
{* Sortie       : état d'erreur                                  *}
{*****************************************************************}

function Formattrack( Lecteur,       { Numéro du lecteur de disquettes }
                      Face,                        { Numéro de la face }
                      Piste,                       { Piste à formater }
                      Nombre   : byte ) : byte;    { Secteurs par Piste }

type FormatTyp = record
                   DPiste, DFace, DCompteur, DLongueur : byte;
                 end;

var Regs      : Registers;{ Registre processeur pour appel interruption}
    Champdonn : array[ 1..18 ] of FormatTyp;     { maximal 18 Secteurs }
    Compteur  : byte;                            { Compteur d'itérations }
    essais    : byte;                            { Nombre essais maximal }

begin
  for Compteur := 1 to Nombre do
    with Champdonn[ Compteur ] do
      begin
        DPiste := Piste;                              { Numéro de la piste }
        DFace := Face;                               { Face de la disquette }
        DCompteur := Compteur;                        { Numéro du secteur }
        DLongueur := 2;                  { Nombre octets par Secteur (512) }
      end;
  essais := EssaisMax;                  { Définir le nombre d'essais maximal }
  repeat
    with Regs do
      begin
        ah := 5;          { Numéro de fonction pour appel interruption }
        al := Nombre;                      { Nombre de secteurs par piste }
        es := Seg( Champdonn );             { Adresse du Champ de données }
        bx := Ofs( Champdonn );               { dans les registres es:bx }
        dh := Face;                                   { Numéro de la face }
        dl := Lecteur;                                { Numéro de lecteur }
        ch := Piste;                                   { Numéro de piste }
      end;
    intr( $13, Regs );              { Appel d'une interruption du BIOS }
    if ( Regs.flags and fcarry = 1 ) then                    { Erreur? }
      DiskReset;                    { Oui, Reset disquette avant nouvel essai }
    dec( essais );
  until ( Regs.flags and fcarry = 0 ) or ( essais = 0 );
  Formattrack := Regs.ah;                             { Lire état erreur }
end;

{*****************************************************************}
{* VerifyTrack  : Vérifier piste                                 *}
{* Entrée       : Lecteur, Face, Piste, nombre de secteurs       *}
{* Sortie       : Code erreur (0=OK)                             *}
{*****************************************************************}

function VerifyTrack( Lecteur, Face, Piste, Secteurs : byte ) : byte;

var essais    : byte;                            { Nombre essais maximal }
    Regs      : Registers;{ Registre processeur pour appel interruption }
```

```
  TamponPiste : PisteBufType;                ( Mémoire pour une piste )

begin
  essais := EssaisMax;                ( Définir le Nombre d'essais maximal )
  repeat
    with Regs do
      begin
        ah := $04;          ( Numéro de fonction pour appel interruption )
        al := Secteurs;                    ( Nombre Secteurs par Piste )
        ch := Piste;                                ( Numéro de Piste )
        cl := 1;                          ( Commencer par le secteur 1 )
        dl := Lecteur;                              ( Numéro de lecteur )
        dh := Face;                                 ( Numéro de la face )
        es := Seg( TamponPiste );                   ( Adresse du tampon )
        bx := Ofs( TamponPiste );
      end;
    intr( $13, Regs );                          ( Appel interruption BIOS )
    if ( Regs.flags and fcarry = 1 ) then                    ( Erreur? )
      DiskReset;                ( Oui, reset disquette avant nouvel essai )
    dec( essais );
  until ( Regs.flags and fcarry = 0 ) or ( essais = 0 );
  VerifyTrack := Regs.ah;
end;

{*****************************************************************}
{* WriteTrack   : Ecrire piste                                   *}
{* Entrée       : Lecteur, Face, Piste, Secteur début, Nombre, Données*}
{* Sortie       : Code erreur (0=OK)                             *}
{*****************************************************************}

function WriteTrack(     Lecteur, Face, Piste,
                         Start, Nombre          : byte;
                     var Buffer ) : byte;

var essais : byte;                             ( Nombre maximal d'essais )
    Regs   : Registers;{ Registre processeur pour appel interruption )

begin
  essais := EssaisMax;                ( Définir nombre maximal d'essais )
  repeat
    with Regs do
      begin
        ah := $03;          ( Numéro de fonction pour appel interruption )
        al := Nombre;                      ( Nombre Secteurs par Piste )
        ch := Piste;                                ( Numéro de Piste )
        cl := Start;                      ( Commencer par le secteur 1 )
        dl := Lecteur;                              ( Numéro de lecteur )
        dh := Face;                                 ( Numéro de la face )
        es := Seg( Buffer );                        ( Adresse pour tampon )
        bx := Ofs( Buffer );
      end;
    intr( $13, Regs );                ( Appel d'une interruption du BIOS )
    if ( Regs.flags and fcarry = 1 ) then                    ( Erreur? )
      DiskReset;                ( Oui, reset disquette avant nouvel essai )
    dec( essais );
  until ( Regs.flags and fcarry = 0 ) or ( essais = 0 );
  WriteTrack := Regs.ah;
end;

{*****************************************************************}
{* PhysicalFormat : Formatage physique de la disquette (écriture des  *}
{*                  pistes et secteurs)                          *}
{* Entrée         : DRIVE  = Code lecteurs                       *}
{*                  PDATA  = Paramètres physiques                *}
{*                  VERIFY = TRUE pour demander vérification     *}
{* Sortie         : FALSE si erreur, sinon TRUE                  *}
{*****************************************************************}

function PhysicalFormat( Drive  : byte;
                         PData  : PhysDataType;
                         Verify : boolean ) : boolean;

var essais : byte;                             ( Nombre d'essais maximal )
    Regs   : Registers;{ Registre processeur pour appel interruption )
    Piste,                                          ( Piste courante )
    Face,                                           ( Face courante )
    Status : byte;          ( VAleur de retour des fonctions appelées )


begin
  {-- Formatage de la disquette piste par piste ------------------------
  ----}

  for Piste := 0 to PData.Pistes - 1 do     ( Ecrire toutes les pistes )
    for Face := 0 to PData.Faces - 1 do     ( Ecrire toutes les faces )
      begin
        Write( #13'Piste: ', Piste : 2, '  Face: ', Face : 2 );
        {-- 5 essais au maximum pour formater une piste ---------------}

        essais := EssaisMax;                ( Définir nombre maximal d'essais )
        repeat
          Status := FormatTrack( Drive, Face, Piste, PData.Secteurs );
          if ( Status = 3 ) then  ( Disquette protégée en l'écriture ? )
            begin
              PhysicalFormat := false;   ( Quitter procédure si erreur )
              WriteLn( #13'Disquette protégée contre l''écriture' );
              exit;                                 ( Terminer la procédure )
            end;
          if ( Status = 0 ) and Verify then
            Status := VerifyTrack( Drive, Face, Piste, PData.Secteurs );
          dec( essais );
          if ( Status > 0 ) then                ( Le formatage a échoué )
            DiskReset;
        until ( Status = 0 ) or ( essais = 0 );
        if ( Status > 0 )  then             ( Erreur pendant le formatage )
          begin
            PhysicalFormat := false;      ( Quitter procédure si erreur )
            WriteLn( #13'Piste Erreur?aft       ' );
            exit;                                   ( Terminer la procédure )
          end;
      end;
  PhysicalFormat := true;                   ( Procédure quittée sans erreur )
end;

{*****************************************************************}
{* LogicalFormat : Formatage logique de la disquette (écriture du     *}
{*                 secteur de boot, des FAT et du répertoire racine)  *}
{* Entrée        : DRIVE = Numéro de lecteur                      *}
{*                 PDATA = informations physiques de formatage    *}
{*                 LDATA = informations logiques de formatage     *}
{* Sortie        : TRUE si aucune erreur ne s'est produite        *}
{*****************************************************************}

function LogicalFormat( Drive : byte;
                        PData : PhysDataType;
                        LData : LogDataType ) : boolean;

var Status         : byte;                ( Retour de la fonction appelée )
    TousSecteurs   : word;                    ( Nombre total de secteurs )
    i              : byte;                     ( Compteur d'itérations )
    AktSector,
    AktSide,
    AktTrack       : byte;
    Nombre         : integer;   ( Nombre des secteurs restant à écrire )
    TamponPiste    : PisteBufType;            ( Mémoire pour une piste )

begin
  fillchar( TamponPiste, word( PData.Secteurs ) * 512, 0 );{Vide tampon}

  {-- Secteur de boot : partie fixe ------------------------------------}

  move( Masqueboot, TamponPiste, 102 );   ( Copier masque secteur boot )
  move( BootMes[1], TamponPiste[ 1, 103 ],       ( Copie textes de boot )
        ord(BootMes[0]) );
  TamponPiste[ 1, 511 ] := $55;       ( Signe de fin du secteur de boot )
  TamponPiste[ 1, 512 ] := $AA;

  {-- Secteur de boot : partie variable --------------------------------}

  TousSecteurs := PData.Pistes * PData.Secteurs * Pdata.Faces;
  TamponPiste[ 1, 14 ] := LData.Cluster;                 ( Longueur Cluster )
  TamponPiste[ 1, 18 ] := LData.RootSize; ( Nombre Entrées ds rép.rac. )
  TamponPiste[ 1, 20 ] := lo( TousSecteurs );{ Nombre total de secteur )
  TamponPiste[ 1, 21 ] := hi( TousSecteurs );          ( sur la disquette )
  TamponPiste[ 1, 22 ] := LData.Media;             { descripteur support }
  TamponPiste[ 1, 23 ] := LData.FAT;                   ( Longueur des FAT )
  TamponPiste[ 1, 25 ] := PData.Secteurs;            ( Secteurs par piste )
  TamponPiste[ 1, 27 ] := PData.Faces;                  ( Nombre de faces )

  {-- Créer FAT et sa copie (contient 00) ------------------------------}

  TamponPiste[ 2, 1 ] := LData.Media;                      ( Créer 1ère FAT )
  TamponPiste[ 2, 2 ] := $FF;
  TamponPiste[ 2, 3 ] := $FF;
  TamponPiste[ LData.FAT + 2, 1 ] := LData.Media;      ( Créer 2ème FAT )
  TamponPiste[ LData.FAT + 2, 2 ] := $FF;
  TamponPiste[ LData.FAT + 2, 3 ] := $FF;

  {-- Boot-Sektor und FAT schreiben ------------------------------------}

  Status := WriteTrack( Drive, 0, 0, 1, PData.Secteurs, TamponPiste );
  if Status <> 0 then
    LogicalFormat := FALSE

  {-- Pas d'erreur? Ecrire répertoire racine ---------------------------}

  else
    begin
      fillchar( TamponPiste, 512, 0 );                      ( Secteur vide )
      AktSector := PData.Secteurs;       ( Ecriture 1ère piste complète )
      AktTrack := 0;                                    ( Piste courante )
      AktSide := 0;                                     ( Face courante )

      {-- Retourner nombre des secteurs restants et les écrire --------}

      Nombre := LData.FAT * 2 + ( LData.Rootsize * 32 div 512 ) +
```

```
                1 - PData.Secteurs;

      i := 1;
      repeat
        inc( AktSector );                                  ( Secteur suivant )
        if ( AktSector > PData.Secteurs ) then      ( Fin de la piste? )
          begin
            AktSector := 1;                        ( Continuer par secteur 1 )
            inc( AktSide );                              ( Face suivante ? )
            if ( AktSide = PData.Faces ) then       ( Déjà 2ème face ? )
              begin
                AktSide := 0;                       ( Revenons à la face 0 )
                inc( AktTrack );
              end;
          end;
        Status := WriteTrack( Drive, AktSide, AktTrack,
                              AktSector, 1, TamponPiste );
        inc( i );
      until ( i > Nombre ) or ( status <> 0 );
      LogicalFormat := ( status = 0 )
    end;
end;

{*********************************************************************}
{*                    PROGRAMM  PRINCIPAL                            *}
{*********************************************************************}

begin
  WriteLn( 'DFP  -  (c) 1992 by Michael Tischer'#13#10 );
  if paramcount > 1 then                          ( Donné paramètres ? )
    begin                                                ( Oui, évaluer )
      Param := paramstr( 1 );   ( REtourner lecteur ( 0 = a:, 1 = b: ) )
      AktDrive := ord( upcase( Param[ 1 ] ) ) - 65;
      AktDriveType := GetDriveType( AktDrive ); ( Type lecteur courant )
      if ( AktDriveType > 0 ) then                  ( Lecteur existant? )
        begin                                ( Oui, le programme peut continuer )
          if GetFormatParameter( paramstr( 2 ), AktDriveType,
                                 PData, LData) then
            begin                    ( Format et lecteur sont compatibles )
              DiskPrepare( AktDrive, PData );
              GetIntVec( $1E, AncDDPT );                ( Stocker anc DDPT )
              SetIntVec( $1E, PData.DDPT );     ( Définir nouvelle DDPT )

              Param := paramstr( 3 );
              ok := PhysicalFormat( AktDrive, PData,
                                    upcase( Param[ 1 ] ) <> 'N' );
              if ok then
                begin
                  WriteLn(#13'Ecriture du secteur de boot et des FAT ');
                  ok := LogicalFormat( AktDrive, PData, LData )
                end;

                      {-- Evaluation du formatage --------------------}

              if ok then
                begin
                  WriteLn( #13'Formatage o.k.                    ' );
                  EndCode := 0;
                end
              else
                begin
                  WriteLn( #13'Une erreur a interrompu le formatage' );
                  EndCode := 1;
                end;
              SetIntVec( $1E, AncDDPT );            ( Restaurer anc DDPT )
            end
          else
            begin
              WriteLn( 'Le format demandé ne peut être ' +
                       'utilisé sur ce lecteur!' );
              EndCode := 2;   ( Valeur de retour au processus appelant )
            end
        end
      else
        begin
          WriteLn( 'Le lecteur de disquettes demandé n''existe pas');
          EndCode := 3;         ( Valeur de retour au processus appelant )
        end
    end
  else
    begin
      WriteLn( 'Appel: DFP Lecteur   Format   [ NV ]' );
      WriteLn( '                                    ' );
      WriteLn( '                                   ' );
      WriteLn( '   A: ou B:                        ' );
      WriteLn( '                                    ' );
      WriteLn( '  360, 720, 1200, 1440              ' );
      WriteLn( '                                    ' );
      WriteLn( '                  NV = pas Verify ' );
      EndCode := 4;           ( Valeur de retour au processus appelant )
    end;
  Halt( EndCode );
end.
```

Listing : DFC.C

```
/*********************************************************************/
/*                              D F C                                */
/*-------------------------------------------------------------------*/
/*    SUJET       : Formate disquettes 3.5" et 5,25"                 */
/*-------------------------------------------------------------------*/
/*    Auteur              : Michael Tischer                          */
/*    Développé le        : 28.08.1991                               */
/*    Dernière Modification : 26.01.1992                             */
/*-------------------------------------------------------------------*/
/*    Modèle mémoire: SMALL                                          */
/*-------------------------------------------------------------------*/
/*    Attention       : Le message d'avertissement                   */
/*                      "warning C4059: segment lost in conversion"  */
/*                      du compilateur Microsoft est voulu.          */
/*********************************************************************/

/*-- Intègre les fichiers include ---------------------------------*/

#include <dos.h>
#include <stdio.h>
#include <string.h>

/*-- macros -------------------------------------------------------*/

#ifdef MK_FP                            /* macro MK_FP déjà définie ? */
  #undef MK_FP                          /* Oui, alors effacer macro */
#endif

#define MK_FP(seg,ofs) ((void far *) ((unsigned long) (seg)<<16|(ofs)))
#define LO( valeur ) ( ( BYTE ) ( valeur & 0xFF ) )
#define HI( valeur ) ( ( BYTE ) ( valeur >> 8 ) )
#define SEG( p ) ( ( unsigned int ) ( ( ( long ) p ) >> 16 ) )
#define OFS( p ) ( ( unsigned int ) ( p ) )

/*-- Konstanten ---------------------------------------------------*/

#define NDN          0x4E                          /* N = NON */
#define NO_DRIVE     0                        /* Lecteur introuvable */
#define DD_525       1                          /* Lecteur: 5,25" DD */
#define HD_525       2                          /* Lecteur: 5,25" HD */
#define DD_35        3                          /* Lecteur: 3,5" DD */
#define HD_35        4                          /* Lecteur: 3,5" HD */

#define EssaisMax  5                    /* Nombre maximal d'essais */

#define TRUE       ( 0 == 0 )           /* les constantes facilitent */
#define FALSE      ( 1 == 0 )           /* la lecture du listing     */

/*-- Typedefs -----------------------------------------------------*/

typedef unsigned char BYTE;                     /* Type données Byte */

typedef BYTE DDPTType[ 11 ];                    /* Champ pour une DDPT */
typedef DDPTType *DDPTPTR;                      /* Pointeur sur une DDPT */

typedef struct {                        /* Paramètres physiques de formatage */
    BYTE    Faces,                              /* Nombre de Faces */
      Pistes,                           /* Pistes par face */
      Secteurs;                         /* Secteurs par Piste */
    DDPTPTR DDPT;                       /* Pointeur sur DDPTR */
  } PhysDataType;

typedef struct {                        /* Paramètres logiques de formatage */
    BYTE Media;                         /* Octet de support */
    BYTE Cluster;                       /* Nombre Secteurs par Cluster */
    BYTE FAT;                           /* Nombre Secteurs pour la FAT */
    BYTE RootSize;                      /* Entrées dans répertoire racine */
  } LogDataType;

typedef BYTE PisteBufType[ 18 ][ 512 ];         /* Mémoire pour une piste */
```

```
/*-- Variables globales ------------------------------------------------*/

/*-- Membre invariable du secteur de boot ------------------------------*/

BYTE Masqueboot[ 102 ] =
        { 0xEB, 0x35,           /* 0000   JMP 0037      */
          0x90,                 /* 0002   NOP           */
          /*-- Données du BPB ------------------------------*/

          0x50, 0x43, 0x49, 0x4E, 0x54, 0x45, 0x52, 0x4E,
          0x00, 0x00, 0x00, 0x01, 0x00, 0x00, 0x00, 0x00,
          0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
          0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
          0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
          0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
          0x00, 0x00, 0x00, 0x00,

          /*-- Programme de chargement ---------------------*/

          0xFA,                 /* 0037   CLI           */
          0xB8, 0x30, 0x00,     /* 0038   MOV   AX,0030 */
          0x8E, 0xD0,           /* 003B   MOV   SS,AX   */
          0xBC, 0xFC, 0x00,     /* 003D   MOV   SP,00FC */
          0xFB,                 /* 0040   STI           */
          0x0E,                 /* 0041   PUSH  CS      */
          0x1F,                 /* 0042   POP   DS      */
          0xBE, 0x66, 0x7C,     /* 0043   MOV   SI,7C66 */
          0xB4, 0x0E,           /* 0046   MOV   AH,0E   */
          0xFC,                 /* 0048   CLD           */
          0xAC,                 /* 0049   LODSB         */
          0x0A, 0xC0,           /* 004A   OR    AL,AL   */
          0x74, 0x04,           /* 004C   JZ    0052   */
          0xCD, 0x10,           /* 004E   INT   10      */
          0xEB, 0xF7,           /* 0050   JMP   0049    */
          0xB4, 0x01,           /* 0052   MOV   AH,01   */
          0xCD, 0x16,           /* 0054   INT   16      */
          0x74, 0x06,           /* 0056   JZ    005E    */
          0xB4, 0x00,           /* 0058   MOV   AH,00   */
          0xCD, 0x16,           /* 005A   INT   16      */
          0xEB, 0xF4,           /* 005C   JMP   0052    */
          0xB4, 0x00,           /* 005E   MOV   AH,00   */
          0xCD, 0x16,           /* 0060   INT   16      */
          0x33, 0xD2,           /* 0062   XOR   DX,DX   */
          0xCD, 0x19 };         /* 0064   INT   19      */

char BootMes[] =
  "\nDFC  -  (C) 1992 by Michael Tischer\n\n" \
  "Disquette non système ou défectueuse!\n" \
  "Veuillez changer de disquette et taper une touche" \
  "\n\n";

/*****************************************************************/
/* upcase       : Convertit un caractère en majuscule            */
/* Entrée       : Caractère                                      */
/* Sortie       : Majuscule                                      */
/*****************************************************************/

char upcase( char letter )
{                                                   /* XXX */
  if ( ( letter > 0x60 ) && ( letter < 0x7B ) )     /* convertir ? */
    return (unsigned char) letter & 0xDF;           /* Oui, masquer bit */
  else
    return letter;                                  /* NON, retourner tel quel */
}

/*****************************************************************/
/* GetIntVec : lit un vecteur d'interruption                     */
/* Entrée    : NUMERO = Numéro d'interruption                    */
/* Sortie    : Vecteur d'interruption                            */
/*****************************************************************/

void far *GetIntVec( int Numero )
{
  return( *( ( void far * far * ) MK_FP( 0, Numero * 4 ) ) );
}

/*****************************************************************/
/* SetIntVec : définit un vecteur d'interruption                 */
/* Entrée    : NUMERO = Numéro d'interruption                    */
/*             Pointeur = Vecteur d'interruption                 */
/* Sortie    : aucune                                            */
/*****************************************************************/

void SetIntVec( int Numero, void far *Pointeur )
{
  *( ( void far * far * ) MK_FP( 0, Numero * 4 ) ) = Pointeur;
}

/*****************************************************************/
/* GetDriveType : Retourne le type d'un lecteur de disquettes    */
/* Entrée       : DRIVE = Numéro de lecteur (0, 1 etc.)          */
/* Sortie       : Codes lecteurs en constantes (DD_525, HD_525 etc.) */
/*****************************************************************/

BYTE GetDriveType( BYTE Drive )
{
  union REGS regs;      /* Registre processeur pour appel interruption */

  regs.h.ah = 0x08;                 /* Fonction: retourner le lecteur */
  regs.h.dl = Drive;                /* Appeler le numéro lecteur de */
  int86( 0x13, &regs, &regs );      /* l'interruption BIOS */
  if ( regs.x.cflag )               /* Erreur à l'appel ? */
    return( DD_525 );               /* Fonct. 0x08 introuvable => XT 360 Ko */
  else
    return( regs.h.bl );            /* Type de lecteur */
}

/*****************************************************************/
/* ResetDisk : Reset disque sur tous les lecteurs                */
/* Entrée    : aucune                                            */
/* Sortie    : aucune                                            */
/* Info      : Indépendamment du numéro de lecteur chargé dans DL, */
/*             le Reset est effectué sur tous les lecteurs de    */
/*             disquettes                                        */
/*****************************************************************/

void DiskReset( void )
{
  union REGS regs;      /* Registre processeur pour appel interruption */

  regs.h.ah = 0x00;     /* Numéro de fonction pour appel interruption */
  regs.h.dl = 0;                    /* Lecteur a: (s. Info) */
  int86( 0x13, &regs, &regs );      /* appel interruption */
}

/*****************************************************************/
/* GetFormatParamter : Retourne les paramètres logiques et physiques */
/*                     requis pour le formatage                  */
/* Entrée            : FORMSTRING = Pointeur sur chaîne de format */
/*                     "360", "720", "1200", "1440"              */
/*                     DRIVETYPE  = Code lecteur retourné par    */
/*                     GetDriveType()                            */
/*                     PDATAP     = Pointeur sur structure contenant */
/*                     les paramètres physiques de formatage     */
/*                     LDATAP     = Pointeur sur structure contenant */
/*                     les paramètres logiques de formatage      */
/* Sortie            : TRUE si format possible, sinon FALSE      */
/* Info              : Vous pouvez ajouter de nouveaux formats en */
/*                     complétant cette procédure                */
/*****************************************************************/

BYTE GetFormatParameter( char         *FormString,
      BYTE         DriveType,
      PhysDataType *PDataP,
      LogDataType  *LDataP )

{
  static DDPTType DDPT_360  = { 0xDF, 0x02, 0x25, 0x02, 0x09, 0x2A,
          0xFF, 0x50, 0xF6, 0x0F, 0x08 };
  static DDPTType DDPT_1200 = { 0xDF, 0x02, 0x25, 0x02, 0x0F, 0x1B,
          0xFF, 0x54, 0xF6, 0x0F, 0x08 };
  static DDPTType DDPT_1440 = { 0xDF, 0x02, 0x25, 0x02, 0x12, 0x1B,
          0xFF, 0x6C, 0xF6, 0x0F, 0x08 };
  static DDPTType DDPT_720  = { 0xDF, 0x02, 0x25, 0x02, 0x09, 0x2A,
          0xFF, 0x50, 0xF6, 0x0F, 0x08 };

  static LogDataType LOG_360  = { 0xFD, 2, 2, 0x70 };
  static LogDataType LOG_1200 = { 0xF9, 1, 7, 0xED };
  static LogDataType LOG_720  = { 0xF9, 2, 3, 0x70 };
  static LogDataType LOG_1440 = { 0xF0, 1, 9, 0xED };

  static PhysDataType PHYS_360  = ( 2, 40,  9, &DDPT_360 );
  static PhysDataType PHYS_1200 = ( 2, 80, 15, &DDPT_1200 );
  static PhysDataType PHYS_720  = ( 2, 80,  9, &DDPT_720 );
  static PhysDataType PHYS_1440 = ( 2, 80, 18, &DDPT_1440 );

  /*-- Lire le format dans la chaîne et stocker les données dans les ---*/
  /*-- structures indiquées                                        ---*/

  if ( strcmp( FormString, "1200" ) == 0 )      /* 1,2 MB sur 5,25"? */
    if ( DriveType == HD_525 )      /* Format compatible avec lecteur ? */
    {
      memcpy( PDataP, &PHYS_1200, sizeof( PhysDataType ) );
      memcpy( LDataP, &LOG_1200, sizeof ( LogDataType ) );
      return TRUE;                              /* Terminé sans erreur */
    }
    else
      return( FALSE );                  /* Lecteur et format incompatibles */
  else if ( strcmp( FormString, "360" ) == 0 )  /* 360 Ko ? */
    if ( ( DriveType == HD_525 ) || ( DriveType == DD_525 ) )
    {           /* Format et lecteur compatibles, renseigner paramètres */
      memcpy ( PDataP, &PHYS_360, sizeof( PhysDataType ) );
      memcpy ( LDataP, &LOG_360, sizeof ( LogDataType ) );
      return TRUE;                              /* Fin sans erreur */
    }
    else
      return( FALSE );                  /* Lecteur et format incompatibles */
```

```
  else if ( strcmp( FormString, "1440" ) == 0 )  /* 1.44 MB auf 3,5"? */
   if ( DriveType == HD_35 )        /* Lecteur et format compatibles? */
    {                               /* Lecteur et format compatibles, Parametre */
     memcpy ( PDataP, &PHYS_1440, sizeof( PhysDataType ) );
     memcpy ( LDataP, &LOG_1440, sizeof ( LogDataType ) );
     return TRUE;                                   /* Fin sans erreur */
    }
    else
     return( FALSE );               /* Lecteur et format incompatibles */
  else if ( strcmp( FormString, "720" ) == 0 )    /* 720 KB auf 3,5"? */
    if ( ( DriveType == HD_35 ) || ( DriveType == DD_35 ) )
    {       /* Lecteur et format compatibles, renseigner paramètres */
     memcpy ( PDataP, &PHYS_720, sizeof( PhysDataType ) );
     memcpy ( LDataP, &LOG_720, sizeof ( LogDataType ) );
     return TRUE;                                   /* Fin sans erreur */
    }
    else
     return FALSE;                  /* Lecteur et format incompatibles */
  else
    return FALSE;                   /* Le format demandé est invalide */
}

/*****************************************************************/
/* DiskPrepare: Prépare le lecteur, paramètre la vitesse de transfert */
/* Entrée     : DRIVE = Numéro de lecteur                        */
/*              PDATA = Table des paramètres physiques           */
/* Sortie     : aucune                                           */
/*****************************************************************/

void DiskPrepare( BYTE Drive, PhysDataType PData )
{
 union REGS regs;       /* Registre processeur pour appel interruption */

 /*-- Type de support pour appel formatage ---------------------------*/

 regs.h.ah = 0x18;      /*  Numéro de fonction pour appel interruption */
 regs.h.ch = PData.Pistes - 1;              /* Nombre Pistes par Face */
 regs.h.cl = PData.Secteurs;              /* Nombre Secteurs par Piste */
 regs.h.dl = Drive;                             /* Numéro de lecteur */
 int86( 0x13, &regs, &regs );                   /* appel interruption */
}

/*****************************************************************/
/* FormatTrack: Formate eine Piste                               */
/* Entrée     : voir plus loin                                   */
/* Sortie     : Les statistiques d'erreurs                       */
/*****************************************************************/

BYTE FormatTrack( BYTE Lecteur,                         /* Numéro de lecteur */
     BYTE Face,                         /* Numéro de face */
     BYTE Piste,                                /* la piste */
     BYTE Nombre )   /* Nombre Secteurs pour cette piste */

{
 struct FormatTyp {         /* Informations sur le secteur pour le BIOS */
     BYTE DPiste, DFace, DCompteur, DLongueur;
    };

 BYTE             essais;/* Nombre tentatives pour appel interruption */
 BYTE             Compteur;                  /* compteur d'itérations */
 struct FormatTyp Champdonn[ 18 ];                  /* 18 Secteurs maxi */
 void far *       dfp = Champdonn;   /* Pointeur sur champ de données */
 union REGS       regs;  /* Registre processeur pr appel interruption */
 struct SREGS     sregs;                          /* Segmentregister */

 for ( Compteur = 0; Compteur < Nombre; Compteur++ )
  {
   Champdonn[ Compteur ].DPiste = Piste;
   Champdonn[ Compteur ].DFace = Face;
   Champdonn[ Compteur ].DCompteur = Compteur + 1;
   Champdonn[ Compteur ].DLongueur = 2;     /* 512 octets par Secteur */
  }

 essais = EssaisMax;                /* Définir le nombre maximal d'essais */
 do
  {
   regs.h.ah = 5;           /* Numéro de fonction pour appel interruption */
   regs.h.al = Nombre;                  /* Nombre de secteurs par piste */
   regs.x.bx = OFS( dfp );                /* Offsetadresse du tampon */
   sregs.es = SEG( dfp );                          /* Adresse segment */
   regs.h.dh = Face;                               /* Numéro de la face */
   regs.h.dl = Lecteur;                            /* Numéro de lecteur */
   regs.h.ch = Piste;                              /* Numéro de la piste */
   int86x( 0x13, &regs, &regs, &sregs );  /* Appel interruption BIOS */
   if ( regs.x.cflag )                                      /* Erreur? */
     DiskReset();
  }
 while ( ( --essais != 0 ) && ( regs.x.cflag ) );
 return( regs.h.ah );                                   /* Lit état erreur */
}

/*****************************************************************/
/* VerifyTrack : Vérifie la piste                                */
/* Entrée      : voir plus loin                                  */
/* Sortie      : état erreur                                     */
/*****************************************************************/

BYTE VerifyTrack( BYTE Lecteur,                         /* Numéro de lecteur */
     BYTE Face,                         /* Numéro de face */
     BYTE Piste,                        /* Numéro piste */
     BYTE Secteurs )        /* Nombre Secteurs par Piste */

{
 BYTE         essais;        /* Nombre essais pour appel interruption */
 union REGS   regs;    /* Registre processeur pour appel interruption */
 struct SREGS sregs;  /* Registre process. appel étendu d'interruption */
 PisteBufType sbuf;                                     /* Tampon piste */
 void far     *sbptr = sbuf;        /* Pointeur FAR sur tampon piste */

 essais = EssaisMax;            /* Définir le nombre limite des essais */
 do
  {
   regs.h.ah = 0x04;    /* Numéro de fonction pour appel interruption */
   regs.h.al = Secteurs;                    /* Nombre Secteurs par Piste */
   regs.h.ch = Piste;                                 /* Numéro de la piste */
   regs.h.cl = 1;                                  /* Commence au Secteur 1 */
   regs.h.dl = Lecteur;                               /* Numéro de lecteur */
   regs.h.dh = Face;                                  /* Numéro de la face */
   regs.x.bx = OFS( sbptr );                /* Adresse Offset du tampon */
   sregs.es = SEG( sbptr );                             /* Adresse Segment */
   int86x( 0x13, &regs, &regs, &sregs );  /* Appel interruption BIOS */
   if ( regs.x.cflag )                                      /* Erreur? */
     DiskReset();
  }
 while ( ( --essais != 0 ) && ( regs.x.cflag ) );
 return( regs.h.ah );                                   /* Lire état erreur */
}

/*****************************************************************/
/* WriteTrack: Ecrire piste                                      */
/* Entrée    : voir ci-dessous                                   */
/* Sortie    : Code erreur (0=OK)                                */
/*****************************************************************/

BYTE WriteTrack( BYTE Lecteur,                          /* Numéro de lecteur */
     BYTE Face,                         /* Numéro de la face */
     BYTE Piste,                        /* Numéro de la piste */
     BYTE Start,                    /* Commence par le secteur */
     BYTE Secteurs,             /* Nombre Secteurs par Piste */
     void far *Donnees ) /* Pointeur sur champ de données */

{
 BYTE essais;                   /* Nombre essais pour appel interruption */
 union REGS regs;       /* Registre processeur pour appel interruption */
 struct SREGS sregs;    /* Registre processeur pour appel étendu int. */

 essais = EssaisMax;                    /* Définir nombre maxi d'essais */
 do
  {
   regs.h.ah = 0x03;    /* Numéro de fonction pour appel interruption */
   regs.h.al = Secteurs;                    /* Nombre Secteurs par Piste */
   regs.h.ch = Piste;                                 /* Numéro de la piste */
   regs.h.cl = Start;                          /* Commencer par le secteur */
   regs.h.dl = Lecteur;                               /* Numéro de lecteur */
   regs.h.dh = Face;                                  /* Numéro de la face */
   regs.x.bx = OFS( Donnees );              /* Adresse Offset du buffer */
   sregs.es = SEG( Donnees );                           /* Adresse Segment */
   int86x( 0x13, &regs, &regs, &sregs );/* Appel interruption du BIOS */
   if ( regs.x.cflag )                                      /* Erreur? */
     DiskReset();
  }
 while ( ( --essais != 0 ) && ( regs.x.cflag ) );
 return( regs.h.ah );                                   /* Lire état erreur */
}

/*****************************************************************/
/* PhysicalFormat : Formatage physique de la disquette (écriture des */
/*                  pistes et des secteurs)                      */
/* Entrée         : voir ci-dessus                               */
/* Sortie         : Formatage terminé sans errreur               */
/*****************************************************************/

BYTE PhysicalFormat( BYTE         Drive,        /* Numéro de lecteur */
         PhysDataType PData,      /* Paramètres physiques */
         BYTE         Verify )          /* Flag de Verify */

{
 union REGS regs;       /* Registre processeur pour appel interruption */
 BYTE       essais,          /* Nombre d'essais pour appel interruption */
        Piste,           /* Compteur itérations : Piste courante */
        Face,            /* Compteur itérations : face courante */
        Status;       /* VAleur retournée par la fonction appelée */

 /*-- Formatage de la disquette piste par piste ----------------------*/

 for ( Piste = 0; Piste < PData.Pistes; Piste++ )
```

```
  for ( Face = 0; Face < PData.Faces; Face++ )
   {
    printf( "\rPiste: %d  Face: %d", Piste, Face );
     /*-- Maximal 5 essais de formatage d'une piste ------------------*/

    essais = EssaisMax;              /* Définir nombre maximal d'essais */
    do
     {
      Status = FormatTrack( Drive, Face, Piste, PData.Secteurs );
      if ( Status == 3 )    /* Disquette protégée contre l'écriture ? */
       {
   printf( "\rDisquette protégée contre l'écriture" );
   return FALSE;                 /* Procédure terminée sur erreur */
       }
      if ( Status == 0  &&  Verify )
       Status = VerifyTrack( Drive, Face, Piste, PData.Secteurs );
      if ( Status > 0 )                          /* Le formatage a échoué */
       DiskReset();
     }
    while ( ( --essais != 0 ) && ( Status != 0 ) );
    if ( Status > 0 )                    /*  Erreur pendant le formatage */
     {
      printf( "\rPiste Erreur?       \n" );
      return FALSE;                    /* Procédure terminée sur erreur */
     }
   }
  return TRUE;                         /* Procédure terminée sans erreur */
}

/*****************************************************************/
/* LogicalFormat : Formatage logique de la disquette : écriture des  */
/*                 secteurs de boot, des FAT et du répertoire racine */
/* Entrée        : voir ci-dessous                                   */
/* Sortie        : TRUE si aucune erreur apparue                     */
/*****************************************************************/

BYTE LogicalFormat( BYTE Drive,                      /* Numéro de lecteur */
        PhysDataType PData,       /* Paramètres physiques */
        LogDataType LData )       /* Paramètres logiques */

{
 BYTE       i,                                /* Compteur d'itérations */
       AktSector,
       AktSide,
       AktTrack,
       Status;
 int        TousSecteurs,                     /* Nombre total de secteurs */
       Nombre;          /* Nombre de secteurs restant à écrire */
 PisteBufType TamponPiste;                    /* stocke une piste entière */

 memset(TamponPiste,0,(int) PData.Secteurs * 512 ); /* Vider la piste */

 /*-- Secteur de boot : partie fixe ------------------------------*/

 memcpy( TamponPiste, Masqueboot, 102 );/* Copie masque du sect. boot */
 memcpy( &TamponPiste[ 0 ][ 102 ], BootMes, sizeof( BootMes) );
 TamponPiste[ 0 ][ 510 ] = 0x55;   /* Signe de fin du secteur de boot */
 TamponPiste[ 0 ][ 511 ] = 0xAA;

 /*-- Secteur de boot : partie variable --------------------------*/

 TousSecteurs = (int) PData.Pistes * (int) PData.Secteurs *
      (int) PData.Faces;      /* Nombre total de secteurs */
 TamponPiste[ 0 ][ 13 ] = LData.Cluster;          /* taille cluster */
 TamponPiste[ 0 ][ 17 ] = LData.RootSize; /* Nbre entrées ds rép. rac.*/
 TamponPiste[ 0 ][ 19 ] = LO( TousSecteurs );
 TamponPiste[ 0 ][ 20 ] = HI( TousSecteurs );
 TamponPiste[ 0 ][ 21 ] = LData.Media;        /* Descripteur support */
 TamponPiste[ 0 ][ 22 ] = LData.FAT;           /* Longueur de la FAT */
 TamponPiste[ 0 ][ 24 ] = PData.Secteurs;      /* Secteurs par Piste */
 TamponPiste[ 0 ][ 26 ] = PData.Faces;           /* Nombre de faces */

 /*-- Créer FAT et la copier -------------------------------------*/

 TamponPiste[ 1 ][ 0 ] = LData.Media;          /* Création 1ère FAT */
 TamponPiste[ 1 ][ 1 ] = 0xFF;
 TamponPiste[ 1 ][ 2 ] = 0xFF;
 TamponPiste[ LData.FAT + 1 ][ 0 ] = LData.Media;/* Création 2ème FAT */
 TamponPiste[ LData.FAT + 1 ][ 1 ] = 0xFF;
 TamponPiste[ LData.FAT + 1 ][ 2 ] = 0xFF;

 /*-- Ecrire secteur de boot et FAT ------------------------------*/

 Status = WriteTrack( Drive, 0, 0, 1, PData.Secteurs, TamponPiste );
 if ( Status )                              /* Erreur en écriture ? */
  return FALSE;                            /* OUI! Retourner Erreur */

 /*-- Ecriture répertoire racine ---------------------------------*/

 memset( TamponPiste, 0, 512 );                   /* Secteur vide */
 AktSector = PData.Secteurs;     /* Première piste entièrement écrite */
 AktTrack = 0;                                    /* Piste courante */
 AktSide = 0;                                     /* Face courante */

  /*-- Retourner le nombre des secteurs retants et les écrire ------*/

 Nombre = LData.FAT * 2 + (LData.RootSize*32/512) + 1-PData.Secteurs;

 for ( i = 1; i <= Nombre; i++ )
  {
   if ( ++AktSector > PData.Secteurs )          /* Fin de la piste ? */
    {
     AktSector = 1;                           /* Commencer au secteur 1 */
     if ( ++AktSide == PData.Faces )               /* déjà 2ème face? */
      {
       AktSide = 0;                            /* Retour à la face 0 */
       AktTrack++;
      }
    }
   Status = WriteTrack( Drive, AktSide, AktTrack,
     AktSector, 1, TamponPiste );
   if ( Status )                                                /* Erreur? */
    break;                  /* Oui, quitter prématurément la boucle FOR */
  }
 return ( Status == 0 );
}

/*****************************************************************/
/*                      PROGRAMM PRINCIPAL                       */
/*****************************************************************/

int main( argc, argv )

int argc;                   /* Nombre d'arguments dans ligne de commande */
char *argv[];                                  /* Champ de paramètres */

{
 BYTE        AktDrive;                  /* Numéro du lecteur à formater */
 BYTE        AktDriveType;  /* Type du lecteur de disquettes courant */
 PhysDataType PData;                 /* Paramètres physiques de formatage */
 LogDataType LData;                  /* Paramètres logiques de formatage */
 void far    *AncDDPT;                          /* Pointeur sur ancien DDPT */
 char        *Param;        /* pour évaluation de la ligne de commande */
 BYTE        ok;                 /* Drapeau pour exécution du programme */
 int         ExitCode;

 printf( "DFC  -  (c) 1992 by Michael Tischer\n\n" );

 /*-- Evaluation ligne de commande ------------------------------*/

 if ( argc > 1 )                                  /* Donné paramètres ? */
  {                                                          /* Oui */
   Param = argv[ 1 ];           /* Retourne lecteur ( 0 = a:, 1 = b: ) */
   AktDrive = upcase( Param[ 0 ] ) - 65;
   AktDriveType = GetDriveType( AktDrive );  /* Type lecteur courant */
   if ( AktDriveType > 0 )                        /* Lecteur existant? */
    if (GetFormatParameter( argv[ 2 ], AktDriveType, &PData, &LData ))
     {
      DiskPrepare( AktDrive, PData );
      AncDDPT = GetIntVec( 0x1E );              /* Stocker ancien DDPT */
      SetIntVec( 0x1E, PData.DDPT );           /* Définir nouveau DDPT */

      Param = argv[ 3 ];
      if ( ok = PhysicalFormat( AktDrive, PData,
    (BYTE) ( upcase( Param[ 0 ] ) != 'N' ) ) )
 {
  printf( "\rEcriture du secteur de boot et de la FAT      \n" );
  ok = LogicalFormat( AktDrive, PData, LData );
 }

      /*-- Evaluation du formatage ------------------------------*/

      if ( ok )
 {
  printf( "\rFormatage o.k.                \n" );
  ExitCode = 0;                 /* Programme quitté sans problème */
 }
      else
 {
  printf( "\rUne erreur a interrompu le formatage\n ");
  ExitCode = 1;          /* Une erreur a interrompu le formatage */
 }
      SetIntVec( 0x1E, AncDDPT );              /* Restaure ancien DDPT */
     }
    else
     {
      printf( "Le format demandé ne peut pas " \
        "être utilisé sur ce lecteur!\n" );
      ExitCode = 2;                 /* Lecteur et format incompatibles */
     }
   else
    {
     printf( "Le lecteur indiqué n'existe pas!\n" );
     ExitCode = 3;                                 /* Lecteur introuvable */
    }
}
```

```
    printf( "\r      \n" );
    printf( "\r      NV = pas de Verify \n" );     /* Appel erroné */
    ExitCode = 4;
  }
return( ExitCode );          /* Termine programme par valeur de retour */
}

else
  {
  printf( "\rAppel: DFC Lecteur    Format  [ NV ]\n" );
  printf( "\r                        \n" );
  printf( "\r                        \n" );
  printf( "\r  A: ou B:              \n" );     \n" );
  printf( "\r  360, 720, 1200, 1440
```

6.4. Accès au disque dur à l'aide du BIOS.

Cette section décrit les différentes fonctions proposées par le BIOS pour l'accès au disque dur. Avant d'examiner les fonctions que le BIOS offre pour l'accès au disque dur, nous devons absolument vous avertir des dangers à jouer avec ces fonctions de façon trop enthousiaste. Sur un lecteur de disquettes vous pouvez toujours, en effet, introduire dans le lecteur une disquette vierge et lui faire subir tous les tests que vous souhaitez. Sur un disque dur, c'est naturellement impossible et un emploi irréfléchi des fonctions d'écriture ou de formatage peut entraîner des pertes de données considérables et irréversibles. En raison de la structure créée par le DOS sur un disque dur, il suffit d'un secteur détruit pour faire disparaître soudainement l'ensemble des fichiers et des répertoires, le DOS n'étant plus capable de les retrouver. Si vous tenez donc absolument à vous livrer à des tests avec les fonctions BIOS, il est plus prudent d'effectuer auparavant une sauvegarde complète du disque dur. Vous pouvez également effectuer ces tests à partir d'un ordinateur que vous n'utilisez plus. C'est là le seul moyen de réparer par la suite une éventuelle perte de données due à des tests intempestifs.

Les interruptions disque dur du BIOS

Comme nous l'avons déjà évoqué lors de l'étude des interruptions disquettes du BIOS, disques durs et disquettes se partagent l'interruption 13h. Les différentes fonctions qui vous sont offertes pour le disque dur et pour le lecteur de disquettes effectuent les mêmes tâches mais le disque dur est géré par le BIOS de manière différente des lecteurs de disquettes. C'est pourquoi le BIOS contient un module distinct pour la gestion du disque dur et un autre pour la gestion des lecteurs de disquettes.

Lorsque l'interruption 13h est appelée, la routine de disque dur est tout d'abord appelée. Celle-ci teste alors, d'après le numéro périphérique dans le registre DL, si c'est le disque dur ou le lecteur de disquettes qui va être appelé. La valeur 80h représente le premier disque dur, la valeur 81h le deuxième. Il n'est pas possible de gérer par le BIOS plus de deux lecteurs de disque dur.

Les fonctions disques durs du BIOS existent depuis l'introduction des PC/XT. Elles n'existaient pas sur les PC précédents. De fait, elles n'auraient pas servi à grand chose car, en 1981, il était encore impensable d'équiper des micro-ordinateurs de disques durs. Avec le temps, et plus précisément depuis l'introduction des PC/XT et du modèle PC/2 de IBM, de nouvelles fonctions sont venues s'ajouter, tel que nous le montre le tableau suivant.

Fonction	Tâche	Depuis...
00h	Réinitialisation	XT
01h	Lecture de l'état	XT
02h	Lecture	XT
03h	Ecriture	XT
04h	Vérification	XT
05h	Formatage	XT
08h	Détection du format	XT
09h	Adaptation de lecteurs étrangers	XT
0Ah	Lecture étendue	XT
0Bh	Ecriture étendue	XT
0Ch	Positionnement des têtes lecture/écriture	XT
0Dh	Réinitialisation	XT
0Eh	Test contrôleur lecture	XT
0Fh	Test contrôleur écriture	seulement PS/2
10h	Teste si l'unité est prête	seulement PS/2
11h	Recalibre une unité	XT
12h	Test contrôleur RAM	XT
13h	Test du lecteur	seulement PS/2
14h	Diagnostique interne du contrôleur	seulement PS/2
15h	Lecture du type de disque	XT
		AT

Sachant que l'accès au disque dur avec le BIOS ne joue aucun rôle dans le travail de programmation quotidien, nous nous limiterons dans ce chapitre à la description des fonctions essentielles. Vous trouverez de plus amples informations dans l'annexe de cet ouvrage dans laquelle l'ensemble des fonctions est décrit ainsi que leurs paramètres d'entrée et de sortie.

Le code d'état

Toute les fonctions disque dur ont ceci en commun qu'elles communiquent, après le lancement de la fonction par le biais du flag-carry, si leur tâche a été correctement exécutée ou si une erreur est apparue au cours de l'exécution. Si tel est le cas, le flag-carry est positionné et un code d'erreur s'inscrit dans le registre AH.

Les différents codes ont la signification suivante :

Codes d'erreur avec l'interruption disque du BIOS 13h pour l'accès à l'unité du disque dur	
Code	**Signification**
00h	Pas d'erreur
01h	Numéro de fonction ou unité non autorisés
02h	Marque d'adresse non trouvée
04h	Secteur appelé non trouvé
05h	Erreur lors de la réinitialisation du lecteur
07h	Erreur lors de la réinitialisation du contrôleur
09h	Transmission de données au-delà de la limite de segment
0Ah	Secteur défectueux
10h	Erreur de lecture
11h	Erreur de lecture corrigée par ECC
20h	Erreur du contrôleur de disquette
40h	Piste non trouvée
80h	Erreur de Time-Out, unité de disque ne réagit pas
AAh	Unité de disque n'est pas prête
CCh	Erreur d'écriture

De manière générale, chaque fois qu'une de ces erreurs (à l'exception de l'erreur 1) se produit, nous vous conseillons de faire exécuter tout d'abord une réinitialisation et de tenter un nouvel appel de la fonction concernée. Dans la plupart des cas, aucune erreur ne sera annoncée à la suite de cette seconde tentative.

Si l'erreur 11h apparaît lors de l'appel d'une fonction de lecture, cela ne signifie pas nécessairement que les données chargées soient erronées. Cette erreur signale en effet qu'une erreur de lecture a bien été détectée mais qu'elle a pu être corrigée à l'aide de l'algorithme ECC ("Error Correction Code"). Cette méthode ressemble à la méthode CRC utilisée sur les lecteurs de disquettes. Les différents octets d'un secteur sont combinés à travers une formule mathématique complexe. Le résultat de ces opérations est stocké sur le disque dur sous forme de 4 octets qui viennent s'ajouter à chaque secteur. Lorsqu'une erreur en lecture est constatée, elle peut ainsi être corrigée dans certains cas à l'aide du résultat ECC qui a été sauvegardé.

Lancement des fonctions disques

Un autre point commun entre les différentes fonctions constitue l'emploi des registres du processeur pour la transmission des données. Si le numéro de lecteur de disque dur à appeler est requis, il doit toujours être transmis dans le registre DL. La valeur 80h représentera toujours le premier lecteur de disque dur, 81h le second. Le numéro de la tête de lecture/écriture (qui fournit indirectement le numéro du disque appelé puisque chaque disque dispose de deux têtes de lecture/écriture) est transmis dans le registre DH.

Le numéro de cylindre doit être inscrit dans le registre CHC. Par définition, ce registre 8 bits ne permet d'appeler que 256 cylindres alors que le disque dur du PC/XT dispose déjà de 306 cylindres. Ce registre ne peut donc suffire, à lui seul, à indiquer le numéro de cylindre.

C'est pourquoi les bits 6 et 7 du registre CL sont également employés pour indiquer le numéro de cylindre. Il constitue les bits 8 et 9 du numéro de cylindre, de sorte que 1024 cylindres (numérotés de 0 à 1023) peuvent être appelés au total. Les bits 0 à 5 ne restent cependant pas inemployés pour autant car ils fournissent le numéro du secteur appelé (ils sont numérotés de 1 à 17 par secteur). Lorsqu'il s'agit d'accéder simultanément à plusieurs secteurs, le registre AL reçoit le nombre de ces secteurs. Pour les opérations de lecture et d'écriture, il faut naturellement également indiquer l'adresse d'un tampon dont les données devront être tirées ou bien dans lequel elles devront être transférées. Dans ce cas, le registre ES reçoit l'adresse de segment et le registre BX l'adresse d'offset de ce tampon.

Réinitialisation du contrôleur de disque dur

La fonction 00h qui, tout comme la fonction 0Dh, déclenche une réinitialisation du contrôleur, n'a toutefois aucunement besoin de toutes ces indications. Le recours à cette fonction est indispensable chaque fois qu'une erreur est apparue de façon à ce qu'une réinitialisation soit effectuée avant le prochain accès aux données. Comme toujours, c'est dans le registre DL que vous indiquez le lecteur de disque dur sur lequel la réinitialisation doit s'appliquer.

Communication de l'état du disque dur

La fonction 01h permet de tester l'état du lecteur de disque dur qui peut vous être communiqué après chaque opération. Ici également, le numéro du lecteur dont il s'agit de connaître l'état doit être inscrit dans le registre DL.

Lecture des secteurs de disques durs

La fonction 02h permet de lire 1 ou plusieurs secteurs. Un maximum de 128 secteurs peuvent être lus au cours d'un seul appel de cette fonction. Cette limitation est due au fait que les données sont transférées directement dans la mémoire RAM par le contrôleur de disque dur à l'aide d'un composant DMA, et donc sans passer par le processeur. Or, le composant DMA ne peut recevoir que 64 Ko à la fois. D'autre part, il ne peut transférer ces 64 Ko qu'à l'intérieur d'un seul segment de mémoire. Il faut donc que le tampon dont l'adresse est transmise dans ES:BX puisse tenir en totalité dans les 64 Ko qui commencent à partir de l'adresse de segment indiquée en ES. Si tel n'est pas le cas, le composant DMA annoncera une erreur.

Si vous souhaitez lire plusieurs secteurs avec cette fonction, il est également intéressant de savoir quels secteurs seront lus par la fonction. Il s'agira bien sûr en premier lieu des secteurs de numéros croissants à l'intérieur du cylindre spécifié, sur la tête spécifiée. Mais une fois que le dernier secteur d'un cylindre a été lu, si d'autres secteurs doivent être lus, la lecture ne se poursuivra pas sur le premier secteur du prochain cylindre sur la tête spécifiée mais sur le premier secteur du même cylindre sur la tête suivante. Ce n'est que si la dernière tête est atteinte de cette façon et s'il reste encore d'autres secteurs à lire que l'opération de lecture passera au premier secteur du cylindre suivant sur la première tête.

Vérification des secteurs de disques durs

La fonction 04h permet de vérifier les différents secteurs d'un cylindre. Les données figurant sur le disque dur ne sont toutefois pas comparées avec les données en mémoire (il n'est donc pas nécessaire de transmettre une adresse de tampon dans ES:BX). Le système utilise simplement le nombre ECC qui a été stocké avec les données pour vérifier si les octets stockés, une fois soumis à l'algorithme ECC, donnent bien à nouveau le résultat qui a été sauvegardé. Le nombre de secteurs à vérifier doit ici également être indiqué dans le registre AL.

Formatage des cylindres des disques durs

Avant d'accéder à un disque dur, il faut naturellement le formater. C'est la fonction 05h qui se charge de cette opération. Comme pour la fonction de formatage d'une disquette, cette fonction attend les numéros de têtes et de cylindres ainsi que l'adresse d'un tampon dans la paire de registres ES:BX. Ce tampon doit avoir la taille de 512 octets même si seuls les 34 premiers octets sont utilisés. Il contient deux octets pour chacun des 17 secteurs à formater. Le premier octet indique si le secteur est en bon état. Comme il n'y a pas de raison de supposer qu'un secteur ne soit pas en bon état avant appel de la fonction, on inscrira chaque fois un zéro dans cet octet. Le second octet reçoit le numéro (logique) qui devra être attribué au secteur correspondant. Le BIOS tirera des deux

premiers octets les informations correspondant au premier secteur physique du cylindre, des octets trois et quatre les informations correspondant au second secteur physique, et ainsi de suite. Remarquez bien que l'ordre physique des secteurs est fixé d'avance alors que l'ordre logique des mêmes secteurs peut être librement défini à l'aide des deux octets d'information de cette table qui correspondent à chaque secteur.

A première vue, il semble que le plus simple serait de faire coïncider les numéros de secteurs logiques et physiques. Cela présente toutefois certains inconvénients pratiques. Si l'on veut obtenir un accès optimal et aussi rapide que possible, il est en réalité préférable (surtout lorsqu'il s'agit de lire plusieurs secteurs consécutifs) de décaler la répartition des secteurs logiques par rapport aux secteurs physiques. Cette méthode est appelée "Sector-Interleaving" en anglais. Nous étudierons ce phénomène d'entrelacement dans la section 6.7.1.

Au cours de l'appel de fonction, le BIOS inscrit dans le premier octet de la marque du secteur dans la table une valeur qui indiquera au programme d'appel si le secteur est en bon état ou bien s'il est préférable de ne pas y écrire du fait d'une magnétisation incorrecte. La valeur 0 signifie ici "en bon état" et 128 "défectueux". Outre les registres dont le rôle a été signalé plus haut, le registre AL reçoit, pour cette fonction, le nombre de secteurs à traiter. Sachant que, à la différence des disquettes, il n'existe pas de format unique pour les disques durs, il peut être très important dans certaines situations qu'un programme puisse être à même de déterminer les caractéristiques du lecteur du disque dur connecté. Il peut utiliser à cet effet la fonction 08h qui attend, lorsqu'elle est appelée, le numéro de fonction ainsi que le numéro du lecteur de disque dur dans le registre DL.

Après appel de la fonction, le registre DL contient tout d'abord le nombre de disques durs connectés. Les valeurs 0, 1 et 2 sont possibles ici. Le registre DH contient en outre le nombre de têtes de lecture/écriture. Comme les têtes sont toujours numérotées en partant de 0, la valeur 7, par exemple, signifie donc ici qu'il y a 8 têtes. Le nombre de cylindre, comme pour les autres fonctions, est indiqué dans le registre CL (bits 0 à 7 du nombre de cylindres) en liaison avec le registre CH (bits 8 et 9 du nombre de cylindres). On compte ici également en partant de 0. La dernière information est fournie par les 6 bits de plus faibles poids du registre CH. Il s'agit du nombre de secteurs par cylindre. Exceptionnellement, ce nombre s'entend en comptant à partir de 1.

Connexion de disques durs étrangers

Chaque BIOS contient les caractéristiques d'un nombre plus ou moins élevé de disques durs qui peuvent être paramétrés à partir du Setup.

Pour connecter un disque dur étranger à l'aide du BIOS, il faut naturellement lui indiquer les caractéristiques de ce disque dur par un autre moyen. A cet effet, il utilise l'interruption 41h pour le lecteur de disque dur 0 et l'interruption 46h pour le lecteur de disque dur 1. Cette interruption fait office de pointeur sur une table dont le format est fixé par le BIOS et qui décrit le disque dur connecté. Habituellement, ces deux

vecteurs d'interruption font appel à un pointeur sur une table stockée dans le BIOS. Il est toutefois possible d'utiliser d'autres tables inconnues du BIOS en les paramétrant à l'aide d'un logiciel.

L'installation d'une telle table et l'initialisation du vecteur d'interruption correspondant ne suffisent toutefois pas. Vous devez naturellement donner au BIOS l'ordre de se régler sur cette table. A cet effet, vous utilisez la fonction 09h qui attend simplement le numéro de fonction ainsi que le numéro du lecteur à adapter (80h ou 81h) dans le registre DL. Dans la plupart des cas, vous pourrez toutefois vous dispenser de ce travail complexe car les fabricants de ces disques durs exotiques fournissent généralement un petit programme de lancement qui vous décharge de ce travail, ou bien ils vous indiquent dans la documentation du programme les paramètres appropriés pour la table de descripteurs de disques durs.

Lecture étendue des secteurs du disque dur

Les fonctions 0AH et OBH constituent deux autres fonctions de lecture et écriture. Elles ne se distinguent des fonctions 02h et 03h que par le fait qu'elles lisent ou sauvegardent non seulement les 512 octets des données d'un secteur mais aussi les 4 octets ECC qui figurent à la fin de chaque secteur. La taille effective de chaque secteur passe donc de 512 octets à 516 et ces fonctions ne permettent donc de lire ou écrire à la fois que 127 secteurs au lieu de 128 pour les fonctions 02h et 03h.

La fonction 10h vous permet de tester si le lecteur de disque dur, dont le numéro figure dans le registre DL, est actuellement prêt ou non à recevoir des instructions. Cette fonction fournit la réponse à cette question à travers le Flag-Carry. Si celui-ci est positionné, cela signifie que le lecteur n'est pas prêt et l'un des codes d'erreurs habituel figure dans le registre AH.

Recalibrer un disque dur

Pour recalibrer un des deux lecteurs de disque dur, vous pouvez utiliser la fonction 0DH. Elle renvoit également l'état d'erreurs suivant les règles habituelles après avoir été appelée avec le numéro de lecteur dans le registre DL.

Auto-test du contrôleur de disque dur

Pour faire exécuter un auto-test par le contrôleur, vous pouvez appeler la fonction 14h. Si le contrôleur est en bon état de marche, cette fonction renvoit un Flag-Carry annulé.

La dernière fonction de disque dur que vous puissiez appeler à travers l'interruption 13h est la fonction 15h qui n'est cependant disponible que sur les AT mais pas sur les XT.

Elle sert à déterminer le type d'un lecteur déterminé. Elle attend également le numéro du lecteur (80h ou 81h) dans le registre DL. Si le lecteur souhaité n'est pas présent, elle renvoit, après appel, la valeur 0 dans le registre AH. Si ce registre contient par contre la valeur 1 ou 2, c'est que le périphérique spécifié est un lecteur de disquette. La valeur 3 indique enfin que c'est un disque dur qui est connecté. Dans ce cas, les registres CX et DX contiennent le nombre de secteurs sur ce disque dur. Les deux registres forment à cet effet un nombre sur 32 bits, le registre CX représentant les 16 bits de plus fort poids et le registre DX les 16 bits de plus faible poids.

6.5. Les disques durs et leurs contrôleurs

Les progrès accomplis dans la technologie des disques durs se résument en quatre noms qui décrivent les différentes catégories de contrôleurs utilisés dans l'univers PC : ST506, ESDI, SCSI et IDE. Du type de contrôleur ne dépend pas uniquement le format dans lequel les données seront stockées sur le disque dur, mais également le débit de transfert de données entre l'ordinateur et le disque dur. En clair, le contrôleur est responsable de la capacité du disque dur et de la vitesse d'exécution. Un disque dur rapide ne sert pas à grand chose si le contrôleur actif ne suit pas.

La table ci-après nous montre les débits de transfert de données maximum que les différents contrôleurs permettent d'atteindre. Précisons toutefois qu'il s'agit de valeurs maximales théoriques rarement atteintes car, dans la pratique, de nombreux facteurs viennent interférer. En effet, le transit des informations du disque dur à l'écran (par exemple, pour charger un document texte) sera perturbé par l'intervention du contrôleur, du BIOS, du DOS, du programme d'application concerné et, éventuellement, par certains programmes résidents.

Le BUS est le premier à limiter la vitesse des contrôleurs de disque dur car il fonctionne à une cadence de 8 MHz alors que le CPU avoisine les 50 MHz. Les valeurs indiquées se réduisent donc au cours du chargement concret des fichiers d'au moins un cinquième.

Débits de transfert de données maximums des différents contrôleurs de disque dur

Contrôleur	Débit de transfert de données maximum
ST506	1 Mo par seconde
ESDI	2,5 Mo par seconde
IDE	4 Mo par seconde
SCSI	5 Mo par seconde

Nous allons maintenant vous présenter les différents contrôleurs de disque dur, examiner leur structure et mettre en évidence les avantages et les inconvénients qu'ils présentent.

Nous évoquerons également le rôle joué par le BIOS dans la programmation de ces contrôleurs.

6.5.1. Le contrôleur ST506

Les premiers disques durs qui eurent du succès dans l'univers PC furent développés en vue d'une collaboration avec un contrôleur ST506 ou compatible. Comme son nom le laisse supposer, ce contrôleur standard a été mis au point par la société Seagate qui joue, aujourd'hui encore, un rôle important dans le monde PC.

A l'heure actuelle, les contrôleurs ST506 sont encore très répandus bien que les disques durs récents soient majoritairement équipés de contrôleurs IDE. En règle générale, on reconnaît un disque dur conçu pour un contrôleur ST506 aux références MFM/RLL. Celles-ci représentent les deux procédures utilisées par le contrôleur pour le stockage des données sur le disque dur. On peut paramétrer le format choisi par le contrôleur à l'aide des switches DIP ; on choisira alors de préférence la procédure RLL qui autorise une capacité plus élevée du disque dur (voir section 6.6).

En raison de sa forte présence sur le marché, le contrôleur ST506 a déterminé différents standards concernant la gestion-machine ; la compatibilité exclusive du BIOS sur ce type de contrôleur y a par ailleurs fortement contribué. Cet état de fait a eu des conséquences ressenties aujourd'hui encore, avec notamment la quasi-obligation pour les contrôleurs IDE et ESDI d'être, sur de nombreux points, compatibles avec le contrôleur ST506. Nous aborderons ce point un peu plus tard.

Le matériel

Le disque dur et le contrôleur constituent deux éléments distincts pour le standard ST506. Le contrôleur est en effet placé sur une platine séparée et enfiché sous la forme d'une carte d'extension dans l'un des slots d'extension.

Un contrôleur de ce type est en général capable de gérer deux disques durs sachant que, la plupart du temps, des "multi-contrôleurs" sont alors utilisés et permettent de plus de gérer deux lecteurs de disquettes. Le contrôleur est relié au disque dur par deux câbles : un câble de données à 20 broches et un câble de commande à 34 broches.

A l'inverse du câble de données, qui existe séparément pour chacun des disques durs, les deux disques se partagent le câble de commande lequel possède, à cet effet, deux extrémités. C'est par le biais du câble de commande que sont pilotées les têtes de lecture/écriture et qu'est lancée la lecture/écriture des secteurs. Les données qui doivent être lues ou écrites sont échangées entre le contrôleur et le disque dur par le câble de données tout comme les informations de pilotage stockées avec chaque secteur sur le disque. Le transfert ne s'effectue cependant pas en mode binaire mais en mode sériel

et analogique, et c'est au contrôleur d'effectuer ensuite la conversion des flux de données en chaînes de bits.

Le débit de transfert de données est toujours de l'ordre de 5 MBit/s si le disque a été formaté selon la procédure d'encodage MFM et de 7,5 MBit/s avec la procédure RLL. Sachant que les informations de pilotage doivent encore être filtrées de ce flux de données, le débit de transfert pur est sensiblement inférieur. Mais la masse données traitées en l'espace d'une seconde au cours de la procédure MFM reste de l'ordre d'un demi Mo, la procédure RLL atteint quant à elle 0,75 Mo. A l'instar de la plupart des informations ayant trait au disque dur, ces valeurs restent somme toute très théoriques ; de fait, elles ne tiennent pas compte du temps de déplacement entre les cylindres et les pistes et partent de plus du principe que la lecture s'effectue secteur après secteur. Ce qui n'est pas toujours le cas, ce que nous verrons dans le chapitre consacré au facteur d'entrelacement.

Le débit de transfert de données, plus élevé avec la procédure d'encodage RLL, provient de la plus grande efficacité du mécanisme d'inscription qui se mesure au nombre de secteurs inscrit sur chaque piste. Il ne sont qu'au nombre de 17 pour la procédure MFM et dix de plus pour la procédure RLL, soit 26. La vitesse de rotation du disque est de 3600 Rotation/Mn pour les deux formats.

A l'époque où IBM a sorti le PC/XT, seuls les contrôleurs ST506 étaient disponibles. Les nouvelles fonctions disque dur du BIOS en ROM furent donc axées sur ce type de contrôleur sans tenir compte de la limitation de ses performances. C'est ainsi que le nombre de lecteurs fut limité à deux, le nombre maximum de cylindres à 1204, le nombre de secteurs par piste à 63 et celui des têtes à 16. De plus, la taille des secteurs fut fixée à 512 octets, ce qui donne, en liaison avec les autres facteurs, une capacité maximale de 504 Mo. Pas plus.

Le fait qu'il existe depuis longtemps des disques durs d'une plus grande capacité s'explique tout simplement par le fait que ceux-ci simulent au système la présence de deux disques correspondant en fait à un seul. On arrive ainsi à une capacité de disque dur pouvant atteindre 1Go bien que ces disques ne soient alors plus utilisables connectés à un contrôleur ST506. En effet, de tels disques requièrent une plus grande rapidité d'accès impossible à atteindre avec un contrôleur ST506.

Ce type de contrôleur sera donc amené à disparaître de plus en plus du marché dans les années à venir.

6.5.2. Le contrôleur ESDI

Après le contrôleur ST506 est arrivé tout d'abord le contrôleur ESDI ("Enhanced Small Devices Interface") dont IBM a notamment équipé ses PS/2. Il s'agit de manière globale d'une évolution du contrôleur ST506 ; par exemple, l'interface de programmation n'a pas été modifiée. De ce fait, les disques ESDI peuvent être utilisés sans problèmes sur

des ordinateurs dont le BIOS aura été programmé pour ne supporter que le contrôleur ST506.

A la différence du ST506, les flux de données ne sont pas transmis directement au contrôleur par le biais du câble de données dans le cas du contrôleur ESDI. Au-lieu de cela, une partie de la logique de décodage, appelée séparateur de données, est d'emblée stockée sur le disque dur. Ce séparateur prépare les informations lues et ne transmet que les données pures sous forme digitale au contrôleur proprement dit.

Vu que le contrôleur et le séparateur données travaillent en parallèle, le débit de transfert peut être augmenté à 10 MBit/s. Il ne s'agit que du double par rapport à un disque MFM-ST506, mais un avantage décisif s'y ajoute. Les contrôleurs ESDI sont, à la différence des contrôleurs ST506, équipés d'une mémoire tampon pour les secteurs qui permet un facteur d'entrelacement de 1:1. Alors qu'un disque ST506 requiert un facteur d'entrelacement de 1:3, voire de 1:6, soit au moins trois, voire 6 rotations pour effectuer la lecture d'une piste entière, le disque ESDI effectue cette lecture en une seule rotation. La vitesse d'accès s'en trouve donc augmentée en plus d'un facteur de 3 à 6.

En outre, le standard ESDI prévoit, en plus du débit de transfert de 10 MBit/s communément utilisé, des débits de transfert de 15, 20 voire 24 MBit/s. Les contrôleur correspondants à ces taux sont toutefois rares et, en général, onéreux, si bien que la plupart des contrôleurs utilisent un débit de transfert de 10 MBit/s.

Par ailleurs, le séparateur de données n'est pas le seul élément d'"intelligence" dont est équipé un disque ESDI performant à la différence d'un disque ST506. En effet, un disque ESDI enregistre des informations concernant son format physique ainsi que l'adresse des secteurs défectueux et transmet sur demande ces informations au contrôleur. Celui-ci est ensuite à même de se paramétrer en fonction du disque dur connecté, une procédure qui, dans le cas de disques ST506, exige de la part de l'utilisateur qu'il aille sélectionner un format de données correspondant dans le programme de configuration du BIOS.

Venons-en au BIOS

Cette information, stockée dans la mémoire CMOS de l'horloge alimentée par pile d'un AT, nécessite toutefois un ordinateur équipé d'un contrôleur ESDI. Le BIOS doit en effet impérativement connaître les caractéristiques du disque dur et les transmettre ensuite au pilote de périphérique du DOS.

Vu que le contrôleur ESDI peut demander les caractéristiques du disque dur, les informations du BIOS ne doivent pas nécessairement coïncider avec le format effectif du disque. Il arrive même souvent que ce décalage ne puisse être évité car chaque BIOS ne connaît qu'un nombre déterminé de disques durs avec leurs caractéristiques.

Le problème qui se pose avec les contrôleurs ST506 est épineux lorsque le disque dont est équipé la machine ne se trouve pas dans la liste. Il est alors nécessaire de choisir un

disque dont le format se rapproche le plus possible du disque concerné, ce qui signifie souvent : renoncer à certaines pistes, voire certaines têtes, vu qu'on ne peut choisir qu'un format plus petit, jamais un format plus grand. En effet, le BIOS irait alors chercher des secteurs qui n'existent pas, provoquant ainsi immanquablement des erreurs.

La situation devient particulièrement périlleuse lorsque le BIOS ne propose aucun type de disque dur comportant le même nombre de secteurs par piste. Il faudra alors choisir une taille de secteur plus petite avec, à la clef, la perte de nombreux secteurs.

Ceci n'est pas à craindre avec les disques ST506 car il fonctionnent systématiquement avec 17 ou 26 secteurs. Il en va cependant différemment avec les disques ESDI. Les formats courants ont en effet ici généralement 34 ou 36 secteurs par piste. Si le BIOS ne connaît que des disques avec 26 secteurs, on est alors contraint de sacrifier sur l'autel du gaspillage un tiers de la capacité du disque dur.

Mais ceci n'a pas lieu d'être car le contrôleur peut interroger les caractéristiques du disque et les mettre en relation avec les valeurs enregistrées dans le BIOS. Celles-ci lui sont automatiquement communiquées lors de son initialisation par le BIOS. Le contrôleur est alors à même d'adapter le format transmis par le BIOS au format réel ; remarquez toutefois que les adresses des secteurs devront être recalculées lors de chaque accès. Cela prend un peu de temps mais permet par ailleurs d'utiliser le disque dur avec pratiquement n'importe quel format BIOS tant que celui-ci n'envisage pas de capacité globale supérieure.

Les capacités à effectuer cette adaptation dépendent entièrement du contrôleur ESDI concerné. Certains ne supportent que certains formats particuliers alors que d'autres se montrent tout à fait flexibles et sont de ce fait capables de reprendre n'importe quel format. Ceci est particulièrement intéressant lorsque des disques ESDI atteignent les limites du BIOS, par exemple les disques comportant plus de 1024 cylindres. Au-lieu d'abandonner les cylindres supplémentaires, ces disques simulent un nombre de secteurs par piste supérieur à la réalité car la limite imposée par le BIOS, à savoir un nombre maximum de 63 secteurs par piste, permet une certaine marge de manoeuvre.

6.5.3. SCSI

Le standard SCSI ne représente pas exactement une interface disque dur mais plutôt une possibilité de connecter jusqu'à 8 périphériques différents, avec notamment, hormis les disques durs, les streamers, les lecteurs de CD-ROM ou les scanneurs conçus pour cette interface.

A la différence des standards de contrôleurs que nous avons vus jusqu'à présent, le "Small Computer System Interface" ne se rencontre pas uniquement sur les PC mais également sur de nombreux systèmes 68000 (Mac, Atari St) et stations de travail. Son attrait consiste dans le découplage des périphériques connectés et du système informatique proprement dit car les périphériques communiquent avec le contrôleur par un système de bus qui leur est propre et qui est distinct du bus PC.

Vu que non seulement les fiches de connexion, mais également les commandes de ce bus SCSI envoyées aux périphériques sont normalisées, les équipements SCSI peuvent être échangés à volonté entre les différents systèmes et seule le contrôleur SCSI doit être paramétré en fonction du système serveur. La réalité est toutefois moins souriante en matière de compatibilité, notamment avec les contrôleurs SCSI bon marché disponibles aujourd'hui pour quelques centaines de francs. Les contrôleurs SCSI de Adaptec constituent le standard de référence, mais Western Digital et Future Domain jouent également dans la cour des grands sur ce segment de marché.

Le bus système qui relie entre eux les différents périphériques et qui se connecte au contrôleur SCSI par une fiche 80 broches, est organisé en parallèle et permet de transférer simultanément 8 bits. La nouvelle version du standard SCSI, SCSI II, autorise des bus à 16 bits mais n'a pas cependant, jusqu'à aujourd'hui, trouvé écho dans le monde des PC.

Les distributeurs de contrôleurs SCSI attirent volontiers leurs clients en agitant l'argument des débits de transfert de données de 4 ou 5 Mo/s qui sonnent comme une promesse aux oreilles des utilisateurs DOS. Mais cette promesse ne tarde pas à apparaître sans fondement au contact de la réalité. Si le contrôleur SCSI est éventuellement capable d'atteindre ce débit, il en va tout autrement du disque dur connecté. Il faut objectivement compter sur débit allant de 1,5 à 2 Mo/s, à moins d'investir dans un contrôleur EISA onéreux qui, grâce aux avantages de son BUS, parvient à atteindre un débit de l'ordre de 2,5 Mo/s.

Les disques SCSI s'inscrivent dans la tendance observée avec les disques ESDI et intègrent une grande partie de l'électronique de pilotage directement sur le disque dur. Il ne doit pas en être autrement avec ce système car le contrôleur doit être indépendant des périphériques et ne peut pas se permettre d'être perturbé par les particularités d'un disque dur.

La grande proximité du disque dur et de son électronique de pilotage offre certains avantages en plus du fait que les distances à parcourir sont plus courtes. Il devient en effet également plus aisé de simuler au contrôleur proprement dit un format de disque particulier qui n'existe pas. Ce n'est effectivement pas en vain que le contrôleur SCSI demande, lors du lancement du système, et de la même manière qu'un contrôleur ESDI, le format des périphériques connectés et transmet ensuite ces informations au système.

Ils sont en cela aidé par le BIOS intégré car les systèmes SCSI dépendent de leur propre BIOS parce qu'ils ne supportent pas, du côté des logiciels, le standard ST506 que le BIOS en ROM suppose à priori. Les fonctions de l'interruption disque initiale du BIOS sont alors remplacées par le BIOS SCSI. Un tel BIOS est toutefois inefficace en mode protégé où certains "systèmes d'exploitation", tels Novel ou OS/2, doivent pouvoir accéder directement au disque et ne supportent, en général, que le standard ST506. Il faut alors faire appel aux pilotes de périphériques correspondants, mais ceux-ci sont justement difficile à trouver. Il s'agit très certainement de la plus grande faiblesse des interfaces SCSI.

En revanche, si l'on dispose du pilote adéquat, l'augmentation de la capacité du disque ne pose plus aucun problème. Il suffit alors d'installer un disque supplémentaire et de le relier au BUS SCSI. Le contrôleur SCSI se charge alors du reste en liaison avec le pilote correspondant.

6.5.4. IDE

Le "must" des contrôleurs de disque dur est sans aucun doute l'interface IDE (Intelligent Drive Electronics) dont est équipée la quasi totalité des nouveaux PC. La mise au point de ce standard remonte à 1984, lorsque le fabricant de micro-ordinateurs Compaq chargea Western Digital, le spécialiste du disque dur, de mettre au point un contrôleur compatible ST506 qui, pour des raisons de place, devait se trouver avec l'électronique de pilotage directement sur le disque dur.

Il en va aujourd'hui encore de même, et l'étroite association du contrôleur au lecteur reste la caractéristique principale des disques IDE. Lorsqu'on achète un disque IDE, on acquiert en même temps le contrôleur. C'est d'ailleurs pour cette raison qu'il n'existe pas pour les contrôleurs IDE de câbles de commande et de données distincts mais uniquement un câble à 40 broches directement relié au BUS système.

C'est pour cette raison que les disques IDE doivent leur surnom ; on les appelle en effet souvent les "Disques BUS AT". Cela ne signifie toutefois nullement que ces systèmes se limitent exclusivement aux AT avec leurs BUS de données étendus à 16 bits. Car il existe de la même manière des disques BUS XT conçus pour les BUS XT 8 bits. Mais cela est peu connu en raison de la disparition progressive du marché des XT.

De nombreux PC récents hébergent directement dans leur unité centrale la connexion pour les câbles BUS IDE. Les disques IDE sont toutefois livrés le plus souvent avec une petite carte d'extension que l'on enfiche dans l'un des slots d'extension et qui est essentiellement destinée à établir la liaison entre le BUS système et le câble IDE.

En raison de l'étroite liaison qui existe entre le disque et le contrôleur, IDE jouit de certaines qualités également rencontrées sur les contrôleurs SCSI. On y trouve notamment la capacité à émuler n'importe quel format, le "Multiple-Zone-Recording" qui permet de gagner de la place sur le disque dur ou encore le cache-pistes qui permet la lecture de pistes individuelles et de les stocker ensuite dans une mémoire cache interne. Les disques IDE fonctionnent ainsi systématiquement avec un facteur d'entrelacement de 1:1, ce qui permet des accès rapides et contribue à la bonne réputation des disques IDE par rapport aux disques ESDI en matière de performance. Le standard IDE réunit les qualités des trois autres standards de contrôleur : il est flexible comme le SCSI, rapide comme le ESDI et est connectable sans problème à la plupart des systèmes PC grâce à sa compatibilité avec le standard ST506.

Les disques IDE se distinguent en outre des autres contrôleurs de disques durs par une très faible consommation de courant, ce qui les prédestinent à équiper les Laptops, Notebooks et autres portables autonomes.

Concernant l'utilisation de tels machines, de nombreux disques IDE reconnaissent certaines commandes spéciales qui les met dans un état de demi sommeil, réduisant encore davantage la consommation de courant. En règle général, ces commandes ne peuvent être utilisées qu'avec des pilotes spéciaux ou avec un BIOS paramétré pour la mise en oeuvre de disques IDE. Car, du point de vue du BIOS, les disques IDE se comportent de la même manière que des contrôleurs ST506 normaux, ce qui rend aisée l'intégration des ces disques dans des systèmes existants.

Le développement d'un standard pour les disques IDE est actuellement en cours, laissant entrevoir à l'avenir un soutien direct des disques IDE par de nombreux systèmes BIOS. Ceci aura pour effet de permettre une meilleure exploitation des possibilités étendues de ces disques qui sont encore actuellement inutilisées.

6.5.5. Du contrôleur à la mémoire

Indépendamment de la vitesse du disque dur et du contrôleur, le mode de transfert des données du contrôleur vers le disque dur détermine la vitesse réelle du tandem contrôleur-disque dur. Il existe quatre procédés :

- ✓ Programmed I/O (PIO)
- ✓ Memory Mapped I/O
- ✓ DMA
- ✓ Busmaster DMA.

Programmed I/O

Dans le cas du Programmed I/O, le transfert de données entre le contrôleur et la mémoire centrale s'effectue par les différents ports I/O du contrôleur qui servent également au transfert de commandes. Du côté logiciel, le programme concerné a recours au commandes en langage machine IN et OUT. Cela signifie que chaque octet ou mot devra être envoyé par le CPU.

Le débit de transfert de données n'est pas seulement limité à cause des limites imposées par le BUS PC mais également par les performances du CPU. Le BUS ISA permet certes un débit de transfert maximum de 5,33 Mo/s lors d'un accès, ce débit ne peut cependant jamais être atteint complètement, même avec un CPU rapide. Les micro-ordinateurs de type 386 et 486 sont toutefois capables d'utiliser efficacement le contrôleur car ils appellent les données plus rapidement que le contrôleur ne les fournit. On peut même atteindre 3 ou 4 Mo/s avec des disques durs extrêmement rapides...et chers.

Memory-Mapped I/O

Le CPU peut réceptionner les données du contrôleur d'autant plus rapidement si elles sont stockées dans une zone fixe de la mémoire ; il s'agira, en général, d'un segment situé au dessus de la mémoire vidéo. Le transfert des données dans une zone de la mémoire utilisée par le programme qui est à l'origine de la demande, est alors possible à l'aide des commandes MOV qui sont plus rapides que les accès au port par IN et OUT.

Cette technique permet également, avec des CPU récentes, d'appeler les données plus rapidement que le contrôleur ne les fournit. En effet, le contrôleur n'atteint jamais la vitesse maximale théorique de 8 Mo/s, et même les valeurs, pourtant réalistes, de 5 à 6 Mo/s ne sont pas ateintes.

DMA

Le transfert en mode DMA est encore plus connu que les deux procédés précédemment cités. Il est supporté par un processeur DMA qui lui est propre. Il est conçu pour permettre un transfert de données direct d'un périphérique (disque dur, disquette, CD-ROM, etc.) vers la mémoire, éliminant ainsi le détour par le CPU (DMA: Direct Memory Access). L'idée est certes bonne, car il suffit au programme de communiquer au contrôleur DMA le nombre d'octets qui doivent être transférés ainsi que l'endroit où il sont stockés et leur destination. Mais la concrétisation de cette idée sur PC se révèle imparfaite.

En effet, le contrôleur DMA utilisé sur PC est relativement peu flexible (ce qu'on pourrait d'ailleurs lui pardonner), mais il est surtout lent. Tellement lent que les accès sur les 386 et 486 selon la procédure I/O est plus rapide. Il s'agit là d'un anachronisme de l'histoire du PC : le contrôleur DMA est cadencé à 4 MHz sur les PC/AT et ses successeurs car le PC/XT était cadencé à 4,77 MHz. Le transfert DMA y était donc plus rapide que Programmed I/O, mais uniquement dans ce cas. Il est impossible de dépasser pour cette raison les 2 Mo/s.

La plupart des disques durs modernes n'utilisent donc plus le mode DMA, même si la plupart d'entre eux le supportent parallèlement au mode Programmed I/O.

Busmaster DMA

Le Busmaster DMA représente une variante de "Direct Memory Access" mais n'a rien à voir avec le processeur DMA intégré dans l'unité centrale de l'ordinateur. Au cours de cette procédure, le contrôleur du disque dur transmet les données à l'aide de son propre contrôleur Busmaster DMA de manière autonome, vers la mémoire. On atteint alors un débit de transfert pouvant aller jusqu'à 8 Mo/s.

Le Busmaster DMA n'est utilisé en règle général qu'avec des contrôleurs SCSI puissants.

Adieu au mode protégé

Les deux modes de transfert DMA ne peuvent être utilisés que dans l'univers du mode réel. Le microprocesseur "plantera" lors d'une utilisation en mode protégé ou en mode 86 virtuel. Le problème se pose avec la gestion virtuelle de la mémoire gerée dans le CPU par le MMU (Memory-Management-Unit). Elle simule les adresses virtuelles, utilisées par un programme tournant sous ce mode, par rapport aux adresses physiques réelles.

Le programme ne s'aperçoit de rien, vu qu'il n'a jamais affaire aux adresses physiques ; le contrôleur DMA lui même n'est pas affecté car il n'a pas accès au MMU du CPU. Conséquence : les données sont déplacées dans des zones non souhaitées ce qui, à terme, entraîne immanquablement un "plantage" du système.

Ce destin ne frappera pas les programmes uniquement lors de l'utilisation de Windows en mode protégé ou d'un système d'exploitation du type OS/2. En effet, le microprocesseur est mis en mode virtuel 86 par le DOS dès l'installation du pilote de périphérique EMM386.SYS destiné à l'émulation de la mémoire paginée et qui dépend de la gestion virtuelle de la mémoire.

Il est donc nécessaire d'effectuer une surveillance du contrôleur DMA comme, par exemple, les contrôles de l'ensemble des ports d'entrée/sortie en mode protégé. Windows, par exemple, installe en arrière-plan un moniteur de contrôle virtuel qui surveille la programmation du contrôleur DMA par le BIOS ou un programme et convertit les adresses virtuelles indiquées en adresses physiques réelles avant qu'elles soient inscrites dans le registre du contrôleur DMA.

6.6. Enregistrement des informations sur un disque dur

Pour comprendre la manière dont les informations sont enregistrées à la surface du disque dur, il est tout d'abord nécessaire d'aborder le principe du codage binaire. Il n'est pas possible d'enregistrer les 0 et les 1 à l'aide des particules magnétiques, car on devrait alors appliquer aux valeurs 0 et 1 les deux états de "magnétisation" et "non magnétisation", ce qui, justement, n'est pas possible.

En effet, si l'on tentait de mettre cette technique en pratique, les suites de 0 et de 1 apparaîtraient sous la forme de suites de particules magnétisées et non magnétisées. La tête de lecture du disque dur ne serait alors pas capable de différencier chacune des particules magnétiques des autres. Il lui serait alors impossible de savoir, par exemple, si l'on a affaire à trois ou cinq 0.

Pour résoudre cette difficulté, il faudrait que la tête de lecture puisse connaître la "taille" d'une particule magnétique et soit à même de mettre en relation le temps écoulé pour une même magnétisation avec le nombre de particules magnétiques, et donc de bits. Il s'agirait en fait d'une sorte de cadence indiquant à chaque cycle l'arrivée d'un nouveau bit.

Mais il n'est pas possible de déterminer précisément la durée constante nécessaire à la production d'une cadence, et ce pour plusieurs raisons.

L'une d'entre elles est la vitesse de rotation du disque dur qui n'est pas complètement stable. D'autre part, il n'est pas possible de ne magnétiser qu'une seule particule à la fois. La magnétisation s'applique toujours un groupe de particules dont le nombre est lui-aussi variable.

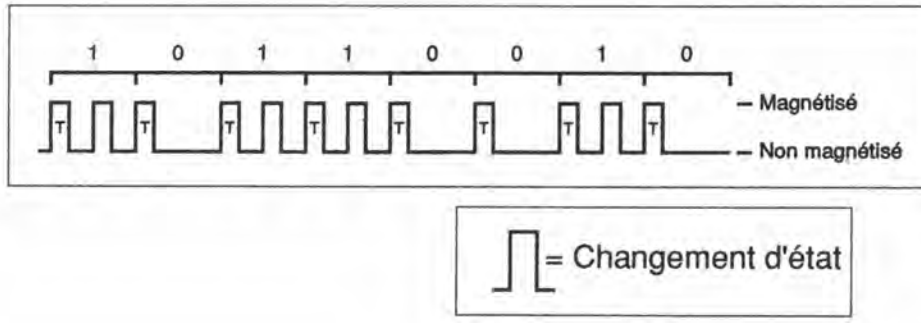
Les variations de signal électrique peuvent cependant être enregistrées sans problème ; il s'agit des brèves transitions qui se trouvent entre les particules magnétisées et celles non magnétisées. Elles provoquent dans la tête de lecture une impulsion de courant transmise aux composants électroniques où elles servira ensuite au décodage des données enregistrées sous la forme de 0 et de 1.

Ce sont justement ces questions de codage et de décodage des informations qui occupent l'esprit des développeurs de matériel. Car le nombre de variations de signal électrique pouvant être enregistrées par millimètre carré sur un disque dur est limité. Certes, le nombre de ces variations dépend des propriétés du matériau magnétique, de l'altitude de flottement de la tête de lecture/écriture et de sa vibrotaxie, mais il est néanmoins limité. Celui qui parviendra un jour à enregistrer sur le disque davantage de zéros et de uns à partir du même nombre de variations de signal électrique aura une bonne longueur d'avance dans la course au disque dur de très grande capacité.

6.6.1. La procédure FM

L'encodage le plus simple des 0 et des 1 à l'aide des variations de signal électrique consiste à enregistrer une variation pour chaque un et à ignorer une variation pour chaque 0. Les problèmes évoqués précédemment ne manqueront toutefois pas de se répéter. Car il faut indiquer ici également une cadence afin que le fait d'ignorer des variations de signal électrique pendant un certain laps de temps puisse être identifié comme une suite de zéros.

Lors du procédé d'enregistrement le plus simple et le plus ancien, appelé Modulation de Fréquence (FM: frequency modulation), le signal de cadence est enregistré directement sur le disque dur sous la forme d'une variation de signal électrique. A intervalles de temps constants, des variations sont ainsi enregistrées sur le disque dur sur lesquelles l'électronique du disque dur se synchronise pour lire les bits de données proprement dit. Ceux-ci se trouvent effectivement entre ces variations de signal électrique cadencées ; une variation pour un un, pas de variation pour 1 0.



Enregistrement selon la procédure FM

Cette procédure est réalisée à l'aide de composants électroniques de pilotages simples. Elle présente toutefois un inconvénient de taille : chaque bit d'information engloutit deux variations de signal électrique, réduisant du même coup la capacité potentielle du disque de moitié.

6.6.2. La procédure MFM

La procédure MFM met un terme à ce "gaspillage" de variations de signal électrique ; la plupart des disques durs en sont aujourd'hui équipés. Elle constitue une modification de la procédure MF, d'où son nom : "modified frequency modulation".

Au cours de cette procédure, les variations de signal électrique cadencées sont détournés sous formes de signaux de données. Dans un premier temps, rien ne change : un 0 se compose toujours et encore d'une variation de signal électrique cadencée suivie d'aucune autre variation jusqu'à la prochaine cadence. Un 1 n'est alors pas enregistré sous la forme d'une variation de signal électrique comme dans le cas de la procédure FM.

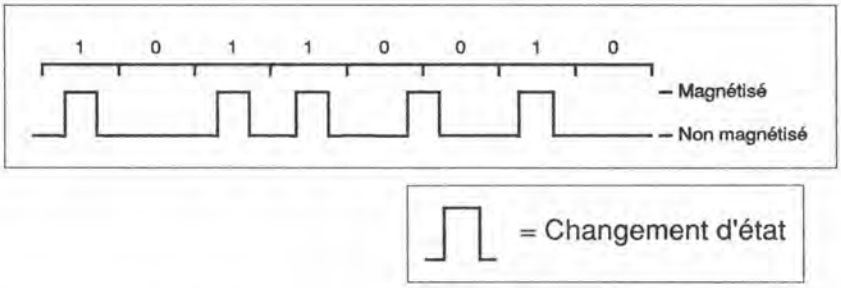
Il est ainsi possible d'enregistrer une suite de 0 et les 1 à partir d'une seule variation de signal électrique. Les suites de 0 et de 1 représentent donc une suite ininterrompue de variations de signal électrique.

Cette procédure d'encodage exige cependant une technique d'enregistrement ainsi qu'un système de pilotage électronique améliorés afin que le disque dur puisse se synchroniser sur le déroulement normal des variations électriques cadencées dans le cas de longues chaînes de 0. Si la lecture d'un 0 est suivie d'un laps de temps plus long que la normale jusqu'à apparition de la prochaine variation, un 1 sera alors identifié.

Un problème peut se poser lorsqu'un 1 suivi d'un 0 qui exige à nouveau une variation de signal électrique en position de cadence normale. Car la durée entre la variation de

signal électrique du un et celle du 0 n'est plus que de la moitié de l'intervalle normal existant entre ces deux variations. Mais le plus petit intervalle entre deux variations ne peut être davantage réduit, la tête de lecture et le système électronique déclarant alors forfait. C'est pourquoi aucune variation de signal électrique n'est enregistrée pour un 0 qui suit un 1.

La prochaine variation de signal électrique se produit alors après un laps de temps égal à une fois et demi celui nécessaire à l'exécution d'une variation de signal électrique (combinaison de bits 100b), voire même après le double de cette durée (combinaison de bits 101b). L'illustration suivante nous le montre clairement.



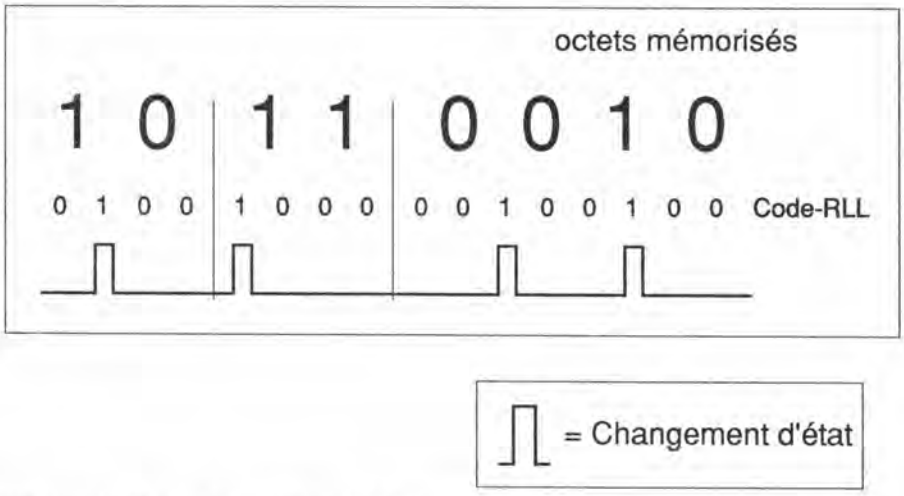
Enregistrement selon la procédure MFM

6.6.3. La procédure RLL

Ceux qui pensaient que l'enregistrement de données avait atteint les sommets de la perfection avec la procédure MFM seront déçus. En effet, la procédure d'encodage RLL permet d'enregistrer 50 pour cent de plus d'informations sur le disque que la procédure MFM.

Dans le cadre de cette procédure, les 1 sont représentés par des variations de signal électrique et les 0 par l'absence de variations, sans tenir compte en plus d'une variation de signal électrique cadencée. Le système électronique du disque dur doit lui même compter les cadences. Même dans le cas d'une rotation constante du disque dur et d'un perfectionnement de la tête de lecture et de son système électronique, ceci n'est envisageable que si le nombre de 0 situés en deux 1 n'est pas trop élevé. En effet, chaque 0 d'une chaîne agrandit l'intervalle jusqu'à la prochaine variation de signal électrique ; avec lui grandit également la probabilité que le système électronique du disque dur perde la cadence.

D'autre part, les différents 1 d'une chaîne ne doivent pas se suivre de trop près ; le système électronique risque en effet de pas pouvoir tenir la cadence.



Enregistrement selon la procédure 2,7 RLL

Afin de tenir compte de ces deux exigences, le système électronique du disque dur, dans le cas de la procédure d'encodage RLL (RLL: run length limited), représente les chaînes de bits par un autre code qui sera ensuite enregistré. Ce nouveau code est deux fois plus long que le code original mais il veille néanmoins à ce que les chaînes de 0 ne soient pas trop longues et à ce que les intervalles entre les 1 ne soient pas trop réduits. Ceci implique, dans tous les cas, que les informations enregistrées puissent être ensuite décodées sous la forme des chaînes de bits initiales sans que naissent des ambigüités.

Les types de codes possibles sont innombrables : tout dépend du nombre maximum et du nombre minimum de 0 consécutifs que l'on souhaite. Deux procédés sont toutefois couramment utilisés et portent, en raison de ces deux tailles, les nom de RLL 2,7 et RLL 3,9.

RLL 2,7 représente le procédé standard dont sont équipés la plupart des disques durs récents. Entre deux 1 apparaissent au moins deux 0, au maximum sept 0. La capacité augmente d'au moins 50 pour cent par rapport à la procédure MFM.

Les informations peuvent être enregistrées avec une plus grande densité avec la procédure RLL 3,9, appelée également "advanced RLL". Dans le cadre de cette procédure, on trouve entre deux 1 au moins trois 0 et au maximum neuf.

La table suivante montre la procédure d'encodage de la procédure RLL 2,7. Vous remarquerez que certains octets, tel, par exemple, l'octet 00000001b, ne peuvent être transposés. Il ne faut cependant pas oublier que, à ce niveau, ce sont les secteurs et non pas les octets qui sont traités. Même l'octet 00000001b se laisse transposer ; il suffit pour cela d'inclure le bit de l'octet suivant dans l'encodage.

Table des codes / Procédure RLL 2,7	
Suite de bits	**Code**
000	000100
10	0100
010	100100
0010	00100100
11	1000
011	001000
001	00001000

Le problème qui se pose au cours de l'encodage concerne le dernier octet d'un secteur qui n'est en effet suivi d'aucun octet. Dans la pratique, ce problème est résolu de la manière suivante : le contrôleur du disque dur complète simplement par un octet qui convient. Les bits superflus sont ensuite enlevés au cours de la lecture et le dernier octet du secteur peut alors être correctement renvoyé.

6.7. Plus petit, plus rapide, moins cher

"Smaller, Faster, Cheaper" : c'est sous ce qu'il est convenu d'appeler un "slogan" qu'est placée, depuis quelques années, le progrès en matière de technologie de disques durs. Quelques faits vont nous éclairer :

- ✓ les temps d'accès sont passés de trente millisecondes à quelques dix millisecondes sur les modèles de pointe.

- ✓ les capacités courantes atteignent aujourd'hui plusieurs centaines de Méga-octets. Les disques durs ayant une capacité de plus de 1 Go ne sont plus une rareté.

Le stockage sur disque dur revient maintenant dix fois moins cher que celui en mémoire vive.

Le disque dur a évolué dans toutes ses dimensions essentiellement parce que chaque phase de traitement, chaque détail relevant de la technique d'enregistrement ou de transmission ont été optimisés. Cela commence déjà avec la disposition des différents secteurs sur le disque dur et avec le fameux facteur d'entrelacement (Interleave Factor).

6.7.1. Le facteur d'entrelacement

Aujourd'hui, les contrôleurs de disque dur sont tellement rapides qu'ils peuvent se permettre, lors de la lecture d'un seul secteur, de lire la totalité de la piste, même lorsque le programme ne leur a pas demandé de les lire. Cela se produit dans la plupart des cas, à condition toutefois que le disque ne soit pas trop fragmenté. Si ce n'est pas le cas, le contrôleur du disque dur ne sera pas obligé, lors de l'accès suivant, d'effectuer une nouvelle lecture depuis le disque dur ; il lui suffira de ressortir les données de son tampon interne. Ceci a pour effet d'accélérer de manière sensible les accès.

Ceci n'est toutefois vrai que pour les contrôleurs récents (SCSI, ESDI et IDE). Les contrôleurs ST506 ne possèdent en général qu'un seul tampon de secteur interne qui ne peut pas contenir la totalité d'une piste. Mais il y a plus grave : sachant que le secteur lu doit, dans un premier temps, être transmis au CPU, dès que le dernier octet a été lu, le tampon de secteur ne sera donc pas encore libéré lorsque le secteur suivant sera déjà sous la tête de lecture. Il ne pourra donc être lu d'avance.

Et si l'accès en lecture suivant concerne effectivement ce secteur, celui-ci ne pourra pas être lu immédiatement ; le disque dur devra effectuer une rotation complète avant que le secteur ne se retrouve à nouveau sous la tête de lecture. Cette procédure se répétant pour chaque secteur, réduisant ainsi considérablement la vitesse d'accès au disque dur, les concepteurs de matériel ont dès le départ recherché une solution à ce problème. Et il l'ont trouvée sous la forme de l'"entrelacement" qui consiste à décaler les secteurs logiques des secteurs physiques.

Ceci permet, suite à un secteur logique, de laisser défiler sous la tête de lecture quelques secteurs avant que n'apparaisse le secteur logique suivant. Le nombre de ces secteurs "sautés" permet que le secteur logique suivant ne se présente sous la tête de lecture qu'au moment précis où le contrôleur de disque dur aura terminé le transfert du précédent et que le BIOS aura donné l'ordre de transférer le suivant.

L'entrelacement se mesure grâce au facteur d'entrelacement qui est beaucoup plus connu que la procédure en elle-même. Ce facteur représente le nombre de secteurs de décalage entre les secteurs logiques et les secteurs physiques. Sur les disques durs dont IBM équipait ses PC/XT, ce facteur était de 1:6. Pour les disques durs des PC/AT, ce facteur est alors passé à 1:3. Le facteur d'entrelacement sur les AT pouvait néanmoins être rabaissé à 1:2, ce qui avait pour effet d'augmenter la vitesse d'accès.

Aujourd'hui, le facteur d'entrelacement habituel est de 1:1, ce qui signifie qu'il n'y a plus d'entrelacement.

En revanche, dans le cas d'un facteur d'entrelacement de 1:6, il est nécessaire de "sauter" 5 secteurs physiques avant de parvenir au prochain secteur logique, et dans le cas d'un facteur 1:3, il y en a deux. La table suivante montre les conséquences du facteur d'entrelacement sur le rapport entre secteurs logiques et physiques en prenant comme exemple une piste contenant 17 secteurs.

Entrelacement / Disques durs IBM, PC/XT et PC/AT			
AT		XT	
Secteurs physiques	Secteurs logiques	Secteurs physiques	Secteurs logiques
1	1	1	1
2	7	2	4
3	13	3	7
4	2	4	10
5	8	5	13
6	14	6	16
7	3	7	2
8	9	8	5
9	15	9	8
10	4	10	11
11	10	11	14
12	16	12	17
13	5	13	3
14	11	14	6
15	17	15	9
16	6	16	12
17	12	17	15

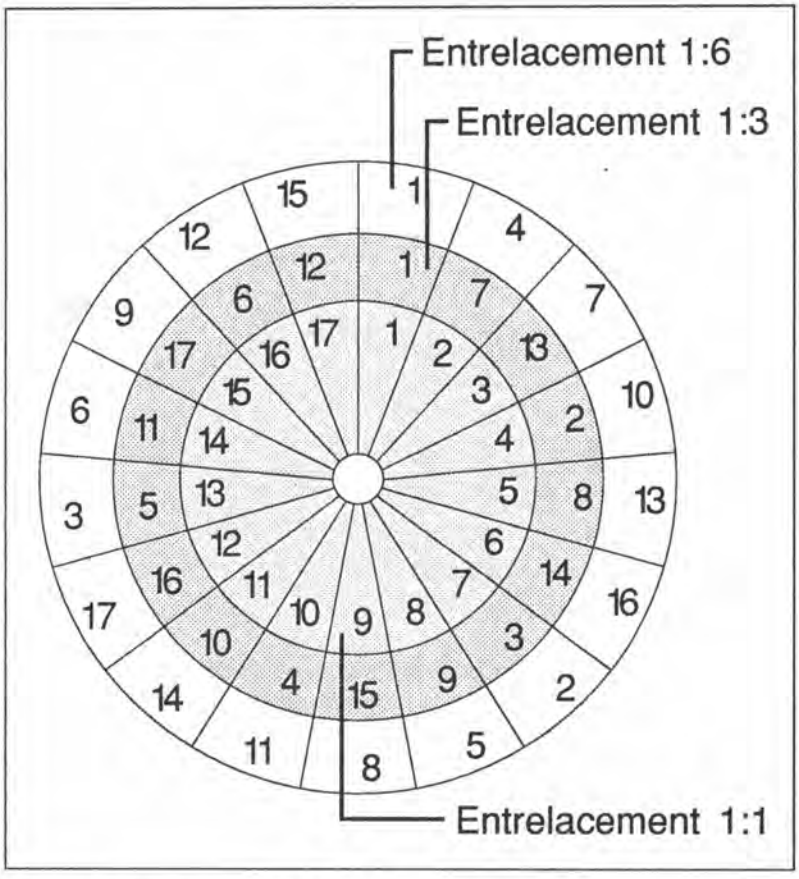
En dépit des avantages incontestables de ce procédé, il ne faut pas oublier qu'il est préférable qu'il n'y ait aucun entrelacement. Ce n'est que dans ce cas de figure qu'une piste complète peut être lue au cours d'une seule et même rotation du disque dur, alors que, selon le facteur d'entrelacement actif, cette lecture nécessitera deux, trois, voire davantage de rotations.

Installation de l'entrelacement

L'entrelacement est installé au cours du formatage de bas niveau du disque dur au cours duquel les marques d'adresse et les numéros de secteurs sont inscrits sur le disque dur encore vierge. Indépendamment du système d'exploitation qui sera ensuite copié sur le disque dur au cours d'une procédure de formatage distincte ; c'est au cours de ce formatage de bas niveau que sont installées les structures fondamentales d'identification des secteurs.

Il est tout à fait possible de modifier la position des secteurs vu que le numéro logique de chaque secteur est choisi par le programme de formatage de bas niveau. C'est d'ailleurs pour cela la plupart de ces programmes demande à un moment donné le facteur d'entrelacement choisi. Si on choisi une valeur erronée, on est sûr d'avoir par la suite de mauvaises surprises. L'ordinateur risque alors de mettre beaucoup plus de temps pour charger programmes et fichiers.

Cela signifie donc que le facteur d'entrelacement choisi ne convient pas. Il est toutefois possible de le modifier à posteriori. Il suffit d'avoir recours aux programmes adaptés, par exemple aux Norton Utilities. Ces programmes communiquent à l'utilisateur le facteur d'entrelacement optimal et sont capables d'effectuer le formatage de bas niveau à posteriori. Ca marche, ça ne dure que quelques minutes et ça en vaut la peine.

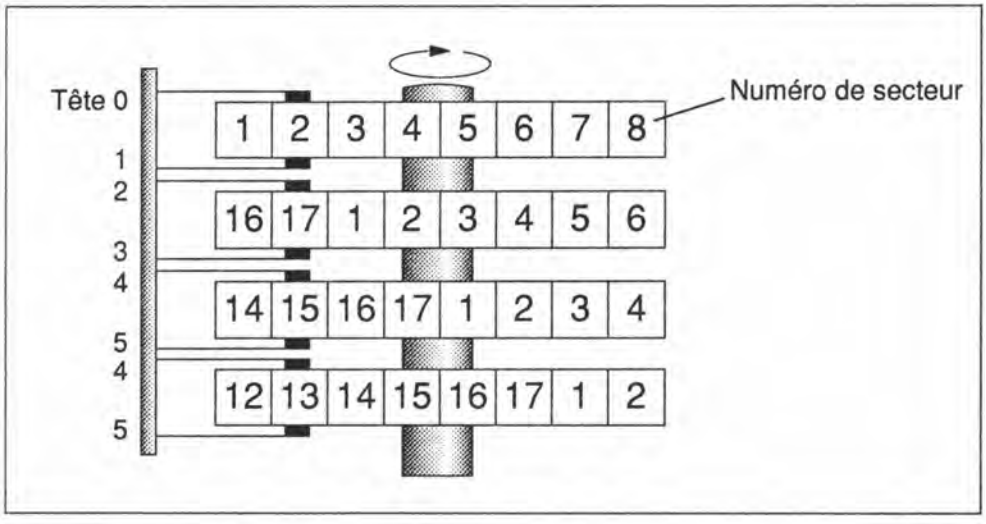


Différents facteur d'entrelacement et leur influence respective sur le formatage d'une piste

6.7.2. Track Skewing et Cylinder Skewing

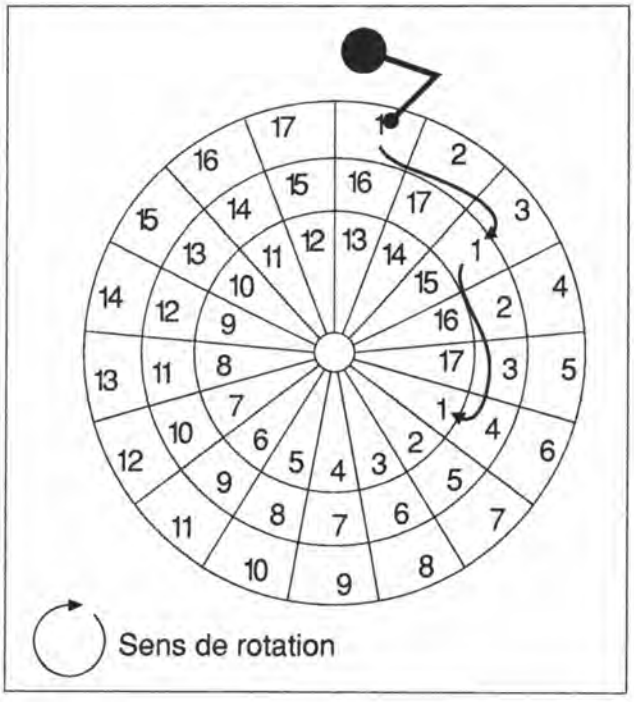
Un encodage intelligent des numéros de secteurs logique ne se ressent pas uniquement en ce qui concerne la lecture séquentielle d'une piste. En effet, un accès au cylindre suivant fait souvent suite à la lecture d'une piste. C'est ce dont se charge le système d'exploitation qui, au cours de la numérotation des secteurs, commence par examiner les différents cylindres d'une piste avant de passer au cylindre suivant. Le passage d'une tête de lecture/écriture à l'autre d'un même cylindre prend effectivement beaucoup moins de temps que le déplacement de l'ensemble du bras de lecture/écriture.

Et pourtant : le passage d'une tête de lecture/écriture à l'autre prend également un peu de temps, et le disque dur continue sa rotation. Lorsque que le dernier secteur d'une piste de cyclindre est lu, le secteur suivant se trouvant sur un autre plateau, est déjà passé sous la tête de lecture/écriture, et il faut donc attendre le temps d'une rotation supplémentaire pour pouvoir de nouveau y accéder.



Pour empêcher ce phénomène, il existe une sorte d'entrelacement concernant les pistes d'un cylindre appelé "Cylinder-Skewing". Au cours de cette procédure, les secteurs de chaque piste d'un cylindre sont placées le plus près possible les uns des autres, de telle manière que, en dépit du passage sur la tête suivante, le premier secteur de la piste peut être lu immédiatement. Cette procédure est également installée au cours du formatage de bas niveau. Nous ne connaissons toutefois aucun programme qui, comme dans le cas du facteur d'entrelacement, permettrait d'en effectuer une optimisation à posteriori.

Parallèlement au Cylinder-Skewing, il existe également une autre procédure appelée "Track-Skewing". elle fonctionne selon le même principe et tient compte de la durée nécessaire au bras de lecture/écriture pour passer au cylindre suivant.



Track-Skewing avec un facteur d'entrelacement 1:1

6.7.3. Multiple Zone Recording

Si on souhaite inscrire davantage de secteurs sur le disque dur, on peut utiliser un truc très simple. En effet, les pistes extérieures disposent de plus de place que les pistes intérieures. Auparavant, il n'était pas possible de tirer profit de cet avantage car le nombre de secteurs par piste était constant, déterminé par la capacité de la piste le plus à l'intérieure, et donc la plus courte.

Avec l'arrivée des disques SCSI et IDE, qui ne font que simuler au BIOS les informations relatives au nombre de têtes, de pistes et de secteurs, il est maintenant possible d'effectuer un formatage variable des différentes pistes. En effet, ces tandems contrôleur-disques convertissent les numéros de tête, de cylindre et de secteur en une autre adresse de secteur et tiennent ainsi compte de la piste la plus à l'extérieur.

Ceci a cependant pour effet d'exiger davantage du contrôleur et de la tête de lecture/écriture, alors contraints de lire/écrire davantage de secteurs sur les pistes extérieures que sur les pistes intérieures au cours d'une même rotation. Alors que

l'amplitude des variations de signal électrique baisse, le nombre de données lues ou écrites au cours d'une rotation augmente, et avec lui le débit de données.

Il est ainsi possible d'augmenter de 20 à 50 pour-cent la capacité d'un disque dur.

6.7.4. Correction d'erreurs

Un des critères principaux ayant freiné au départ la diffusion des disques durs fut leur propension à l'erreur. De fait, des impuretés pénètrent inévitablement sur la surface magnétique, rendant ainsi certains secteurs du disque dur inutilisables.

Auparavant, les numéros des secteurs défectueux détectés au cours du contrôle de qualité étaient notés sur le capot du disque dur. Lors du lancement du programme de formatage de bas niveau, l'utilisateur devait indiquer les numéros des secteurs défectueux afin qu'ils soient repérés et identifiables par la suite par le système d'exploitation. Ces zones étaient alors repérées comme défectueuses dans la FAT (Table d'Allocation des Fichiers) et donc exclues.

Sur les disques durs modernes, notamment sur les disques IDE, ces listes d'erreurs ne sont plus présentes sur le capot du disque dur. Leurs adresses sont soit consignées directement par le fabricant sur des pistes de données distinctes, soient identifiées au cours du formatage de bas niveau. Ces secteurs sont ensuite tout simplement ignorés ou remplacés par d'autres secteurs déplacés vers des zones plus sûres. C'est là que se trouvent également les tables de gestion des secteurs défectueux ou remplacés ; il suffit alors au contrôleur de disque de les charger au moment de son initialisation. Lors d'un accès à un secteur défectueux, il peut alors se positionner sur celui qui le remplace. Cela ralentit certes l'accès au disque mais est à peine perceptible en raison de la proportion peu élevée de secteurs défectueux.

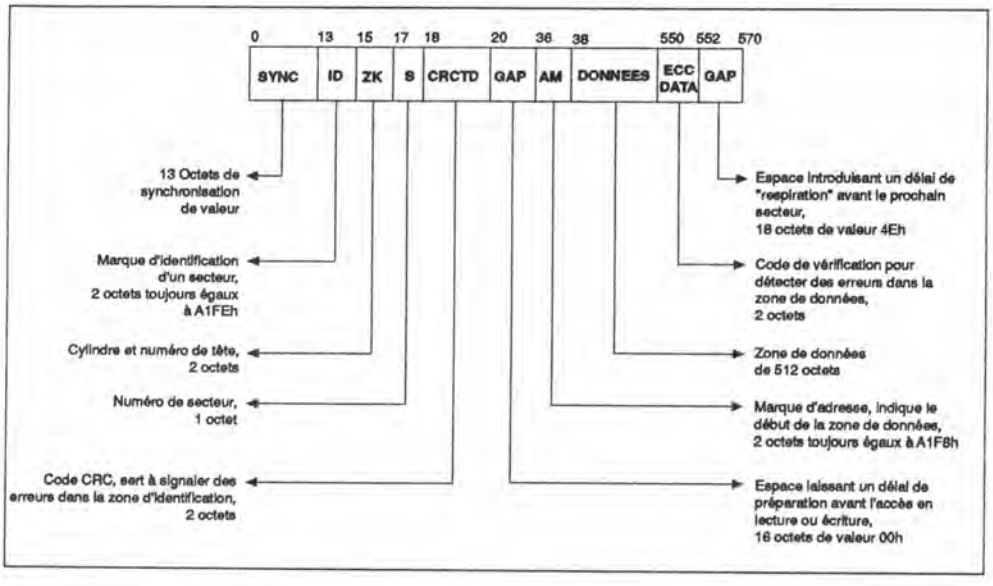
Qui plus est, de nombreux disques sont capables d'identifier en cours d'utilisation les secteurs défectueux. C'est d'ailleurs pourquoi chaque secteur est enregistré avec un code de correction d'erreur qui permet d'identifier les erreurs d'enregistrement de secteurs et, parfois, de les corriger. Ces secteurs sont alors repérés comme étant défectueux, puis remplacés par un secteur de réserve dans lequel sont stockées les informations reconstruites.

Cette procédure n'est cependant possible que dans le cas d'un collaboration étroite entre le disque dur et le contrôleur, donc exclusivement avec les disques IDE.

6.7.5. Autres informations du disque dur

Le disque dur doit contenir de nombreuses informations complémentaires. Un simple secteur doit être accompagné de toute une série d'informations complémentaires afin de pouvoir être identifié et lu correctement. Le format des informations diffère entre

selon le type de contrôleur mais, en principe, ce sont toujours les mêmes informations qui sont enregistrées : les numéros de cylindre et de secteur ainsi que le code d'erreur et de correction qui permet l'identification des erreurs. L'illustration suivante montre la structure d'un secteur avec un contrôleur de type ST506.



Format d'un secteur lors de l'utilisation d'un contrôleur de type ST506

Comme vous pouvez le voir, chaque secteur débute par une chaîne de 13 octets de synchronisation ayant la valeur 00h. Il en résulte une suite stable de variations du signal électrique qui permet au contrôleur de se synchroniser sur le secteur.

Vient ensuite la zone ID grâce à laquelle le secteur s'identifie. Il s'agit notamment des numéros de cylindre, de tête et de secteur du secteur concerné. Ces informations commencent par un octet ID spécial et se terminent par un second octet incluant le code d'erreur qui permet de vérifier la validité de ces informations.

Vient ensuite un trou qui offre une pause au contrôleur. Cette pause est nécessaire pour qu'il puisse traiter les informations de la séquence ID et constater que le secteur recherché est bien celui auquel il à affaire. Cette pause lui permet également de se resynchroniser en cas de nécessité.

Ce n'est que maintenant que viennent les données proprement dites rangées dans la zone de données du secteur. Elle est précédée d'un octet d'identification (champ : AM). Et, afin que le contrôleur puisse vérifier l'ordre d'arrivée des données, deux octets de correction viennent s'ajouter (champ : ECC-DATA).

Et pour laisser au contrôleur le temps de vérifier les octets de correction, le secteur sont suivis une fois encore par une zone sans signification qui ne contient, cette fois, que des octets ayant la valeur 4Eh. Un secteur occupe de ce fait 570 octets sur le disque dur.

Extra-pistes

En dehors des informations sur les secteurs, le disque dur contient d'autres informations. Par exemple les "Servo-Pistes" dans lesquelles sont enregistrées des variations de signal électrique selon un schéma connu. C'est en lisant ces pistes que l'électronique va se synchroniser sur la cadence des données nécessaire à la bonne compréhension des informations stockées, notamment lors de la procédure RLL.

Certains disques contiennent en outre des "pistes de réserve" dont l'existence n'est pas déclarée au BIOS et qui ne peuvent donc être sollicitées pour l'enregistrement de données. Elles constituent une sorte de réservoir et sont destinées à remplacer les secteurs défectueux.

Mentionnons pour finir la zone de parquage où se retirent les têtes de lecture/écriture lorsque l'ordinateur est éteint. Ce parquage automatique est aujourd'hui présent sur toutes les nouvelles machines et permettent d'éviter les "Head-Crashs", jadis fréquents lors du déplacement des ordinateurs, et qu'il était nécessaire d'avoir recours à des programmes spéciaux pour parquer les têtes de lecture/écriture.

6.7.6. Temps d'accès et mesure

La qualité d'un disque dur ne se mesure pas essentiellement à sa taille ou à sa capacité, mais en premier lieu à la vitesse des temps d'accès. Les fabricants et distributeurs de PC évoquent souvent des chiffres fluctuant entre 10 et 30 millisecondes et qui sont présentés comme étant le temps d'accès du disque dur.

Cette valeur relève malheureusement plus de l'"épate" (le mien est plus rapide) et ne reflète pas de manière réaliste la vitesse du disque dur, et encore moins celle du contrôleur. Il ne s'agit en effet que du temps d'accès moyen (Average-Seek-Time) qui s'écoule entre deux accès à des données. Cette valeur ne représente donc qu'une moyenne communiquée par le fabricant qui n'engage que lui.

La plupart des fabricants de disques durs communiquent en général deux autres valeurs, exprimées elles aussi en millisecondes : le Track-to-Track-Seek-Time et le Maximum-Seek-Time.

Le concept de "Track-to-Track-Seek-Time" est un peu flou ; pour les uns, il représente le temps de passage d'une tête à une autre dans un même cylindre, pour les autres, il s'agit du temps nécessaire au bras de lecture/écriture pour se positionner sur la piste suivante. En toute honnêteté, nous sommes incapables de trancher.

Le concept de "Maximum-Seek-Time" est moins contesté. Il représente le cas le plus extrême, lorsque le bras de lecture/écriture doit se déplacer sur tout le diamètre du disque, de la première à la dernière piste.

Les facteurs énoncés précédemment influent certes fortement sur la vitesse du disque dur lors d'accès aux fichiers à partir du DOS, mais il existe d'autres sources qui ont leur importance. En effet, à quoi pourrait bien servir un disque dur rapide si le contrôleur ne suit pas ou si le disque dur est fortement fragmenté, entraînant par là-même de nombreux déplacements des têtes de lecture/écriture ? De plus, il faut tenir compte des cheminements à travers les différentes couches de programmes, les accès au DOS, aux périphériques, au BIOS, à certains pilotes spéciaux de disque dur.

Souvent, lorsqu'on ne fait pas confiance aux informations fournies par le fabricant ou que l'on souhaite tester dans des conditions réelles les performances du disque dur, on a recours à des programmes d'évaluation. On notera par exemple le programme SystemInfo (SI) des Norton Utilities, ou encore le célèbre programme CORETEST du fabricant américain de disques durs, Core International.

Ces programmes mesurent en général les débits de transfert de données en procédant à la lecture d'un échantillon de données sur le disque dur. Toutefois, la taille du bloc de données concerné est limitée à la taille d'un cylindre afin que le bras de lecture/écriture n'ait pas à effectuer de déplacement pendant l'accès. C'est d'ailleurs ce qui rend discutable le résultat de ces programmes d'évaluation, notamment avec les disques ESDI, SCSI et IDE. Dès que le contrôleur cache au BIOS les véritables "caractéristiques" du disque dur et lui simule un format connu, un changement de piste au cours de ces accès n'est alors plus exclu. Ce changement provoque un délai inattendu et réduit le résultat de l'évaluation, en faveur bien sûr du disque dur.

Si l'on souhaite ensuite mesurer le temps de déplacement d'une piste à une autre, le même problème va se poser. Vu que le changement de piste n'est possible que par une lecture de deux secteurs sur deux pistes contigües, la mécanisme de conversion interne des adresses de secteur du disque dur veille entre autre à ce qu'aucun changement de piste n'ait lieu. La valeur 0, représentant le temps de changement de piste, s'affiche alors à l'écran, vu que les secteurs ont pu être lus consécutivement.

Cache contrôleur et programmes cache

Les résultats de ces programmes de test deviennent plus que suspects lorsque le système est équipé de programmes cache. Concernant les logiciels, on citera le programme SMARTDRV de Microsoft. Il s'implémentent dans l'interruption disque dur du BIOS et interceptent les appels en lecture ou en écriture des programmes d'application et des pilotes de périphériques du DOS.

Ils transmettent tout à fait normalement les appels en lecture au contrôleur du disque dur mais enregistrent les informations lues dans un tampon cache qui leur est propre avant de les renvoyer.

Selon la taille de la mémoire cache réservée, celle-ci contient un volume plus ou moins important de données lues. Lorsque le programme cache détecte en mémoire cache le secteur appelé, il peut l'en extraire directement, sans avoir à effectuer une nouvelle lecture sur le disque dur. Ceci a pour effet d'accélérer sensiblement les accès, et l'absence d'accès due à la mémoire cache n'est naturellement pas prise en compte par le programme de test.

Mais quelques astuces permettent de contourner l'écueil. Il suffit en effet que le programme test, lors de son lancement, examine l'interruption disquette 13h du BIOS et qu'il vérifie si celle-ci pointe encore vers le BIOS. Si tel n'est pas le cas, il est fort probable qu'un programme cache est installé et le programme test peut exiger de l'utilisateur qu'il le désactive avant le début des tests.

Les mécanismes de cache restent toutefois absolument indétectables lorsqu'ils ont été implémentés au niveau de la machine. il existe par exemple depuis longtemps des contrôleurs de disques durs qui se chargent de cette fonction et qui disposent eux-mêmes d'une mémoire cache qui peut être de l'ordre de 16 Mo. L'ordre de lecture sera alors intercepté non pas au niveau du BIOS, mais déjà au niveau de la machine - le programme de test n'y voit alors que du feu.

6.8. Les partitions d'un disque dur

Si vous avez déjà eu l'occasion d'installer vous-même un disque dur ou bien d'étendre votre PC à l'aide d'un système d'exploitation tel que XENIX ou OS/2, vous avez déjà eu à faire à FDISK, le programme de partition du disque dur de MS DOS qui ouvre la porte à l'exploitation d'un disque dur haute capacité mais également à l'installation en parallèle de plusieurs systèmes d'exploitations sur un ordinateur.

Procédure de formatage

FDISK ne représente en réalité qu'une des trois étapes de l'opération de formatage qui consiste à organiser la partition d'un disque dur en le préparant à recevoir un ou plusieurs systèmes d'exploitations. Le formatage de bas niveau constitue la première étape ; il subdivise le disque dur physiquement en pistes (cylindres) et en secteurs, en écrivant les marques d'adresses appropriées sur le disque. Ce (pré)formatage est rendu nécessaire par le fait que beaucoup de disques durs sont "soft-sectored", comme on dirait dans le cas d'une disquette. Les marques d'adresses, nécessaires à l'identification des différents secteurs, ne sont donc pas pré-installées sur le disque dur.

Les possesseurs de PC compatibles XT se souviennent certainement que le formatage de bas niveau devait être lancé sur ces machines à l'aide du programme DEBUG du DOS. Une routine du BIOS devait en effet être appelée à partir de DEBUG. Le formatage de bas niveau n'est heureusement plus aussi incommode car la plupart des fabricants de disques durs joignent à leur lecteur des programmes appropriés se chargeant automatiquement de ce travail.

Partitionnement du disque dur

L'étape suivante consiste à organiser la partition du disque dur, c'est à dire à le subdiviser en zones bien délimitées. La fonction première de la partition était à l'origine de préparer des zones distinctes sur le disque dur qui puissent être prises en charge et gérées par des systèmes d'exploitations différents sans que la différence dans la structure mise en place par les divers systèmes soit source de conflits.

La chute des prix à laquelle ont a assisté dans le domaine des matériels PC a fait naître une raison supplémentaire pour la partition du disque dur en rendant abordable des disques durs disposant d'une capacité non plus de 10 ou 20 méga-octets mais de 40,80 méga-octets voir beaucoup plus. Or, le DOS ne pouvait, du moins jusqu'à la version 3.3, exploiter pleinement de telles capacités car, d'une part, il ne pouvait gérer que des mémoires de masses d'une capacité n'excédant pas 32 méga-octets, et, d'autre part, il n'était pas à même de diviser un disque dur en plusieurs unités utilisables sous DOS.

Ce problème a été résolu par la version 3.3 du DOS qui, tout en maintenant la limite de 32 méga-octets, permet de mettre en place, à côté d'une "partition primaire" (primary Partition), qui doit se trouver à l'intérieur des 32 premiers méga-octets du disque dur, jusqu'à 23 "partitions étendues" (extended Partition), portant les désignations des lecteurs D à Z. Chaque partition étendue pouvant elle aussi comporter jusqu'à 32 méga-octets, la capacité de disque dur soutenue s'est ainsi trouvé rehaussée sous DOS 3.3 à 768 méga-octets.

La version 4.0 du DOS va encore plus loin puisque, grâce à quelques réorganisations dans le domaine des pilotes de périphériques, elle soutient désormais des mémoires de masse d'une capacité pouvant atteindre 2 Go. Nombreux sont cependant les utilisateurs qui continuent à préférer la partition du disque dur en petites unités (c'est-à-dire, en secteurs logiques), car cela permet plus facilement d'obtenir une organisation plus limpide qu'une seule grosse unité de disque sur laquelle on peut loger des centaines et des centaines de fichiers.

Alors que la partition primaire doit se trouver à l'intérieur des 32 premiers méga-octets, la partition étendue peut être installée à n'importe quel endroit. FDISK identifie ce type de partition en leur attribuant les noms "PRI DOS" et "EXT DOS".

Le secteur de partition

Sous toutes les versions du DOS, la partition du disque dur repose sur le secteur de partition que FDISK met en place dans le premier secteur du disque dur (tête 0, cylindre 0, secteur 1). Ce n'est donc pas le secteur d'amorçage du DOS mais ce secteur que le BIOS, après une réinitialisation ou un lancement du système, charge tout d'abord à l'adresse 0000:7C00 de la mémoire si aucune disquette ne figure dans le lecteur A:. S'il rencontre la séquence de code 55h, AAh dans les deux derniers des 512 octets de ce secteur, il considère qu'il s'agit d'un secteur exécutable et il commence l'exécution du programme par le premier octet du secteur. Dans le cas contraire, le BIOS sort un message d'erreur et, suivant la version et le fabricant, il s'installe dans une boucle sans fin ou lance le BASIC.

Structure du secteur de partition d'un disque dur		
Adresse	**Contenu**	**Type**
+000h	Code de partition	Code
+1BEh	1ère entrée dans la table de partition	16 OCTETS
+1CEh	2ème entrée dans la table de partition	16 OCTETS
+1DEh	3ème entrée dans la table de partition	16 OCTETS
+1EEh	4ème entrée dans la table de partition	16 OCTETS
+1FEh	Code d'identification (AA55h) permettant d'identifier le secteur de partition comme tel	2 OCTETS
Longueur : 200h (512 octets)		

Il incombe au code programme du secteur d'amorçage, lorsqu'il est lancé, d'identifier quelle est la partition activée et ainsi de détecter le système d'exploitation à lancer, de charger son secteur d'amorçage puis de lancer l'exécution du code programme qu'il contient. Comme ce code programme doit, par définition, figurer également à l'adresse 0000:7C00 de la mémoire, le code de partition se décale tout d'abord vers l'adresse 0000:0600 et fait ainsi la place pour le secteur d'amorçage.

La table des partitions

La routine identifie le positionnement du secteur d'amorçage sur le disque dur ainsi que la partition à laquelle il appartient à partir du contenu de la table des partitions qui se trouve à l'intérieur du secteur de partitions à l'adresse 1BEH.

Structure d'une entrée de la table de partition		
Adr.	**Contenu**	**Type**
+00h	Etat de partition 00h = non activée 80h = partition boot	1 OCTET
+01h	Tête de lecture/écriture où commence la partition	1 OCTET
+02h	Secteur et cylindre où commence la partition	1 MOT
+04h	Type de partition 00h entrée non utilisée 01h DOS avec FAT 12 bits (primary part.) 02h XENIX 03h XENIX 04h DOS avec FAT 16 bits (primary part.) 05h extended DOS partition (à partir de DOS 3.3) 06h partition DOS 4.0 avec plus de 32 Mo DBh concurrent DOS D'autres codes sont possibles avec d'autres systèmes d'exploitation ou avec des drivers particuliers.	1 OCTET
+05h	Tête de lecture/écriture où se termine la partition	1 OCTET
+06h	Secteur et cylindre où se termine la partition	1 MOT
+08h	Distance en secteurs du premier secteur de la partition (secteur boot) au secteur de partition	4 OCTETS
+och	Nombre de secteurs de cette partition	4 OCTETS
Longueur : 10h (16 octets)		

Chaque partition est décrite dans cette table par une structure de 16 octets. Cette table se trouvant presque déjà à la fin du secteur de partition, elle n'offre de place que pour 4 entrées, ce qui limite, par conséquent, le nombre de partitions à 4. C'est pourquoi certains fabricants fournissent des programmes de configuration spéciaux qui permettent de loger un plus grand nombre de partitions sur un disque dur en avançant la table à l'intérieur des secteurs de partition et en installant un nouveau code de partitions accédant à la table décalée sans que la structure de base de cette table ne soit modifiée pour autant.

De plus, l'utilisateur dispose d'une option "Multiple-Boot" qui lui permet de décider, au cours de l'exécution du code de programme, quelle partition doit être lancée. Ceci permet d'utiliser parallèlement plusieurs systèmes d'exploitation et de sélectionner lors de chaque lancement du système le système d'exploitation souhaité.

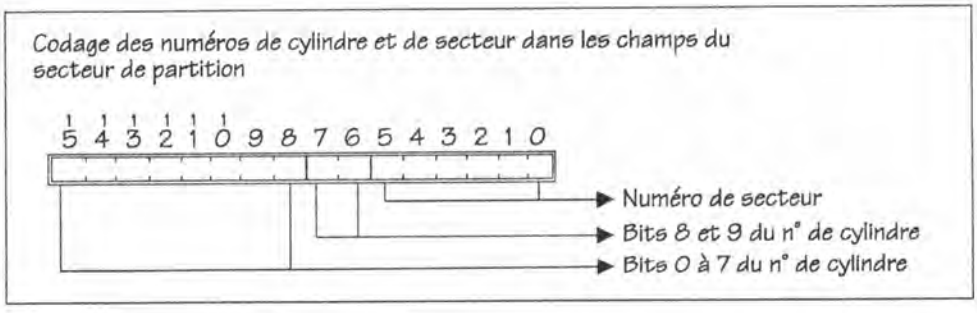
Ces différents programmes ne modifient toutefois nullement la structure fondamentale. Notez cependant que les diverses entrées de partitions ne commencent pas toujours avec la première entrée de la table. Il est par exemple parfaitement possible que la

partition unique d'un disque dur ne soit pas décrite par la première entrée de la table mais par la deuxième, la troisième, voire même la quatrième.

Lancement de la partition d'amorçage

Le premier champ de la structure de partition permet de savoir s'il s'agit de la partition boot. La valeur 00h signifie ici "non activé" alors que la valeur 80h désigne la partition à utiliser pour le démarrage du système. Si, en examinant la table, le code de partition ne trouve pas de partition boot ou s'il en trouve plusieurs, ou encore s'il rencontre un code inconnu, il interrompt l'opération d'amorçage avec un message d'erreur et se lance dans une boucle sans fin que seule une réinitialisation peut interrompre.

Lorsque le code de partition a identifié la partition boot, c'est-à-dire celle qui doit servir au lancement du système, les deux champs suivants lui indiquent la situation de cette partition sur le disque dur. Les numéros de secteurs et de cylindres sont codés ici sous la même forme que pour l'interruption BIOS 13h (disquette/disque dur), les bits 6 et 7 du numéro de secteur représentant les bits 8 et 9 du numéro de cylindre. Cette conformité n'est d'ailleurs pas un hasard car l'interruption 13h et ses fonctions représentent, au moment de cette opération, la seule possibilité d'accès au disque dur. Les fonctions DOS ne sont en effet pas encore disponibles puisque le DOS doit justement encore être lancé.



Des informations supplémentaires dans la structure d'amorçage

Bien que ces informations suffisent amplement pour charger le secteur boot de la partition à lancer, la table de partitions contient quelques informations supplémentaires qui sont précieuses en vue de modifications et extensions ultérieures. La situation du secteur boot est ainsi suivie d'un champ décrivant le type de système d'exploitation utilisé dans la partition. La figure précédente vous montre les différents codes qui peuvent apparaître dans ce champ.

Le secteur de partition indique, outre le secteur de départ, le dernier secteur d'une partition. La situation de ce secteur est elle aussi définie par l'indication des numéros de têtes, de cylindres et de secteurs. Les deux premiers champs d'une entrée de la table indiquent encore le nombre de secteurs que comptent la partition ainsi que la distance, exprimée en secteurs, du secteur boot de la partition au secteur de partition.

En examinant la table de partition, on constatera généralement que la première partition ne se situe pas immédiatement à la suite du secteur de partition mais commence seulement au premier secteur de la piste 0 de la deuxième tête de lecture/écriture. La piste 0 de la première tête de lecture/écriture est donc entièrement inutilisée, à l'exception bien sûr du secteur de partition dans le premier secteur de cette piste. D'autres incohérences sont d'ailleurs à déplorer notamment en ce qui concerne les partitions étendues du DOS.

Structure des partitions étendues sous DOS

DOS 3.3 n'autorise ainsi qu'une seule partition étendue en plus de la partition primaire. Et FDISK dote cette partition étendue elle aussi d'un secteur de partition qui ne contient bien sûr pas de code programme mais comporte, comme il se doit, une table de partitions. Cette table de partitions se compose toujours de deux entrées, la première décrivant la partition étendue dans laquelle figure justement ce secteur de partition, avec comme type de partitions le code 1 ou 4 (partition DOS avec une FAT de 12 ou 16 bits respectivement). La seconde entrée décrit la prochaine partition DOS étendue sur le disque dur, s'il y en a une.

Pour soutenir d'autres partitions étendues, la partition étendue suivante débute à son tour par un secteur de partition tel que nous venons de le décrire. Ainsi se forme une sorte de liste enchaînée qui ne s'achève que lorsque le champ "type de partition" de la seconde entrée de la table d'une partition de cette chaîne contient la valeur 0.

6.8.1. Examen d'une structure de partition

Pour que vous puissiez vous faire une idée de la structure de votre propre disque dur, voici maintenant trois programmes en BASIC, PASCAL et en C qui commentent le contenu du secteur de partition et recherche sur le disque dur les secteurs de partition des partitions étendues (s'il y en a). Les programmes sont prévus pour accéder normalement au premier disque dur avec le numéro 0, mais vous pouvez spécifier un autre numéro (1, 2, 3 etc..) lors de l'appel des programmes pour accéder à d'autres disques durs.

Listing : FIXPARTC.C

```
/*****************************************************************/
/*                        F I X P A R T C . C                    */
/*---------------------------------------------------------------*/
/*   Sujet          : Affiche les partitions d'un disque dur     */
/*---------------------------------------------------------------*/
/*   Auteur         : MICHAEL TISCHER                            */
/*   développé le   : 26.04.1989                                 */
/*   dernière m. à j.: 12.01.1992                                */
/*---------------------------------------------------------------*/
/*   Modèle mémoire : SMALL                                      */
/*---------------------------------------------------------------*/
/*   Appel          : FIXPARTC [ numéro de lecteur]              */
/*                    Le lecteur par défaut est le 0 ("C")       */
/*****************************************************************/

#include <dos.h>
#include <string.h>
#include <stdlib.h>
#include <stdio.h>

/*-- Constantes ---------------------------------------------------*/

#define TRUE  ( 1 == 1 )
#define FALSE ( 1 == 0 )

/*-- Macros -------------------------------------------------------*/

#define HI(x) ( *((BYTE *) (&x)+1) )    /* Retourne HI BYTE d'un WORD */
#define LO(x) ( *((BYTE *) &x) )        /* Retourne LO BYTE d'un WORD */

/*-- Déclarations des types ---------------------------------------*/

typedef unsigned char BYTE;
typedef unsigned int WORD;

typedef struct {                    /* fournit la position d'un secteur  */
               BYTE Tete;                   /* Tête lecture/écriture */
               WORD SecCyl;         /* N° de secteur et de cylindre */
             } SECPOS;

typedef struct {                /* Entrée dans la table des partitions */
               BYTE         Status;          /* état de partition */
               SECPOS       StartSec;          /* premier secteur */
               BYTE         PartTyp;         /* type de partition */
               SECPOS       EndSec;            /* dernier secteur */
               unsigned long SecOfs;    /* Offset du secteur de boot */
               unsigned long NbreSec;          /* Nombre de secteurs */
             } PARTENTRY;

typedef struct {                    /* Fournit le secteur de partition */
               BYTE      BootCode[ 0x1BE ];
               PARTENTRY PartTable[ 4 ];
               WORD      IdCode;                           /* 0xAA55 */
             } PARTSEC;

typedef PARTSEC far *PARSPTR; /* Pointeur sur secteur partit. en mém. */

/*****************************************************************/
/*  ReadPartSec : Copie un secteur de partition du disque dur dans un */
/*                tampon                                           */
/*  Entrée : - Lecteur  : Code BIOS du lecteur (0x80, 0x81 etc.)   */
/*           - Tete     : Numéro de tête de lecture/écriture       */
/*           - SecCyl   : n° de secteur et de cylindre au format BIOS */
/*           - Tamp     : tampon dans lequel le secteur est chargé */
/*  Sortie : TRUE si lecture correcte du secteur sinon FALSE       */
/*****************************************************************/

BYTE ReadPartSec( BYTE Lecteur, BYTE Tete, WORD SecCyl, PARSPTR Tamp )

{
 union REGS  Regs;   /* Registres processeur pour appel interruption */
 struct SREGS SRegs;

 Regs.x.ax = 0x0201;            /* N°de fonction de "Read", 1 secteur */
 Regs.h.dl = Lecteur;                /* Charge les autres paramètres */
 Regs.h.dh = Tete;                         /* dans les registres */
 Regs.x.cx = SecCyl;
 Regs.x.bx = FP_OFF( Tamp );
 SRegs.es  = FP_SEG( Tamp );

 int86x( 0x13, &Regs, &Regs, &SRegs );   /* appel interruption d. dur */
 return !Regs.x.cflag;
}

/*****************************************************************/
/*  GetSecCyl: retourne les numéros de secteur et de cylindre après */
/*             conversion des informations au format du BIOS       */
/*  Entrée : SecCyl   : valeur à décoder                           */
/*           Secteur  : référence à la variable secteur            */
/*           Cylindre : référence à la variable cylindre           */
/*  Sortie : aucune                                                */
/*****************************************************************/

void GetSecCyl( WORD SecCyl, int *Secteur, int *Cylindre )

{
 *Secteur  = SecCyl & 63;                   /* Masquer les bits 6 et 7 */
 *Cylindre = HI(SecCyl) + ( ( (WORD) LO(SecCyl) & 192 ) << 2 );
}

/*****************************************************************/
/*  ShowPartition: affiche les partitions du disque dur           */
/*  Entrée : LC : numéro du lecteur disque dur demandé (0, 1, 2 etc.) */
/*  Sortie : aucune                                               */
/*****************************************************************/

void ShowPartition( BYTE LC )
{
 #define AP ( ParSec.PartTable[ Entry ] )

 BYTE       Tete,                       /* tête de la partition courante */
       Entry;                                 /* compteur d'itérations */
 int        Secteur,                    /* stocker les n° de secteur et  */
       Cylindre;                                    /* de cylindre */
 PARTSEC    ParSec;                /* le secteur courant de la partition */
 union REGS Regs;        /* registres processeur pr appel interruption */

 printf("\n");
 LC |= 0x80;                    /* Prépare le n° de lecteur pour le BIOS */
 if ( ReadPartSec( LC, 0, 1, &ParSec ) ) /* lire secteur de partition */
  {                                      /* Lecture correcte du secteur */
   Regs.h.ah = 8;                /* interroge identification du lecteur */
   Regs.h.dl = LC;
   int86( 0x13, &Regs, &Regs ); /* appel de l'interruption disque dur */
   GetSecCyl( Regs.x.cx, &Secteur, &Cylindre );
   printf( ""
       "\n");
   printf( " Lecteur %2d: %2d têtes avec chacune %4d"
       " cylindres de %3d secteurs          \n",
       LC-0x80, Regs.h.dh+1, Cylindre, Secteur );
   printf( " Table de partitions dans le secteur de partition    "
           "                    \n");
   printf( ""
       "\n");
   printf( "                                    Début     "
       "    Fin      Distance       \n");
   printf( "N°BootType                 Tête Cyl. Sec."
       "Tête Cyl. Sec.BootSectNombre \n");
   printf( ""
       "\n");

       /*-- lire les tables partitions ----------------------------*/
   for ( Entry=0; Entry < 4; ++Entry )
    {
     printf( " %d", Entry );
     if ( AP.Status == 0x80 )                   /* Partition active? */
      printf("Oui ");
     else
      printf ("Non ");
     printf("");
     switch( AP.PartTyp )           /* Evaluation du type de partition */
      {
       case 0x00 : printf( "vide              " );
           break;
       case 0x01 : printf( "DOS, FAT 12 bits  " );
           break;
       case 0x02 :
       case 0x03 : printf( "XENIX             " );
           break;
       case 0x04 : printf( "DOS, FAT 16 bits  " );
           break;
       case 0x05 : printf( "DOS, extended Part." );
           break;
       case 0x06 : printf( "DOS 4.0 > 32 MB   " );
       break;
       case 0x0B : printf( "Concurrent DOS    " );
           break;
       default   : printf( "inconnu   (%3d)   ",
             ParSec.PartTable[ Entry ].PartTyp );
      }

     /*-- Evaluation des données physiques et logiques ------------*/
     GetSecCyl( AP.StartSec.SecCyl, &Secteur, &Cylindre );
     printf( "%2d %5d %3d ", AP.StartSec.Tete, Cylindre, Secteur );
     GetSecCyl( AP.EndSec.SecCyl, &Secteur, &Cylindre );
     printf( "%2d %5d %3d ", AP.EndSec.Tete, Cylindre, Secteur );
```

```
    printf( "%7lu %6lu \n", AP.SecOfs, AP.NbreSec);
  }
  printf( ""
      "\n" );
 }
else
 printf("Erreur d'accès au secteur de boot !\n");
}

/*****************************************************************
*              P R O G R A M M E   P R I N C I P A L             *
*****************************************************************/

int main( int argc, char *argv[] )
{
 int Lecteur;

 printf( "\n FIXPARTC - (c)"
     " 1989, 92 by MICHAEL TISCHER \n" );
 Lecteur = 0;                          /* 1er disque dur par défaut */
 if ( argc == 2 )                      /* demander un autre lecteur ? */
 {                                                    /* oui */
  Lecteur = atoi ( argv[1] );
  if ( Lecteur == 0 && *argv[1] != '0' )
  {
   printf("\nNuméro de lecteur invalide!");
   return( 1 );                        /* Quitter le programme */
  }
 }
 ShowPartition( (BYTE) Lecteur ); /* Afficher le secteur de partition */
 return( 0 );
}
```

Listing : FIXPARTP.PAS

```
{*****************************************************************}
{*                      F I X P A R T P . P A S                  *}
{*---------------------------------------------------------------*}
{*   Sujet          : Affiche les partitions d'un disque dur.    *}
{*---------------------------------------------------------------*}
{*   Auteur         : MICHAEL TISCHER                            *}
{*   développé le   : 26.04.1989                                 *}
{*   dernière m. à j.: 12.01.1992                                *}
{*---------------------------------------------------------------*}
{*   Appel          : FIXPARTP [ numéro de lecteur ]             *}
{*                    Lecteur par défaut : lecteur 0 ("C")       *}
{*****************************************************************}

uses Dos;                                       { Intègre l'unité DOS }

{=== Déclarations de types ====================================}

type  SecPos    = record              { donne la position d'un secteur }
                    Tete : byte;          { tête de lecture/écriture }
                    SectCyl : word;     { n° de cylindre et de secteur }
                  end;

     PartEntry = record         { Entrée dans la table des partitions }
                   Status   : byte;                  { état partition }
                   StartSec : SecPos;              { premier secteur }
                   PartTyp  : byte;                { type de partition }
                   EndSec   : SecPos;              { dernier secteur }
                   SecOfs   : longint;       { Offset du boot secteur }
                   NbreSec  : longint;           { nombre de secteurs }
                 end;

     PartSec   = record              { décrit le secteur de partition }
                   BootCode  : array [0..$1BD] of byte;
                   PartTable : array [1..4] of PartEntry;
                   IdCode    : word;                          { $AA55 }
                 end;

{*****************************************************************}
{*  ReadPartSec : Copie un seceur de partition du disque dur dans un  *}
{*                tampon                                           *}
{*---------------------------------------------------------------**}
{*  Entrée : - Lecteur  : Code BIOS du lecteur ($80, $81 etc.)     *}
{*           - Tete     : Numéro de la tête de lecture/écriture    *}
{*           - SecCyl   : n° de secteur et de cylindre au format BIOS *}
{*           - Tamp     : tampon dans lequel le secteur est chargé  *}
{*****************************************************************}

function ReadPartSec( Lecteur, Tete : byte;
                      SecCyl        : word;
                      var Tamp      : PartSec ) : boolean;

var Regs : Registers;  { registres du processeur pour appel interrupt. }

begin
 Regs.AX := $0201;                  { n° de fonction "Read", 1 secteur }
 Regs.DL := Lecteur;                          { charger les autres    }
 Regs.DH := Tete;                             { paramètres dans les   }
 Regs.CX := SecCyl;                           { registres             }
 Regs.ES := seg( Tamp );
 Regs.BX := ofs( Tamp );
 Intr( $13, Regs);                    { appel interruption disque dur }
 ReadPartSec := ( Regs.Flags and 1 ) = 0;          { drapeau Carry }
end;                                               { signale erreur}

{*****************************************************************}

{*  GetSectCyl: renvoie les n° de secteur et de cylindre après     *}
{*              conversion des codes BIOS du secteur et du cylindre *}
{*---------------------------------------------------------------**}
{*  Entrée : SecCyl   : valeur à décoder                           *}
{*           Secteur  : référence à la variable Secteur            *}
{*           Cylindre : référence à la variable Cylindre           *}
{*****************************************************************}

procedure GetSectCyl( SecCyl : word; var Secteur, Cylindre : integer );

begin
 Secteur  := SecCyl and 63;                  { masque les bits 6 et 7 }
 Cylindre := hi( SecCyl ) + ( lo( SecCyl) and 192 ) shl 2;
end;

{*****************************************************************}
{*  ShowPartition: affiche les partitions du disque dur           *}
{*---------------------------------------------------------------**}
{*  Entrée : LC : n° de lecteur du disque dur demandé (0, 1, 2 etc )  *}
{*****************************************************************}

procedure ShowPartition( LC : byte );

var Tete     : byte;                    { tête de la partition courante }
    SectCyl  : byte;    { Secteur et cylindre de la partition courante }
    ParSec   : PartSec;              { le secteur de partition courant }
    Entry    : byte;                          { compteur d'itérations }
    Secteur,                                { stockent les n° de secteur }
    Cylindre : integer;                                { et de cylindre }
    Regs     : Registers; { registres processeur pour appel interrupt. }

begin
 writeln;
 LC := LC + $80;                      { prépare n° lecteur pour le BIOS }
 if ReadPartSec( LC, 0, 1, ParSec ) then   { lit secteur de partition }
   begin                                     { lecture correcte du secteur }
     Regs.AH := 8;                        { interroge identité du lecteurs }
     Regs.DL := LC;
     Intr( $13, Regs);                    { appel interruption duisque dur }
     GetSectCyl( Regs.CX, Secteur, Cylindre );
     writeln(''+
             '');
     writeln(' Lecteur ', LC-$80, ' : ', Regs.DH+1:2,
             ' têtes avec ', Cylindre:5, ' cylindres de',
             Secteur:3, ' secteurs                ');
     writeln(' Table des partitions dans secteur de partition '+
                                                  ');
     writeln(''+"
             '');
     writeln('                                   Début    '+
             '     Fin     Distance       ');
     writeln('N°BootType                    tête cyl. sec.'+
             'Tête cyl. sec.BootSec.Nombre ');
     writeln(''+
             '');
     for Entry:=1 to 4 do                 { parcourt table des entrées }
       with ParSec.PartTable[ Entry ] do
         begin
         write(' ', Entry, '');
         if Status = $80 then write ('Oui ')
                         else write ('Non ');
         write('');
         case PartTyp of                        { établit type de partition }
           $00       : write('vide            ');
           $01       : write('DOS, FAT 12 bits  ');
```

```
            $02 or $03 : write('XENIX              ');
            $04        : write('DOS, FAT 16 bits   ');
            $05        : write('DOS, extended Part.');
            $06        : write('DOS 4.0 > 32 MB    ');
            $08        : write('Concurrent DOS     ');
            else         write('inconnu (',PartTyp:3,')    ');
          end;
          GetSectCyl( StartSec.SectCyl, Secteur, Cylindre );
          write('', StartSec.Tete:2,' ',Cylindre:5,'  ',Secteur:3 );
          GetSectCyl( EndSec.SectCyl, Secteur, Cylindre );
          write(' ', EndSec.Tete:2,' ',Cylindre:5,'  ',Secteur:3 );
          writeln(' ', SecOfs:8,'', NbreSec:7,'');
        end;
      writeln(''+
              ''@13@10);
    end
  else
    writeln('Erreur en accès au secteur de boot ! ');
end;

{*****************************************************************
*            P R O G R A M M E   P R I N C I P A L              *
*****************************************************************}
```

```
{*****************************************************************)

var Lecteur,                                  { variables de conversion }
    Erreur    : integer;                        { de l'argument passé }

begin
  writeln( @13@10' FIXPARTP - (c)',
           ' 1989, 92 by MICHAEL TISCHER ' );
  Lecteur := 0;                              { 1er disque dur par défaut }
  if ParamCount = 1 then                      { autre lecteur demandé ? }
    begin                                                    { oui }
      val( ParamStr(1), Lecteur, Erreur );       { Ascii vers décimal }
      if Erreur <> 0 then                   { erreur de conversion ? }
        begin                                                { oui }
          writeln(@13@10'Numéro de lecteur invalide!');
          exit;                                  { quitter programme }
        end;
    end;
  ShowPartition( Lecteur );          { afficher le secteur de partition }
end.
```

Listing : FIXPARTB.BAS

```
'*****************************************************************
'*                          FIXPARTB                             *
'*---------------------------------------------------------------*
'*  Objet          : Afficher les partitions d'un disque dur     *
'*---------------------------------------------------------------*
'*  Auteur         : Michael Tischer                             *
'*  Développé le   : 21.05.1991                                  *
'*  Dernière Modification : 12.01.1992                           *
'*---------------------------------------------------------------*
'*  Appel          : FIXPARTB [numéro de lecteur]                *
'*                   Lecteur par défaut : 0 ("C")                *
'*****************************************************************
'
DECLARE SUB GetSecCyl (SecCyl AS LONG, Sect AS INTEGER, Cyl AS INTEGER)
DECLARE FUNCTION ReadPartSec% (LW%, Tete%, SecCyl%, PartEntree() AS ANY)
DECLARE SUB Showpartition (LW AS INTEGER)
DECLARE FUNCTION MLNG& (Nombre AS INTEGER)

'$INCLUDE: 'QB.BI'      'Inclure fichier Include pr appels interruptions

CONST TRUE = -1                                          'Définir "VRAI"
CONST FALSE = NOT TRUE

TYPE SecPos                                 'décrit la position d'un secteur
     Tete AS INTEGER                             'tête de lecture/écriture
     SecCyl AS LONG                        'Numéro de Secteur et Cylindre
END TYPE

TYPE PartEntry                            'Entrée n de la table de partition
     Status AS INTEGER                              'Etat de la partition
     StartSec AS SecPos                                 'Premier secteur
     PartType AS INTEGER                              'Type de partition
     EndSec AS SecPos                                    'Dernier secteur
     SecOfs AS LONG                          'Décalage du Boot secteur
     NbreSec AS LONG                                 'Nombre de secteurs
END TYPE

DIM Lecteur AS INTEGER      'Variable de conversion des arguments passés

CLS
PRINT " FIXPARTB - (C)";
PRINT " 1991, 92 by Michael Tischer "
Lecteur = 0                                  'Premier disque dur par défaut
IF COMMAND$ <> "" THEN                         'L'autre lecteur est donné
  Lecteur = VAL(COMMAND$)                              'ASCII vers décimal
END IF
CALL Showpartition(Lecteur)                  'Affiche secteur de la partition
END

'*****************************************************************
'* GetSecCyl : renvoie les numéros de secteur et de cylindre obtenus à *
'*             partir du codage utilisé par le BIOS              *
'* Entrees   : Variable de secteur en codage BIOS                *
'* Sortie    : Variables secteur et cylindre par référence de variable *
'*****************************************************************
SUB GetSecCyl (SecCyl AS LONG, Sect AS INTEGER, Cyl AS INTEGER)

 Sect = SecCyl AND 63                                 'Masque bits 6 et 7
 Cyl = SecCyl \ 256 + ((SecCyl MOD 256) AND 192) * 4
END SUB
```

```
'
'*****************************************************************
'* MLNG    : Convertit un entier en son équivalent long          *
'* Entree  : Un entier                                           *
'* Sortie  : Le "long" correspondant au contenu du registre      *
'*****************************************************************
FUNCTION MLNG& (Nombre AS INTEGER)
'
IF Nombre >= 0 THEN                              'Le bit 16 n'est pas mis
  MLNG = Nombre              'Entier correspondant au contenu du registre
ELSE                                             'Le bit 16 est mis
  MLNG = 65536 + Nombre      'ce nombre correspond au contenu du registre
END IF
'
END FUNCTION
'
'*****************************************************************
'* ReadPartSec : Lit le secteur de partition d'un disque dur     *
'* Entrees     : LW          = Code BIOS du lecteur              *
'*               Tete        = Numéro de la tête de lecture/écriture *
'*               SecCyl      = Numéro secteur et cylindre format BIOS *
'* Sorties     : PartEntry() = Les 4 tables de part. du sect. de part. *
'*****************************************************************
'
FUNCTION ReadPartSec% (LW%, Tete%, SecCyl%, PartEntr() AS PartEntry)
'
DIM Register AS RegTypeX  'Registre processeur pour appel d'interruption
DIM PartSecteur AS STRING * 512          'stocke le secteur de partition
DIM IdCode AS LONG          'stocke le code d'identification de partition
DIM i AS INTEGER                          'compteur d'itérations de boucle
DIM j AS INTEGER                          'compteur d'itérations de boucle
'
Register.ax = &H201       'N° de fonct. pour "READ" dans premier secteur
Register.cx = SecCyl%      'Secteur et Cylindre de la partition courante
Register.dx = Tete% * 256 + LW%              'N° de tête et de lecteur
Register.es = VARSEG(PartSecteur)  'adresse Seg. du secteur de partition
Register.bx = VARPTR(PartSecteur) 'adresse Offset du seteur de partition
CALL INTERRUPTX(&H13, Register, Register)    'Appel d'int. de disque dur
DEF SEG = VARSEG(PartSecteur)     'Définit adresse seg. instruciton PEEK
Pointer = VARPTR(PartSecteur) + &H1BE      'Ptr sur 1ere entrée de part.
offset = 0                      'Décalage de la première entrée de partition
FOR i = 1 TO 4                          'Lire les quatre entrées de la partition
  PartEntr(i).Status = PEEK(Pointer + offset + 1)
  PartEntr(i).StartSec.Tete = PEEK(Pointer + offset + 2)
  PartEntr(i).StartSec.SecCyl = PEEK(Pointer + offset + 4) * 256& +
 PEEK(Pointer + offset + 3)
  PartEntr(i).PartType = PEEK(Pointer + offset + 5)
  PartEntr(i).EndSec.Tete = PEEK(Pointer + offset + 6)
  PartEntr(i).EndSec.SecCyl = PEEK(Pointer + offset + 8) * 256& +
 PEEK(Pointer + offset + 7)
  PartEntr(i).SecOfs = 0
  FOR j = 0 TO 3
    PartEntr(i).SecOfs = PartEntr(i).SecOfs * 256& + PEEK(Pointer + off-
 set + 12 - j)
  NEXT
  PartEntr(i).NbreSec = 0
  FOR j = 0 TO 3
    PartEntr(i).NbreSec = PartEntr(i).NbreSec * 256& + PEEK(Pointer +
 offset + 16 - j)
```

```
    NEXT
    offset = offset + 16      'Cale Offset sur entrée suivante de partition
  NEXT
  IdCode = PEEK(Pointer + offset + 2) * 256& + PEEK(Pointer + offset + 1)
  IF (Register.flags AND 1) = 0 THEN
    ReadPartSec = TRUE
  END IF

END FUNCTION

'*********************************************************************
'* ShowPartition : Affiche le partitionnement du disque dur          *
'* Entree        : Numéro du disque dur demandé                      *
'* Sortie        : aucune                                            *
'*********************************************************************
SUB Showpartition (LW AS INTEGER)

DIM Tete AS INTEGER                       'Tete de la partition courante
DIM SecCyl AS INTEGER      'Secteur et cylindre de la partition courante
DIM Entry AS INTEGER                    'Compteur d'itérations de boucle
DIM Sect AS INTEGER                                  'Numéro de secteur
DIM Cyl AS INTEGER                     'Numéro de cylindre de l'entrée
DIM PartEntree(4) AS PartEntry                'courante de la partition
DIM Register AS RegTypeX     'Registre processeur pour appel interception

PRINT :
LW = LW + &H80                'Prépare le numéro du lecteur pour le BIOS
IF ReadPartSec(LW, 0, 1, PartEntree()) THEN 'Lit le secteur de partition
  Register.ax = &H0800          'AH = n° fonction, demande identif. lecteur
  Register.dx = LW                                          'N° lecteur
  CALL INTERRUPTX(&H13, Register, Register)     'Appelle Int. disque dur
  CALL GetSecCyl(MLNG(Register.cx), Sect, Cyl)   'N° secteur et cylindre
  PRINT "";
  PRINT ""
  PRINT USING " Lecteur #  : ## têtes avec "; LW - &H80; Register.dx \ 256
 + 1;
  PRINT USING " chacune ####  cylindres de "; Cyl;
  PRINT USING "### secteurs                 "; Sect
  PRINT " Table de partitions dans secteur de partition     ";
  PRINT "                                   "
  PRINT "";
  PRINT ""
  PRINT "                                  Début     ";
  PRINT "     Fin      Distance             "
  PRINT "N°Boot Type                Tête ";
  PRINT "Cyl. Sec.Tete Cyl. Sec.BootSect.  Nombre "
  PRINT "";
  PRINT ""
  FOR Entry = 1 TO 4                                  'Les entrées défilent
    PRINT USING "##"; Entry;
    IF PartEntree(Entry).Status = &H80 THEN
      PRINT "Oui ";
    ELSE
      PRINT "Non ";
    END IF
    SELECT CASE PartEntree(Entry).PartType
      CASE 0
    PRINT "Libre              ";
      CASE 1
    PRINT "DOS, 12-Bit-FAT    ";
      CASE 2 OR 3
    PRINT "Xenix              ";
      CASE 4
    PRINT "DOS, 16-Bit-FAT    ";
      CASE 5
    PRINT "DOS, extended Part. ";
      CASE 6
    PRINT "DOS 4.0 > 32 MB    ";
      CASE &HDB
    PRINT "concurrent DOS,    ";
      CASE ELSE
    PRINT USING "Inconnu      (###)  "; PartEntree(Entry).PartType
    END SELECT
    CALL GetSecCyl(PartEntree(Entry).StartSec.SecCyl, Sect, Cyl)
    PRINT USING "###  ###  "; PartEntree(Entry).StartSec.Tete; Cyl;
    PRINT USING "### "; Sect;
    CALL GetSecCyl(PartEntree(Entry).EndSec.SecCyl, Sect, Cyl)
    PRINT USING "###  ###  ### "; PartEntree(Entry).EndSec.Tete; Cyl;
Sect;

    PRINT USING "  #######  "; PartEntree(Entry).SecOfs;
    PRINT USING "  #######  "; PartEntree(Entry).NbreSec
  NEXT
  PRINT "";
  PRINT ""
ELSE
  PRINT "Erreur d'accès au secteur de boot"
END IF

END SUB
```

7. Interface parallèle

Qu'est-ce qu'un PC sans imprimante ? Un poisson sans vélo, un livre sans caractères ou une voiture sans roues ? Je n'en sais rien . Mais en tant que programmeur, on finit toujours par être obligé d'imprimer des informations sur papier. Pour arriver à cette fin, il existe trois moyens : la programmation directe du matériel, le passage par le BIOS ou l'exploitation des différentes fonctions de DOS

Ce chapitre est consacré à la programmation directe et aux fonctions du BIOS qui permettent d'accéder à l'interface parallèle. Comme on le verra, les fonctions du BIOS couvrent toutes les tâches que nécessite la communication avec une imprimante. Il n'y a donc aucune raison de programmer directement le matériel, la vitesse ne jouant aucun rôle dans ce cas. Les fonctions du BIOS permettent d'émettre plus de 10000 caractères par seconde : à ce rythme-là l'imprimante ne peut de toute façon pas suivre.

Par ailleurs les fonctions du BIOS présentent un grand avantage par rapport à leurs homologues de DOS : elles contrôlent mieux l'état de l'imprimante. Lorqu'une impression échoue, DOS essaye d'arrêter toute l'exécution en cours en émettant une interruption d'erreur critique. Nous verrons dans la première section qu'avec les fonctions du BIOS les choses ne se passent pas aussi brutalement.

La deuxième section montre qu'il existe encore des circonstances où il est avantageux de programmer directement l'interface parallèle. Nous étudierons la liaison entre deux ordinateurs par le moyen de câbles dits null-modems qui permettent des transmissions de données ultra-rapides. Des logiciels comme LapLink ont montré les avantages de cette technique qui sera appliquée ici à un programme complet de transmission de fichiers. Vous apprendrez à cette occasion toute une série d'informations à propos de l'interface parallèle souvent dénommée Centronics.

7.1. Accès à l'imprimante par le BIOS

Parmi les différentes interruptions que le BIOS se réserve pour ses services, l'interruption 17h est exclusivement consacrée à la communication avec l'interface parallèle. En principe il est possible de brancher sur cette interface des périphériques autres qu'une imprimante. Mais l'interruption 17h est couramment appelée "interruption imprimante du BIOS" parce qu'en pratique on ne trouve jamais que des imprimantes ainsi connectées. Les termes d'"imprimante" et d'"interface parallèle" sont donc interchangeables.

Les fonctions de l'interruption imprimante du BIOS

Comme le confirmera la section consacrée à la programmation directe du matériel, le nombre d'interface parallèles gérables sur un PC est au maximum de trois. Elles sont commandées par trois fonctions de l'interruption 17h du BIOS. Ces trois fonctions ne correspondent absolument pas aux trois interfaces éventuellement connectées. Elles ont en charge des rôles différents et chacune d'elle peut agir sur l'une des trois interfaces possibles.

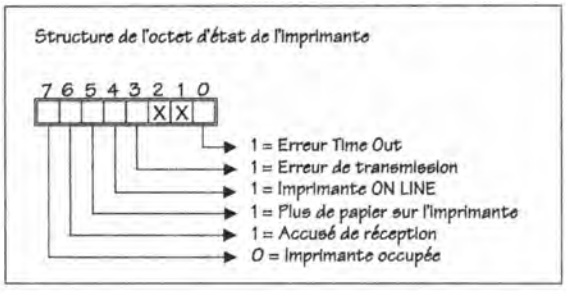
Rôle des trois fonctions de l'interruption imprimante du BIOS

Fonction	Rôle
00h	Envoie un caractère
01h	Initialise l'imprimante
02h	Lit l'état de l'imprimante

Notez que les fonctions équivalentes de DOS ne s'appliquent qu'à la première interface parallèle, appelée "PRN" ou "LPT1". Les trois fonctions du BIOS sont plus souples : on leur communique par le registre DX le numéro de l'interface adressée. On peut donc y charger les nombres 0, 1 ou 2 selon que l'on s'adresse à l'une ou l'autre des interfaces. En DOS, le numéro 0 correspondrait à LPT1, le numéro 1 à LPT2, le numéro 2 à LPT3.

L'état de l'imprimante

Toutes les trois fonctions renvoient dans le registre AH l'indicateur d'état de l'imprimante. Les différents bits de cet indicateur (qui occupe un octet) fournissent diverses informations sur l'activité de l'imprimante, la présence de papier, la détection des erreurs de transmission, etc.. L'indicateur d'état de l'imprimante joue un rôle important dans la communication avec l'interface parallèle.



L'erreur de Time Out ou dépassement du temps imparti se produit lorsque que le BIOS essaye de transmettre des données à l'imprimante, alors que celle-ci les refuse ou bien annonce qu'elle est encore en action (-BUSY = le bit 7 vaut 0). Il est très fréquent que l'imprimante ne puisse plus accepter de caractères : n'oubliez pas qu'en théorie une interface parallèle peut émettre jusqu'à 100000 caractères/s. Aucune imprimante, si perfectionnée soit-elle, ne peut tenir cette cadence, même avec un buffer interne.

Le nombre de tentatives d'émissions que le BIOS effectuera avant de signaler une erreur de Time Out dépend du contenu d'une variable du BIOS. L'adresse stratégique est 0040:0078: à cet endroit, sur trois octets, sont stockés les nombres de tentatives associés à chacune des trois imprimantes soutenues par le BIOS.

Compteurs de Time Out gérés par le BIOS pour les trois interfaces parallèles	
Adresse	**Signification**
0040:0078	Compteur de Time-Out pour la première interface parallèle
0040:0079	Compteur de Time-Out pour la seconde interface parallèle
0040:007A	Compteur de Time-Out pour la troisième interface parallèle

Les nombres mémorisés ne sont pas des durées mais représentent le nombre d'échecs qui déterminent une erreur de Time Out. Il s'agit en fait de compteurs d'itérations dans une boucle. Le programme du BIOS effectue un test dans le cadre d'une boucle. La boucle en question ne comporte que quelques instructions machine. Elle est parcourue en l'espace de quelques microsecondes, c'est pourquoi le nombre mémorisé n'est pas à proprement parler le nombre d'itérations mais une base multipliée par le facteur 262140 (4 * 65535). Ainsi la valeur 20 mémorisée à l'initialisation du système exprime que l'erreur de Time Out ne se déclenchera qu'au bout de plus de 5 millions de tentatives d'émissions infructueuses.

L'utilisation d'un compteur d'itérations en lieu et place d'un compteur de temps entraîne une conséquence notable : le délai d'apparition de l'erreur de Time Out dépend de la vitesse de fonctionnement de l'ordinateur, c'est-à-dire du type de processeur et de sa cadence. Dans le cas d'un ordinateur très rapide, le compteur doit être augmenté en conséquence. Lorsqu'on achète un 486, on risque , faute de cet ajustement, de se trouver devant des erreurs de Time Out au moment d'une demande d'impression, alors que le bon vieil AT marchait correctement.

Les fabricants de BIOS effectuent généralement ledit ajustement en modifiant le facteur de multiplication rencontré précédemment, par exemple en remplaçant 4*65536 par 8*65536 lorsque le système est deux fois plus rapide qu'AT classique.

Cette intervention est tout à fait justifiée sous cette forme car les programmes d'application accèdent souvent aux trois variables du BIOS pour changer le Time Out de l'une des interfaces. Un programmeur peut librement augmenter un Time Out lorsque

l'imprimante branchée le requiert. Cette augmentation serait rapidement absorbée par un système rapide si le facteur de multilication n'était pas remanié en conséquence.

Les autres composants de l'indicateur d'état

Le bit de Time Out n'est pas le seul intéressant dans l'indicateur d'état. Le bit 3 signale une erreur de transmission, c'est-à-dire des données incohérentes sur la ligne. Le bit 4 indique si l'imprimante est en ligne (ON LINE) ou au contraire hors ligne (OFF LINE) : il correspond au commutateur de même nom situé sur le panneau avant de l'imprimante avec une diode témoin.

Le bit 5 détecte si l'imprimante est alimentée en papier. Le bit 6 est un accusé de réception (Acknowledge flag) par lequel l'imprimante signale la réception du dernier caractère. Pour savoir si une imprimante est connectée à une interface données, il faut lire ce bit : s'il est égal à 1, aucune imprimante n'est branchée.

Le bit 7 rend compte du signal d'activité dit -BUSY, par lequel l'imprimante exprime qu'elle est occupée et qu'elle ne peut plus accepter de caractère pour le moment. Ce bit joue un rôle en liaison avec l'erreur de Time Out : c'est lorsqu'il est actif que la boucle d'émission d'un caractère est recommencée.

Notez que la logique du bit -BUSY est inversée. Il est égal à 0 lorsque l'imprimante est en action, et à 1 lorsque l'imprimante est prête à recevoir des caractères. Les différentes situations dans lesquels se trouvent une imprimante peuvent affecter toute une série de bits de l'indicateur d'état. Par exemple lorsque l'imprimante est prête et en ligne, les bits 4 et 7 sont à 1. Si on met l'imprimante hors ligne, par exemple pour faire avancer le papier, les bits 4 et 7 passent à 0 cependant que le bit 3 signale une erreur de transmission.

Exploitation de l'indicateur d'état

A quoi sert concrètement l'indicateur d'état dans un programme ? D'abord à connaître les circonstances de transmission d'un caractère. Vous pouvez tester s'il reste du papier, si l'imprimante est en ligne et s'il y a bien une imprimante rattachée à la connexion.

Vous pouvez aussi vérifier globalement que les conditions de réception d'un caractère sont remplies. Tel n'est pas le cas si l'un des bits 1 (erreur de Time Out), 3 (Erreur de transmission) ou 5 (Défaut de papier) est armé ou si l'un des bits 4 (imprimante en ligne) ou 7 (imprimante en action) est nul. On peut formuler cette condition dans une sorte de pseudo-code :

```
pstatus = EtatImprimante;
if ( ( (pstatus and 29h) <> 0 ) or
     ( (pstatus and 80h) = 0  ) or
```

```
    ( (pstatus and 10h) = 0  ) ) then
  ImprimanteOK = FALSE
else
  ImprimanteOK = TRUE;
```

Les trois fonctions du BIOS

Mais revenons aux trois fonctions du BIOS qui permettent d'exploiter l'interruption 17h. La première d'entre elles, qui porte le numéro 0, sert à transmettre un caractère à l'imprimante.Il faut lui communiquer son numéro en AH et le caractère à transmettre (ASCII) en AL. Au retour, le registre AH contient l'indicateur d'état, comme ce sera d'ailleurs le cas pour toutes les autres fonctions.

La deuxième fonction sert à initialiser l'interface parallèle et l'imprimante. Elle est à invoquer chaque fois que l'on envoie pour la première fois des données à l'imprimante. Elle signifie en quelque sorte :"Attention, il faut remettre les compteurs à zéro, une nouvelle impression va démarrer". Cette fonction ne nécessite aucun argument, à part évidemment le numéro 01h à mettre en AH.

La dernière fonction qui porte le numéro 02h est spécialisée dans la lecture de l'indicateur d'état qu'elle renvoie en AH. Elle ne procède à aucune émission de caractère et n'entreprend aucune initialisation, comme c'est le cas des fonctions 00h et 01h. Comme argument, elle ne nécessite que la donnée de son numéro.

7.1.1. Appel des fonctions du BIOS

Les fonctions qui viennent d'être présentées sont trop simples pour justifier des programmes de démonstration. Chacune d'elles peut être appelée à partir des langages évolués par le biais des instructions ou fonctions d'interruption classiques.

Par ailleurs les bibliothèques des compilateurs C offrent souvent des instructions ou fonctions spécialisées. Le tableau suivant montre quelles sont les routines concernées dans le cas des compilateurs Microsoft et Borland :

Routines d'appel des fonctions d'impression chez Borland et Microsoft

Compilateur	Fonction
Turbo C	
Borland C et C++	biosprint
Microsoft C	
Quick C	_bios_printer

QuickBasic et Turbo Pascal ont aussi des instructions d'impression de bas niveau mais ces dernières n'exploitent pas les fonctions du BIOS ci-dessus. Elles se servent des fonctions de sorties de DOS, et sont donc grevées de tous leurs défauts : en cas d'erreur de transmission elles déclenchent ainsi l'interruption d'erreur critique 24h au lieu d'envoyer un code d'erreur au programme. Il est vrai qu'il est possible d'intercepter cette interruption comme le fait automatiquement Turbo Pascal. L'ordre d'impression dans QuickBasic est LPRINT. Turbo Pascal utilise WRITE et WRITELN, mais il faut auparavant créer une variable fichier, l'associer à l'imprimante et la mentionner à chaque appel de WRITE ou WRITELN, à moins que l'unité PRINTER.TPU ne vous décharge de cette tâche. Pour plus de précisions, reportez-vous aux manules de Turbo Pascal.

7.1.2. Détournement de l'interruption imprimante du BIOS.

L'interruption associée à l'imprimante n'est pas seulement un instrument aux mains des logiciels d'application, elle est également très appréciée des programmes résidents qui ont l'habitude de la détourner pour modifier son fonctionnement. C'est ainsi que sont implémentés les fameux Print-Spoolers qui commencent par stocker dans un buffer les caractères destinés à l'imprimante avant de les transférer à l'impression.

Je voudrais vous présenter un programme écrit en assembleur qui sera utile à tous ceux qui ont une imprimante munie d'un jeu de caractères différent de celui du PC. Il existe notamment certaines imprimantes Epson dont les caractères accentués (é,ê,è) ne se trouvent pas au même endroit que dans le jeu ASCII habituel du PC. Le programme va détourner l'interruption de l'imprimante pour y insérer la transformation des caractères litigieux. On évite ainsi de définir une table de conversion comme le font généralement les traitements de texte.

Conversion automatique de caractères

Le nouveau gestionnaire d'interruption qui se trouve au centre du programme résident PRACCENT.ASM commence par vérifier si la fonction 00h d'impression d'un caractère a été appelée. Seul le fonctionnement de cette dernière est à modifier. Si c'est une autre fonction qui a été invoquée, l'appel est transmis à l'ancien gestionnaire d'interruption.

Si un caractère doit être imprimé, le programme le recherche dans la table CODETAB. Cette table est située au début du listing de PRACCENT.ASM et comporte des éléments à deux octets : le premier octet (de poids faible) est le code du caractère converti, l'octet suivant étant le code du caractère à convertir. Le table se termine par un octet nul.

La nouvelle fonction 00h de l'interruption imprimante du BIOS commence par chercher si le caractère à imprimer se trouve dans une ligne de la table, en deuxième position. Si cette recherche échoue, le caractère est renvoyé sur l'ancienne interruption. Si le caractère est repéré dans la table, il est remplacé par le premier octet de la ligne avant d'être envoyé à l'impression.

Le reste du programme ne réclame pas beaucoup de commentaires : sa structure est typique des nombreaux programmes résidents décrits dans ce livre. PRACCENT est réalisé sous forme de programme COM : son appel se fait sans paramètre. Lorsqu'on le lance, il commence par vérifier s'il est déjà installé. Si tel n'est pas le cas, il s'installe, sinon il se retire de la mémoire.

Ce programme ne sert pas seulement à gérer les impressions commandées par l'interruption 17h du BIOS. N'oubliez pas en effet que même les fonctions d'impression du DOS recourent à l'interruption du BIOS. Donc les impressions effectuées par DOS sont également converties par PRACCENT.

Listing : PRACCENT.ASM

```
;*****************************************************************;
;*                          P R A C C E N T                      *;
;*--------------------------------------------------------------*;
;*   Fonction       : Redirige l'interruption imprimante du BIOS sur *;
;*                    une routine utilisateur permettant ainsi, par *;
;*                    exemple, la conversion d'ASCII-IBM en EPSON.  *;
;*                    Tous les deux appels, le programme est à      *;
;*                    nouveau désactivé et éliminé de la mémoire.   *;
;*                                                                  *;
;*--------------------------------------------------------------*;
;*   Auteur         : MICHAEL TISCHER                             *;
;*   développé le   : 02/08/1987                                  *;
;*   Dernière modif.: 01/12/1991                                  *;
;*--------------------------------------------------------------*;
;*   Assemblage     : MASM PRACCENT;                              *;
;*                    LINK PRACCENT;                              *;
;*                    EXE2BIN PRACCENT PRACCENT.COM;              *;
;*--------------------------------------------------------------*;
;*   Appel          : PRACCENT                                    *;
;*****************************************************************;

;== Ici commence le programme véritable ========================

code    segment para 'CODE'             ;Définition du segment de CODE

        org 100h

        assume cs:code, ds:code, es:code, ss:code

start:  jmp praccentini                 ;La première instruction exécutée

;== Données (restent en mémoire) ==============================

ancint    equ this dword                ;Ancien vecteur d'interruption 17h
ofsancint dw (?)                        ;Adresse d'offset vecteur d'interruption 17h
segancint dw (?)                        ;Adresse de segment vecteur d'interr. 17h
;-- La table suivante présente chaque fois le nouveau ------------
;-- code suivi de l'ancien ---------------------------------------

codetab db  64,133                      ; 'à'  ----------> '@'
        db 125,138                      ; 'è'  ----------> '}'
        db 123,130                      ; 'é'  ----------> '{'
        db  91,248                      ; '°'  ----------> '['
        db 124,151                      ; 'ù'  ----------> '|'
        db  92,135                      ; 'ç'  ----------> '\'
        db  93,21                       ; '§'  ----------> ']'
        db  0                                   ;Fin de la table

;== Voici la nouvelle interruption imprimante (reste en mémoire) ==

newpri  proc far

        jmp  short newpri_1

        db "PR"                         ;Identification du programme

newpri_1: or   ah,ah                    ;Imprimer caractère (Fonction 0) ?
        jne  aint                       ;NON --> Appeler ancienne interruption

        pushf                           ;Tous les registres modifiés par la
        push bx                         ;suite doivent être sauvegardés
        push si
        push ds

        push cs                                 ;Ranger CS sur la pile
        pop  ds                                 ;Retirer de la pile pour DS

;-- Le code doit-il être converti ? ------------------------------

        cld                             ;Augmenter comptage pour instr. de chaîne
        mov  si,offset codetab          ;Adresse de la table de codes
        mov  bl,al                              ;Ranger code dans BL
testcode: lodsw                         ;Charger ancien (AH) et nouveau (AL) codes
        or   al,al                              ;Fin de table atteinte ?
        je   notfound                           ;OUI --> code non trouvé
        cmp  ah,bl                              ;Est-ce le code à convertir
        jne  testcode                   ;NON --> poursuivre examen de la table
        jmp  short convertir            ;C'était un code à convertir

notfound: mov  al,bl                    ;Ramener ancien code dans AL
convertir: xor  ah,ah                   ;Fixer à nouveau numéro de fonction 0
        pop  ds                         ;Retirer registres sauvegardés
        pop  si
        pop  bx
        popf

aint:   jmp  cs:[ancint]                ;Vers l'ancienne routine imprimante

newpri  endp

instend equ this byte                   ;Jusqu'à cette cellule de mémoire
                                        ;tout doit rester résident

;== Données (peuvent être effacées par le DOS) ================

installm db 13,10,"PRACCENT (c) 1987 by Michael Tischer",13,10,13,10
        db "PRACCENT a été installé et peut à nouveau être",13,10
        db "désactivé par un second appel",13,10,"$"

elimine db "PRACCENT a été désactivé$",13,10

;== Programme (peut être effacé par le DOS) ====================
;-- Routine de lancement et initialisation -----------------------

praccentini  label near

        mov  ax,3517h           ;Lire contenu vecteur d'interruption 17h
        int  21h                                ;Appeler fonction DOS
        cmp  word ptr es:[bx+2],"RP"    ;Tester si programme PRACCENT
        jne  install                    ;Pas de PRACCENT installé --> INSTALL

;-- PRACCENT est désactivé ---------------------------------------

        mov  dx,es:ofsancint    ;Adresse d'offset de l'interruption 17h
        mov  ax,es:segancint ;Adresse de segment de l'interruption 17h
        mov  ds,ax                                      ;dans DS
        mov  ax,2517h       ;Replacer contenu vecteur d'interruption 17h
        int  21h                                ;sur ancienne routine

        mov  bx,es              ;Ranger adresse de segment du programme
        mov  es,es:[2Ch]  ;Retirer du PSP adr. seg. de l'environnement
        mov  ah,49h                             ;Libérer à nouveau mémoire
        int  21h                                ;d'ancien environnent

        mov  es,bx                              ;Libérer à nouveau la mémoire
        mov  ah,49h                             ;de l'ancien PRACCENT
```

```
        int  21h                    ;à travers la fonction 49h du DOS

        push cs                                 ;Sauver CS sur la pile
        pop  ds                                     ;Ramener dans DS

        mov  dx,offset elimine          ;Message : programme éliminé
        mov  ah,9                  ;Numéro de fonction pour Ecrire chaîne
        int  21h                              ;Appeler fonction DOS

        mov  ax,4C00h                    ;Terminer programme correctement
        int  21h                   ;Appeler fonction pour terminer progr.

;-- Installation de PRACCENT ----------------------------------------

install  label near

        mov  ax,3517h       ;Lire contenu du vecteur d'interruption 17h
        int  21h                                      ;Appeler fonction DOS
        mov  segancint,es       ;Ranger adresses de segment et d'offset
        mov  ofsancint,bx           ;Ranger vecteur d'interruption 17h

        mov  dx,offset newpri     ;Adresse offset new routine d'interr.
        mov  ax,2517h             ;Contenu du vecteur d'interruption 17h
        int  21h                                  ;sur routine utilisateur

        mov  dx,offset installm          ;Message : programme installé
        mov  ah,9                  ;Numéro de fonction pour Sortir chaîne
        int  21h                                      ;Appeler fonction DOS

          ;-- Seuls le PSP, la nouvelle routine d'interruption et les
          ;-- données correspondantes doivent rester résidentes.

        mov  dx,offset instend ;Calculer nombre de paragraphes (unités
        mov  cl,4         ;de 16 octets) dont doit disposer le programme
        shr  dx,cl
        inc  dx
        mov  ax,3100h       ;Terminer programme par code de fin 0 (o.k)
        int  21h                                      ;mais rester résident

;-- Fin ---------------------------------------------------------------

code    ends                                          ;Fin du segment de CODE
        end  start
```

7.2. Programmation directe de l'interface parallèle

Tant que le récepteur est capable de suivre l'émetteur, les fonctions du BIOS qui permettent l'émission des caractères sur l'interface parallèle sont parfaites. Mais si un ordinateur communique, non pas avec une imprimante, mais avec son semblable, les choses deviennent plus délicates. Car les vitesses de transfert exigées sont au-delà des capacités des fonctions du BIOS. Par ailleurs, une liaison entre deux ordinateurs nécessite un câble spécial (dit câble null-modem). Il faut alors changer un peu les règles du jeu.

7.2.1. Les ports d'entrée-sortie des interfaces parallèles

Il est possible de brancher jusqu'à 3 interfaces parallèles sur un PC. Des espaces d'adressage correspondants sont soigneusement réservés à cet usage :

Port	Interface
3BCh-3BFh	interface parallèle sur la carte MDA
378h-37Fh	1ère interface parallèle
278h-27Fh	2ème interface parallèle

Dans le tableau précédent les adresses des ports ne se présentent pas dans l'ordre croissant. Tel est en effet l'ordre dans lequel le BIOS recherche les interfaces au démarrage du système. Sa premier tâche est en effet de tester la présence-même des interfaces. L'ordre de découverte fixe alors l'attribution des dénominations LPT1, LPT2, LPT3.

Le BIOS commence par s'intéresser à la zone mémoire 3BCh-3BFh. Elle fait partie d'un espace allant de 3B0h à 3BFh et réservé à la carte monochrome MDA ou la carte

graphique Hercules. Jusqu'au milieu des années 80, la plupart des PC étaient livrés avec des cartes de ce type qui possédaient, en plus des circuits d'affichage, une interface parallèle.

Si une carte de ce type est décelée, le BIOS la baptise LPT1. La prochaine sera alors enregistrée comme LPT2. S'il n'y a pas de carte d'écran avec interface parallèle, la prochaine interface parallèle découverte sera considérée comme LPT1.

Les deux autres zones de mémoire sont réservées à des interfaces parallèles indépendantes. Compte tenu des progrès incessants de la miniaturisation, il n'est pas sûr que ces deux interfaces soient situées sur des cartes distinctes. On les trouve non seulement sur la carte mère mais même à l'intérieur du circuit du processeur. Cette tendance s'affirme encore avec les ordinateurs portables Notebooks et Palmtops.

BIOS interroge donc les ports d'interface pour connaître leur emplacement indépendamment de leur implémentation effective. Il peut en résulter des conséquences inattendues. Si une seule interface parallèle est installée et si elle occupe l'espace mémoire destiné à la deuxième, elle sera malgré tout adressée comme LPT1.

Sélection de LPT1 à LPT3

L'attribution des dénominations LPT1, LPT2 et LPT3 dépend de l'enregistrement de leurs adresses de base dans le segment des variables du BIOS. A l'offset 0008h se trouve une table de quatre mots qui mémorisent les adresses des ports des interfaces parallèmes.

0040:0008h	adresse de base LPT1
0040:000Ah	adresse de base LPT2
0040:000Ch	adresse de base LPT3
0040:000Eh	adresse de base LPT4

Bien qu'au moment de l'initialisation du système le BIOS ne recherche que trois interfaces au maximum, il y a apparemment de la place pour quatre. L'expérience montre effectivement que les fonctions du BIOS peuvent gérer une quatrième interface parallèle si on reporte son adresse de base à l'offset 000Eh et si dans les appels on lui attribue le numéro 3.

La terminologie LPT1 ne provient pas du BIOS qui se contente de numéroter les interfaces de 0à 3. Elle a été introduite par DOS qui désigne les périphériques parallèles par LP1, LPT2 et LPT3. Notez que LPT4 n'existe pas du point de vue de DOS.

Pour échanger deux interfaces, par exemple LPT1 et LPT2, il suffit de permuter les adresses des ports mémorisées par les variables du BIOS. En pseudo code; l'affaire se traite de la façon suivante :

```
DummyWord = MEM[ 0040h: 0008h ]
MEM[ 0040h: 0008h ] = MEM[ 0040h: 000Ah ]
MEM[ 0040h: 000Ah ] = DummyWord
```

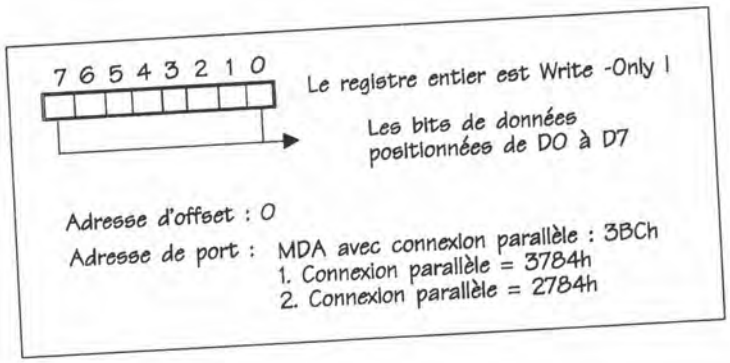
7.2.2. Les registres de l'interface

Indépendamment de leur adressage, toutes les interfaces parallèles présentent trois registres situés au début de leur zone de mémoire : par exemple en 378h, 379h et 37Ah pour la première interface parallèle. Les figures suivantes décrivent la signification des différents bits des registres d'interface. Si vous confrontez la dénomination de ces bits à la description de la structure d'un câble Centronics (chapitre 7.2.4), vous trouverez de nombreuses concordances. Ceci n'est pas dû au hasard car les bits des registres d'interface sont directement reliés aux lignes de transmission du câble Centronics. Si on charge la valeur 1 dans l'un des bits, la ligne associée est aussitôt mise sous tension. Inversement, si le bit est mis à 0, la ligne est placée en mode "low". En principe chaque ligne garde son état jusqu'à ce que le bit associé soit changé par programme.

Notez cependant que certaines de ces lignes ont une logique inversée : leur nom est alors surligné ou précédé d'un signe -. La condition exprimée par une telle ligne est réalisée si elle est à 0. Par exemple la ligne -ERROR signale une erreur d'impression lorsque le bit correspondant est nul. Tant que la ligne est sous tension à l'état "high", aucune erreur ne s'est produite.

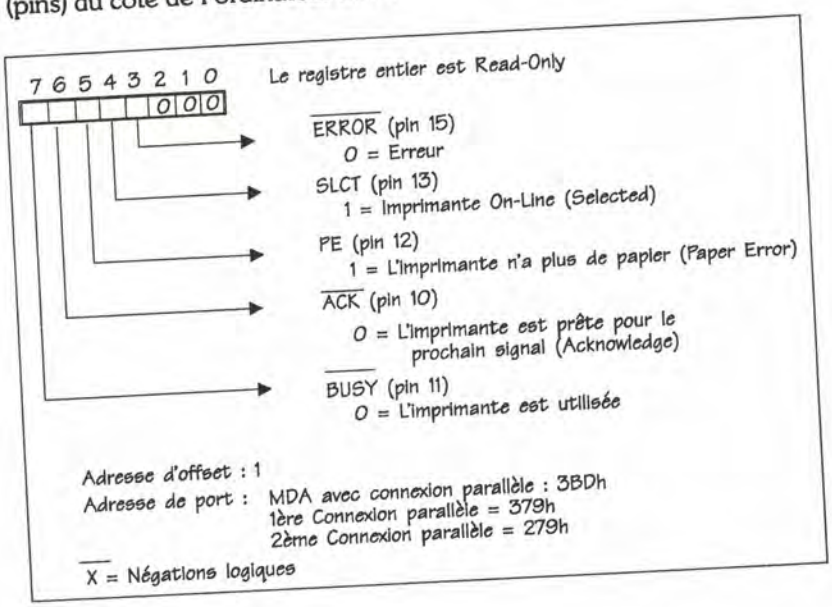
Lignes de données

Les huit bits du premier registre ne sont pas inversés. Ils représentent les données qui doivent être transférées sur les lignes D0 à D7. Il faut bien se rendre compte que ce registre est conçu comme un pur registre de sortie qui n'est pas destiné à réceptionner des donnnées. Il est vrai qu'une imprimante n'est pas là pour envoyer des données à l'ordinateur, et que l'interface Centronics n'a pas été pensée pour faire communiquer deux ordinateurs. En conséquence, un certain nombre de problèmes vont apparaître lorsqu'on cherchera à développer un logiciel de communication pour relier deux ordinateurs, car ce logiciel devra gérer des émissions et des réceptions. Mais nous verrons cela plus tard.



Etat de l'imprimante

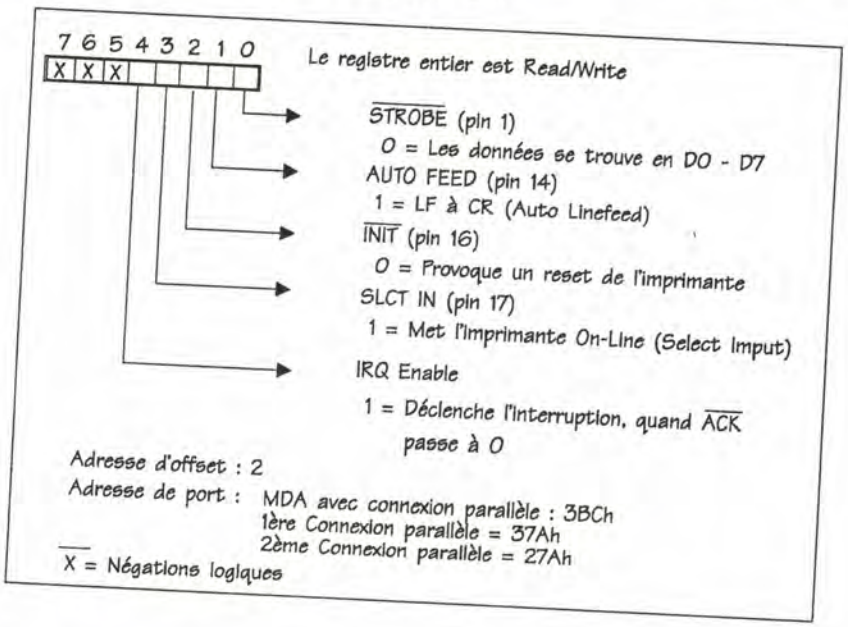
L'état courant de l'imprimante est décrit par le second registre qui n'est accessible qu'en lecture et ne peut pas recevoir de données. Il reflète l'état des différentes lignes d'état de l'imprimante. La figure suivante mentionne la correspondance des bits avec les broches (pins) du côté de l'ordinateur hôte.



Commande de l'imprimante

Le troisième registre sert à commander l'imprimante et le matériel. Par ailleurs il joue un rôle important dans la transmission des caractères. Les bits de 0 à 3 sont en correspondance directe avec des lignes de l'interface Centronics.

Un autre bit sert à déclencher une interruption matérielle dès que le signal -ACK passe à Low et que l'imprimante signale ainsi que la réception du dernier caractère. Le choix de l'interruption en question peut en général être fixé par des micro-interrupteurs DIP situés sur la carte d'interface. On peut prendre IRQ7 ou IRQ5 qui sont liés aux interruptions 0Fh et 0Dh. Contrairement à ce qui se passe pour les interfaces série, cette possibilité est rarement exploitée car les interfaces parallèles fonctionnent généralement en mode de consultation (polling) et non pas par interruption. Dans ce cas le BIOS ne touche pas aux vecteurs d'interruptions.



7.2.3. Structure de la communication

La signification des différentes lignes et des bits associés se conclut normalement par un coup d'oeil dans les coulisses de la communication entre ordinateur et imprimante. L'octet à imprimer est d'abord chargé dans le premier registre de l'interface parallèle, et par conséquent, dans les lignes D0 à D7. S'il est vrai que ces signaux arrivent instantanément à l'imprimante, cette dernière a pourtant besoin d'un signal supplémentaire pour les traiter. Il ne faut pas oublier en effet que les lignes D0 à D7 portent en permanence des informations, l'imprimante doit savoir si c'est un caractère à imprimer ou un reste de caractère déjà traité.

La ligne -STROBE

C'est alors qu'intervient la ligne -STROBE. En la mettant à 0, l'ordinateur signale la présence de données à imprimer sur les lignes de données. Il faut ensuite que ce signal

soit très rapidement retiré, sinon l'imprimante risque d'effectuer deux fois la lecture du caractère. Mais il faut aussi un délai minimal d'une microseconde pour laisser à l'électronique de l'imprimante le temps de lire les caractères.

-BUSY et -ACKnowledge

Une microseconde, ce n'est pas très long, et l'imprimante ne peut pas suivre ce rythme infernal, même en disposant rapidement les caractères dans un buffer interne. Elle signale alors par la ligne -BUSY qu'elle n'est pas encore prête pour la suite. Normalement ce signal est envoyé immédiatement après réception du signal -STROBE pour que l'imprimante ait le temps de lire tranquillement le caractère et de le traiter convenablement.

Comme le signal -BUSY suit une logique inversée, il prend la valeur 0 lorsque l'imprimante est affairée. Le programme -ici le BIOS - doit alors attendre son retour à 1 avant d'émettre un autre caractère. La ligne -BUSY est la seule de l'interface parallèle à être inversée au moment de la réception. Pour que l'ordinateur réceptionne 0, l'imprimante doit mettre la ligne à 1.

Mais ce n'est pas tout. le signal -ACK émis sur la ligne de même nom doit également être mis à 0 par l'imprimante pour avertir l'ordinateur que le caractère a été réceptionné. Si on étudie les allers et retours de ces signaux et leur durée de persistance (indépendamment de leur vitesse de transmission), on trouve un total de 10 microsecondes, ce qui permet en théorie de transmettre quelques 100000 caractères par seconde. Mais dans la réalité, la communication avec une imprimante ne dépasse guère le centième de ce chiffre (1000 caractères par seconde), même avec un buffer de réception des caractères.

L'imprimante contre-attaque

S'il est vrai que la communication entre ordinateur et imprimante ressemble à un monologue, l'imprimante n'est quand même pas complètement muette. Elle dispose de trois lignes pour donner diverses informations sur son état : -ERROR, SLCT et PE. Ces trois lignes aboutissent au premier registre de l'interface parallèle où elles peuvent être lues par programme.

SLCT veut dire Select et correspond à l'interrupteur ON-LINE situé sur le panneau de commandes de l'imprimante. Lorsque l'imprimante est désélectée par l'utilisateur, elle en fait part à l'ordinateur par le moyen de cette ligne.

PE signifie Paper Error et permet à l'imprimante d'indiquer que l'alimentation en papier est interrompue. Ce type d'erreur est maintenu distinct de l'erreur qui passe par la ligne ERROR. En effet le manque de papier peut être facilement corrigé par une intervention

de l'utilisateur, tandis qu'une erreur de transmission n'est pas réparable de la sorte. Dans ce dernier cas les responsables sont généralement un câble défectueux ou des signaux parasites que l'utilisateur ne peut identifier immédiatement.

Commande par l'ordinateur

Dernier perfectionnement qui n'est pas le moindre, l'ordinateur dispose de diverses lignes qui lui permettent de commander l'imprimante. Elles s'appellent : AUTO FEED, INIT et SCLT IN. Elles sont associées à divers bits du troisième registre et peuvent être librement mises à 0 ou à 1 par programme.

Lorsque le signal AUTO FEED est à 1, il indique à l'imprimante qu'elle doit ajouter à chaque caractère Entrée (retour chariot CR de code ASCII 13) un caractère de passage à une nouvelle ligne (LF ou LineFeed). Ce mécanisme a été introduit parce que toutes les imprimantes ne se comportent pas de la même façon en recevant un caractère CR. Certaines reviennent en début de ligne sans passer à la ligne suivante. Il faut alors simuler l'émission d'un caractère d'avancement de la ligne (LF) sinon l'impression d'une ligne recouvre la précédente. La ligne AUTO FEED permet d'éviter cet accident.

Avec la ligne SLCT IN l'ordinateur est en mesure de mettre l'imprimante hors ligne (OFF LINE) : il lui suffit d'envoyer le signal 0. Mais normalement c'est plutôt la valeur 1 qui doit être maintenue.

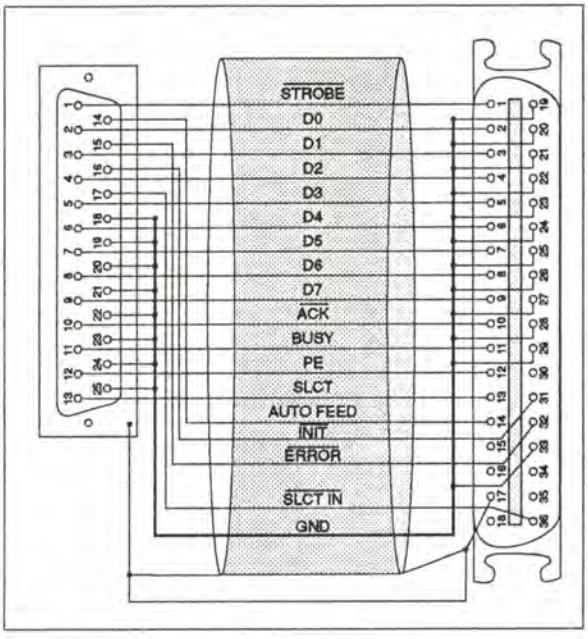
La ligne INIT permet à l'ordinateur de réinitialiser l'imprimante. Sa logique est inversée : pour provoquer le Reset, le bit correspondant doit être annulé. Mais il doit aussitôt être remis à 1, sinon l'imprimante entre dans une séquence interminable d'initialisations.

7.2.4. Les câbles

La communication entre ordinateur et imprimante ne fonctionne correctement que si les broches des deux interfaces sont convenablement reliées par le câble. Il existe un standard qui définit la répartition des signaux sur les différentes broches et comment les broches doivent être reliées. Ce standard s'appelle Centronics, il porte à la fois sur les connecteurs d'interface et les câbles de liaison.

Le tableau suivant en donne l'illustration.

Liaison par câble Centronics entre ordinateur et imprimante			
Ordinateur Pin	**Imprimante Pin**	**Nom Ligne**	**Rôle**
1	---------> 1	-STROBE	Déclenche la transmission
2	---------> 2	D0	Ligne de données Bit 0
3	---------> 2	D1	Ligne de données Bit 1
4	---------> 2	D2	Ligne de données Bit 2
5	---------> 2	D3	Ligne de données Bit 3
6	---------> 2	D4	Ligne de données Bit 4
7	---------> 2	D5	Ligne de données Bit 5
8	---------> 2	D6	Ligne de données Bit 6
9	---------> 2	D7	Ligne de données Bit 7
10	<--------- 10	-ACK	Réception du dernier caractère
11	<--------- 11	-BUSY	Imprimante occupée
12	<--------- 12	PE	Plus de papier sur l'imprimante
13	<--------- 13	SLCT	Imprimante ON-LINE
14	---------> 14	-AUTO FEED	LF succède automatiquement à LF
15	<--------- 32	-ERROR	Erreur de transmission
16	---------> 31	-INIT	Réinitialisation de l'imprimante
17	---------> 36	SLCT IN	Mise ON-LINE de l'imprimante
18-25	<-----> 19-30	GND	Masse



Structure d'un câble Centronics

Bricolage d'un câble null-modem

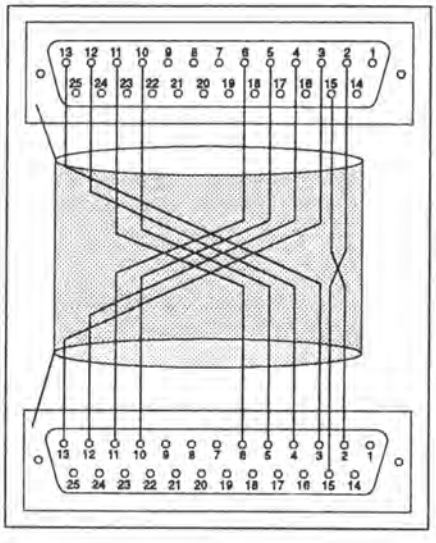
Si l'on veut tirer profit des interfaces parallèles pour transmettre des données entre ordinateurs, le câble Centronics ordinaire ne fait pas l'affaire. Des deux côtés les ordinateurs présentent des prises semblables femelles. Le câble Centronics ne peut pas s'y connecter, il faudrait commencer par taper dessus pour en réduire la taille. Mais ce n'est pas là le seul problème. D'après le schéma usuel, la transmission ne peut se faire que dans un seul sens : l'émetteur ne peut pas recevoir de données sur les lignes D0 à D7 , et inversement le récepteur ne peut pas utiliser ces lignes pour envoyer des données. Il faut établir une liaison bidirectionnelle pour que le récepteur puisse retourner des informations à l'émetteur, par exemple un total de vérification. Ce n'est qu'ainsi que l'émetteur pourra s'assurer que les données émises ont été transmises correctement.

On utilise à cette fin les différentes lignes d'état qui véhiculent habituellement des informations d'état en provenance de l'imprimante. Il s'agit des lignes -ERROR, SLCT, PE, -ACK et -BUSY qui sont associées au deuxième registre et peuvent y être lues sans problème. Elles sont reliées aux lignes D0 à D4 pour que les émissions de l'émetteur puissent être lues par le récepteur sur les lignes d'état. Inversement du côté du récepteur

les lignes D0 à D4 sont reliées aux lignes d'état pour qu'une communication soit également possible dans l'autre sens.

Concrètement on croise les lignes D0 à D4 avec les lignes d'état -ERROR, SLCT, PE, -ACK et -BUSY. La règle suivante est valable à la fois pour l'émetteur et le récepteur : celui qui écrit des données dans les cinq bits inférieurs du premier registre de l'interface parallèle les envoie sur les bits 3 à 7 du deuxième registre de son interlocuteur. Grâce à cette symétrie, il est indifférent d'échanger les extrémités des câbles.

La figure suivante montre comment sont reliées les broches des deux connecteurs d'un câble null-modem Centronics.



Câblage d'une liaison parallèle par null-modem

Malheureusement ce genre de câble se trouve difficilement dans le commerce. Voici donc quelques indications de bricolage. Il faut deux prises mâles SUB-D à 25 broches et un câble blindé qui ne doit pas dépasser 3 m de long (au-delà de cette longueur on risque de se trouver confronté à des problèmes de transmission).

Comme le montre le tableau suivant, il faut d'abord relier les broches 2 à 6 de l'une des prises avec les broches 15 à 10 de l'autre (à l'exception de 14). Choisissez cinq fils quelconques du câble et soudez-les aux broches 2 à 6 de la première prise. A l'autre extrémité, fixez les mêmes fils aux broches 15 à 10 en veillant évidemment à respecter la correspondance indiquée : par exemple ne reliez pas D0 à SLCT ou -BUSY mais à -ERROR.

Mise en relation des broches d'un câble parallèle null-modem			
Pin	Pin	Pin	Pin
2	15	15	2
3	13	13	3
4	12	12	4
5	10	11	6
6	11	10	5

Renouvelez ensuite la même opération dans l'autre sens de façon à obtenir le croisement recherché. N'oubliez pas de souder le blindage à la masse de la prise. Le câble ainsi constitué peut être utilisé pour exploiter des logiciels de communication du commerce, par exemple LapLink. Ces programmes en effet ne travaillent avec rien d'autre qu'un câble parallèle null-modem. S'il ne fonctionne pas, c'est soit en raison d'un mauvais bricolage, soit parce que à une autre répartition des lignes entre données et signaux d'état. Car les lignes de données D0 à D4 peuvent évidemment être reliées de façon différente aux lignes d'état, bien que la programmation s'en trouve compliquée.

7.2.5. Un programme de transfert de données

Avec un câble parallèle null-modem, il est possible de relier deux PC pour leur faire échanger des données. La première application qui vient à l'esprit est le transfert des données de l'un à l'autre mais le même câble peut aussi servir à commander un PC à partir d'un autre.

Dans cette section nous allons cependant étudier la première possibilité : j'ai réalisé pour vous deux programmes de transfert écrits en Pascal et C. Ils s'appellent PLINKP.PAS et PLINKC.C et leurs listings sont imprimés un peu plus loin. Avant d'expliquer le fonctionnement interne des deux programmes, qui de par leur structure est identique dans les deux cas, je voudrais décrire comment on les appelle.

Les deux programmes peuvent être émetteurs récepteurs. Le mode d'utilisation dépend de la forme de l'appel : si le nom du programme est suivi d'un ou plusieurs noms de fichiers, il se considère comme émetteur et cherche à envoyer à l'extérieur les fichiers mentionnés. Si aucun nom de fichier n'est mentionné dans l'appel, le programme se considère comme récepteur. Notez que les caractères génériques dits jokers peuvent être utilisés dans les noms de fichiers pour déclencher la transmission de toute une série de fichiers.

Les noms des fichiers peuvent être eux-mêmes suivis des paramètres optionnels /P et /T/. /P sert à indiquer que la transmission ne doit pas se faire par la première interface parallèle, il faut alors préciser le numéro de l'interface souhaitée (entre 1 et 4).

Le paramètre /T sert à fixer le délai de Time-Out. A priori, ce délai est fixé à 10 s mais vous pouvez imposer une autre valeur. Mettez à la suite du T le nombre de secondes souhaité. Ainsi avec la version en Pascal on pourra déclencher une émission de fichiers par une commande du genre :

```
PLINKP /p2 /t30 *.txt image.bmp
```

Transmission d'informations par le câble null-modem

Le câble null-modem présenté permet d'envoyer simultanément cinq bits par les lignes de données D0-D4. La transmission est même possible dans les deux sens en même temps puisque le câble est croisé. Mais il est indispensable de disposer d'un protocole approprié, autrement dit d'une sorte de ligne -STROBE qui imprime le rythme de la communication. L'un des cinq bits doit donc être sacrifié à cette fin. Le bit -BUSY semble tout indiqué. Seuls les bits situés aux extrémités entrent en ligne de compte, sinon les données sont coupées en deux : on peut donc choisir entre -ERROR et -BUSY. La ligne -BUSY est préférable car n'oublions pas qu'elle est inversée automatiquement par l'électronique de l'interface. Il faudrait donc lui restituer sa valeur d'origine si elle sert à transmettre des données, et ce serait là une perte de temps inutile.

Que le bit -STROBE soit inversé n'est pas gênant, il suffit d'en tenir compte dans l'écriture du protocole de transmission. Un 0 à l'une des extrémités signifiera un 1 à l'autre bout. Nous reviendrons là dessus dans un moment.

Le protocole de transmission dont il est question ici est celui des deux programmes de démonstration présentés. On peut imaginer une infinité de ces protocoles, et le nôtre est tout à fait particulier. Il travaille sur deux niveaux séparés : le niveau des octets et le niveau des blocs de données. Au niveau des octets le fonctionnement est orienté matériel tandis qu'au niveau des blocs il s'agit d'un protocole logiciel.

Transmission des octets

Examinons d'abord le niveau des octets du côté de l'émetteur. Un octet donné ne peut pas être transmis en une seule fois : il n'existe que quatre lignes pour cela. L'octet est donc divisé en deux quartets ("nibbles") émis l'un après l'autre selon le même schéma. Le quartet de poids faible est chargé dans les bits 0 à 3 du premier registre de l'interface et émis sur les lignes D0 à D3. Le bit de la ligne D4 est mis à 0 pour que le récepteur recueille la valeur 1 sur sa ligne -BUSY, ce qui lui indique que le quartet émis est prêt. Le récepteur attend patiemment cet événement en interrogeant en permanence la ligne -BUSY jusqu'à ce qu'elle monte à 1.

Le quartet est alors chargé depuis le deuxième registre de l'interface dans une variable. Le contenu ainsi lu est renvoyé à l'expéditeur par les lignes de données. Le bit de la

ligne D4 est mis à 0 pour que le récepteur trouve sa ligne -BUSY à 1. Ce dernier lit le quartet renvoyé et le mémorise.

Le quartet émis et le quartet renvoyé peuvent être comparés : il existe donc une vérification de la communication au plus bas niveau. Il faut dire cependant qu'en général on effectue plutôt ce genre de contrôle au niveau des blocs, sinon le temps perdu est trop important. Dans notre cas il n'y a pas de perte de temps car la ligne -BUSY doit de toute façon renvoyer un signal -STROBE à l'émetteur. Le fait de joindre à ce signal le quartet réceptionné n'est pas pénalisant.

Le même petit jeu se répète avec la transmission du second quartet qui correspond aux 4 bits de poids fort de l'octet. Mais cette fois ci l'émetteur se préoccupe d'abord de faire parvenir un 0 à la ligne -BUSY du récepteur. Dès que ce dernier s'en aperçoit, il lit le quartet, le renvoie et remet à 0 la ligne -BUSY de l'émetteur. Il reste à recomposer l'octet transmis avec les deux quartets.

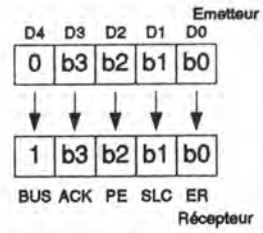
L'émetteur relit le deuxième quartet renvoyé pour contrôle, le concatène au second et teste s'il y a conformité avec l'original. En cas d'erreur, la routine d'émission transmet l'information au programme appelant, pour que les conséquences puissent en être tirées au plus haut niveau, par exemple par une demande de réémission du bloc. La quasi totalité des erreurs de transmission peut être ainsi déctectée, mis à part quelques cas rarissimes de dysfonctionnements dans les deux sens.

Comme pour la transmission ordinaire des données vers l'imprimante, tout repose ici sur la commutation répétitive du signal -STROBE sur la ligne -BUSY. Il faut bien veiller au moment du croisement du câble que l'on dispose de deux lignes -BUSY distinctes. Pour l'émetteur comme pour le récepteur, le bit -BUSY se trouve sur la ligne D4 au moment de son émission, sa destination étant la ligne d'état de même nom.

Le graphique suivant ne tient pas compte de ce fait. Destiné à visualiser le flux de données, il suppose qu'émetteur et récepteur utilisent les lignes D0 à D4 pour leurs émissions et qu'une ligne -STROBE séparée est à leur disposition.

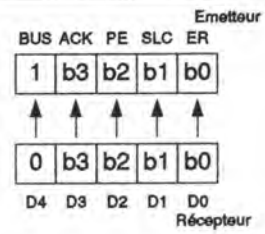
1.

L'émetteur commence par transmettre le quartet de poids faible. Il place le quartet sur les lignes DO à D3 et règle la ligne D4 sur 0 pour que le bit BUSY corresponde à 1 chez le récepteur.



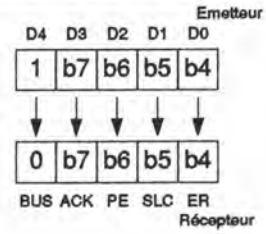
2.

Le récepteur s'attendait à ce que le bit BUSY commute sur 1. Il place le quartet reçu sur ses lignes D0 à D3 pour les retourner. En guise de caractère de réception, il met le bit D4 sur 0 pour que le bit BUSY corresponde à 1 chez l'émetteur.



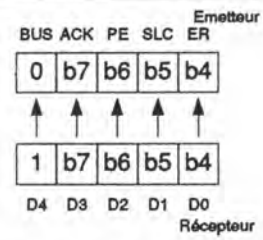
3.

L'émetteur espère aussi que son bit BUSY corresponde à 1 et accepte le quartet transmis. A cet effet, il place le quartet de poids fort sur les lignes de données et règle D4 sur 1. Le bit BUSY se transforme en 0 chez le récepteur.



4.

Cette fois-ci, c'est le récepteur qui s'attendait à ce que le bit BUSY devienne 0. Il retourne le quartet reçu et règle son bit de données D4 sur 1 pour que l'émetteur reçoive la valeur 0 dans le bit BUSY.



5.

La communication est terminée. Le récepteur confectionne un octet à partir des 2 quartets. A l'aide du message retourné, l'émetteur vérifie si les données peuvent être transmises correctement.

Protocole de transmission orienté matériel au niveau de l'octet

Problèmes de Time Out

Les protocoles de transmission fonctionnent bien tant que le flux de données est continu. Mais en cas d'interruption, ils s'effondrent immédiatement : l'un des deux interlocuteurs attend en vain que l'autre lui réponde, alors qu'il se trouve déconnecté ou dans l'impossibilité de réagir. Pour maîtriser cette difficulté, on fixe un seuil de Time Out. Il s'agit d'un délai au-delà duquel, en l'absence de réponse, un émetteur considère que son vis-à-vis n'est plus en état de prolonger la communication. Celle-ci est alors interrompue.

Comme l'a montré la section 7.1, le BIOS utilise aussi un compteur de Time Out lorsqu'il accède à l'interface parallèle. Le nombre d'itérations de la boucle d'interrogation est limité par ce paramètre. Pour des programmes qui doivent tourner sur toutes sortes de système différents, cette méthode n'est pas très sûre. Car la durée d'exécution de la boucle en question varie avec la vitesse du processeur. Les boucles d'attente que j'ai introduites dans mes programmes pour gérer les erreurs de Time Out se fondent aussi sur un comptage d'itérations. Mais le compteur n'est pas décrémenté par la boucle proprement dite mais par un gestionnaire d'interruption lié au timer de l'ordinateur. La fréquence de ce circuit est indépendante de la cadence du processeur, de ce fait la limite de Time Out est constante d'un système à l'autre.

Lorsque l'émetteur, après avoir envoyé le quartet inférieur, attend que son bit -BUSY soit mis à 1 par le récepteur, il donne d'abord à la variable de Time Out une valeur maximale. Puis il entre dans la boucle d'interrogation qui tourne jusqu'à ce que le bit -BUSY revienne effectivement à 1 ou que la variable de Time Out s'annule. Le gestionnaire d'interruption du timer commence à décrémenter le compteur à partir du moment où il découvre qu'il est différent de 0. En pseudo-code on peut écrire :

```
TimeOutCount = MAX
WHILE ( Bit-BUSY = 0 ) AND ( TimeOutCount > 0 ) DO
  BEGIN
  END

IF TimeOutCount = 0 THEN
  Erreur
ELSE
  o.k.
END
```

Dans nos deux programmes d'exemple ce sont les routines appelées EmetOctet et RecOctet qui activent le protocole de transmission avec la gestion du Time Out.

Bien que ce protocole fonctionne impeccablement une fois lancé, il peut quand même donner lieu à quelques difficultés. Il suppose en effet qu'à l'origine l'émetteur et le récepteur trouvent un bit -BUSY égal à 0. Si tel n'est pas le cas, le récepteur pense aussitôt qu'un quartet est déjà arrivé, alors que l'émetteur n'a encore rien envoyé.

Il faut donc commencer par synchroniser émetteur et récepteur. Si on ignore lequel des deux a été lancé en premier, on entre dans un problème compliqué qui plonge dans les tréfonds de la théorie des communications. C'est pourquoi dans le cadre des programmes présentés ici on a prévu une initialisation simple en supposant que le récepteur a été déclenché en premier.

Celui-ci attend que l'émetteur mette à 0 son bit -BUSY pour faire ensuite de même. L'émetteur attend en retour le même événement : ainsi la synchronisation finit par annuler les bits -BUSY des deux interlocuteurs.

Dans les programmes ci-après elle est mise en oeuvre par la routine PortInit. Bien entendu les boucles d'attente comportent une limite de Time Out selon le mécanisme décrit plus haut. Par ailleurs le programme doit être interruptible par la touche d'échappement. Sinon, en fonction de la valeur de Time Out, il peut s'écouler plusieurs minutes avant que le récepteur ne décèle l'impossibilité de joindre l'émetteur, alors que vous-même en tant qu'utilisateur vous avez déjà changé vos intentions et décidé de ne plus lancer le programme d'émission.

Interruption par Esc

En plus du gestionnaire d'interruption associé au timer les deux programmes introduisent aussi un gestionnaire d'interruption du clavier qui surveille la frappe de la touche Esc. Ce gestionnaire communique avec le programme comme le gestionnaire d'interruption du timer par l'intermédiaire d'une variable mais de type Booléen. Elle est mise à TRUE dès que le gestionnaire d'interruption du clavier détecte la frappe de Esc.

Pour que cette variable n'ait pas besoin d'être testée spécialement dans la boucle d'attente, il est possible de la coupler à la variable de Time Out. Lorsque cette association est active, le gestionnaire d'interruption du clavier ne répond pas seulement à la pression de Esc par la mise à TRUE de la variable Escape, mais il met aussi la variable de Time Out à 0.

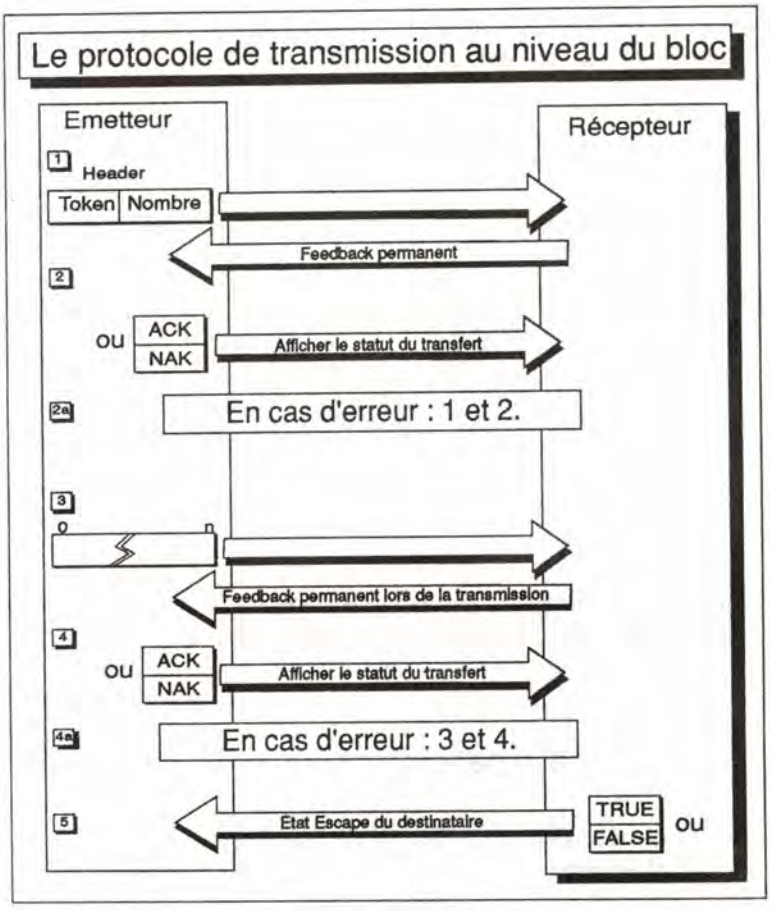
Pour le programme, tout se passe comme si la boucle d'attente était interrompue par un dépassement de Time Out, mais la lecture subséquente de la variable Escape révèle la véritable cause de l'interruption.

Le protocole de niveau supérieur

Au-delà du niveau des octets se situe le niveau supérieur des blocs. Géré par un protocole purement logiciel, il se rend indépendant du matériel en faisant appel aux routines d'émission et de réception de niveau inférieur. Dans nos programmes ces routines sont appelées EmetBloc et RecBloc.

Un bloc est constitué de trois informations : un token qui précède le bloc et décrit son contenu, la taille du bloc, et évidemment le bloc proprement dit. Le token et le nombre d'octets sont groupés dans une sorte de préfixe ou en-tête séparé des données. Le bloc de données ne doit pas être expédié avant que le récepteur n'ait reçu correctement le token et le nombre d'octets. Supposez par exemple que l'émetteur désire envoyer un bloc de 12 octets alors que le récepteur interprète la longueur comme ayant la valeur 200. Les 80 octets du bloc suivant vont être décomptés avec le premier bloc et le transfert finit en quenouille.

Comme au niveau inférieur le récepteur renvoie aussitôt chaque octet reçu, la transmission de l'en-tête peut être contrôlée sans problème. Malheureusement le récepteur n'a pas connaissance de cette information. Pour ne pas lui faire supposer à tort que l'en-tête a été bien transmis, l'émetteur lui signale la bonne marche des choses par un caractère convenu.



Protocole de transmission au niveau des blocs

Il émet ainsi un accusé de réception ACK (Acknowledge) lorsque la transmission s'est bien passée, ou un caractère NAK (Non-Acknowledge) en cas d'erreur. Le caractère ACK n'a rien à voir avec la ligne de même nom, il remplit simplement la même fonction. Dans les deux programmes présentés, ACK et NAK sont respectivement représentés par les codes 00h et FFh. D'autres codes pourraient être utilisés. Leur transmission est elle-même contrôlée au niveau inférieur des octets.

Lorsque l'en-tête et le caractère ACK sont parvenus sans erreur au récepteur, l'émetteur se met à envoyer le bloc de données proprement dit. En cas d'erreur, la transmission de l'en-tête est reprise. Le récepteur est au courant car il a été averti par le caractère NAK. Il ne se règle sur la réception du bloc de données qu'à partir du moment où il a réceptionné un caractère ACK. il est peu problable qu'une erreur transforme un caractère ACK en NAK si on prend la précaution de les exprimer par des codes différents.

L'émetteur ne recommence pas indéfiniment la transmission de l'en-tête, car après un certain nombre d'erreurs limité par la constante MAXTRY, il interrompt ses tentatives.

Lorsque le bloc est transmis, le même jeu des accusés ACK et NAK recommence : l'ensemble du bloc est émis jusqu'à ce que le récepteur reçoive confirmation du bon déroulement.

Grâce à ce mécanisme de transmission avec contrôles et répercussion des erreurs, les défauts sont détectés systématiquement. On évite ainsi d'avoir recours à des totaux de vérification, comme le font beaucoup de protocoles.

Prise en compte de la touche Esc

Lorsque le bloc a été transmis, un octet supplémentaire est envoyé en sens inverse car le récepteur doit pouvoir signaler à l'émetteur que la touche Esc a été frappée. Il est vrai que le récepteur pourrait mettre fin à l'exécution du programme dès qu'il a connaissance de cet événement. Mais du côté de l'émetteur il se produirait une erreur de Time Out car les lignes de transmission deviendraient soudain muettes. Ce procédé n'est donc pas très élégant. C'est pourquoi quand l'émetteur a envoyé le bloc de données, il attend le retour de l'octet Escape. Si la valeur est TRUE, il arrête le programme avec un message approprié, sinon l'exécution se poursuit. Mais il se peut aussi que ce soit l'émetteur qui ait directement connaissance de la frappe de la touche Escape. Il se comporte à cet égard différemment du récepteur. Au début de EmetBloc, la variable Escape est testée pour voir si l'utilisateur l'a actionnée. Si tel est le cas, au lieu d'envoyer le token habituel dans l'en-tête, l'émetteur envoie un token spécial qui est connu du récepteur. Celui-ci en l'identifiant mettra fin à l'exécution du programme tout comme l'émetteur à l'issue de la transmission de l'en-tête.

Cette imbrication du mécanisme d'échappement dans le protocole des blocs évite l'interrogation permanente de la touche Esc à un niveau supérieur. Car au-dessus du

niveau des blocs se trouve un autre niveau, celui des fichiers, qui se sert du niveau des blocs pour transmettre les fichiers morceau par morceau. Je ne souhaite pas entrer dans le détail de ce niveau car les deux programmes sont suffisamment explicites sur ce point.

Retenons simplement à ce stade que les routines des deux premiers niveaux (octet et bloc) permettent d'échanger des données quelconques entre émetteur et récepteur. La transmission peut se faire dans les deux sens, les fonctions d'émission et de réception pouvant être permutées. Les routines présentées ici peuvent donc servir de base au développement de vos propres programmes de transmission : il ne tient qu'à vous de les doter de la convivialité des logiciels commerciaux comme lapLink. Il est vrai que pour atteindre la même vitesse que ces derniers vous devrez écrire la totalité du niveau inférieur en assembleur. Avec le savoir-faire acquis dans ce chapitre, cela ne devrait pas vous poser trop de problème si vous connaissez bien l'assembleur.

Le niveau supérieur

Avant de vous livrer les listings des deux programmes étudiés avec leurs modules en assembleur, je voudrais évoquer un problème qui revient chaque fois des programmes suivent une hiérarchie très stricte, comme c'est le cas des protocoles de transmission (ici il s'agit des trois niveaux : octet, bloc et fichier). En cas d'erreur, par exemple s'il survient un Time Out au niveau de l'octet ou si la touche Esc a été détectée au niveau du bloc, il est important de pouvoir regagner rapidement le niveau supérieur. Dans un langage procédural, les niveaux correspondent à des fonctions et des procédures dont les appels sont imbriqués. Il n'est pas possible de remonter sans repasser par les niveaux successifs. Le bon vieux GOTO fait sentir son absence. mais les compilateurs C les plus modernes comportent les fonctions setjmp() et longjmp() qui permettent des branchements interprocéduraux. La fonction setjmp() sert à marquer une destination de branchement. Le saut se réalise lorsqu'en cas d'erreur on fait appel à la fonction associée longjmp(). les manuels de votre compilateur décrivent en détail le mode d'emploi de ces fonctions..

Listing : PLINKP.PAS

```
{*********************************************************************}
{                          P L I N K P . P A S                        }
{---------------------------------------------------------------------}
{    Fonction       : transmet des fichiers par le port parallèle     }
{---------------------------------------------------------------------}
{    Auteur         : Michael Tischer                                 }
{    Développé le   : 27.09.1991                                      }
{    Dernière MAJ   : 28.11.1991                                      }
{*********************************************************************}
{$M 65520, 0, 655360 }

uses dos, crt;                          { Intègre les unités DOS et CRT }

{-- Constantes ------------------------------------------------------}

const UNESEC      = 18;                              { Une seconde }
      DIXSEC      = 182;                             { Dix secondes }
      TO_DEFAULT  = DIXSEC;                          { Time-Out par défaut }

      MAXBLOC     = 4096;          { Taille d'un bloc 4 Ko ( Cluster ) }

{-- Constantes pour le protocole de transmission --------------------}

      ACK         = $00;                         { Accusé de réception }
      NAK         = $FF;                     { Non accusé de réception }
      MAX_TRY     = 5;                   { Nombre de tentatives autorisées }

{-- Tokens pour la comunication entre émetteur et récepteur --------}

      TOK_DATSTART = 0;                            { Début de fichier }
      TOK_DATNEXT  = 1;                      { Suite d'un bloc de données }
      TOK_DONNEESD = 2;             { Transmission du fichier terminée }
      TOK_FIN      = 3;                            { Fin du programme }
      TOK_ESCAPE   = 4;{ Interruption par ESC sur l'ordinateur distant }

{-- Codes pour les appels LongJump ----------------------------------}

      LJ_OKEMETTEUR = 1;       { Tous les fichiers correctement émis }
      LJ_OKRECEPT   = 2;       { Tous les fichiers correctement reçus }
      LJ_TIMEOUT    = 3;    { Time-out: le correspondant ne répond pas }
      LJ_ESCAPE     = 4; { Interruption par Esc sur l'ordinateur local }
      LJ_REMESCAPE  = 5;{ Interruption par Esc sur l'ordinateur distant }
      LJ_DATA       = 6;                  { Erreur de communication }
      LJ_NOLINK     = 7;                          { Pas de liaison }
      LJ_NOPAR      = 8;                       { Pas d'interface }
      LJ_PARA       = 9;                { Paramètres d'appel invalides }

{-- Définitions de types --------------------------------------------}

type BHEADER = record             { En-tête pour transmission des blocs }
                 case boolean of
                   true  : ( Token : byte;
                             Len   : word );
                   false : ( Champ : array[ 0..2 ] of byte );
               end;

     JMPBUF  = record             { Enregistrement des informations , }
         BP,                                { nécessitées pour le branchement }
             SP,
         CS,                       { ne pas changer l'ordre des variables ! }
             IP  : word;
               end;

     DBloc = array[ 1..MAXBLOC ] of byte;          { Données d'un bloc }

{-- Variables globales ----------------------------------------------}

var InPort       : word;                    { Adresse du port d'entrée }
    OutPort      : word;
    Escape       : boolean;           { Pas de touche ESCAPE enfoncée }
    Timeout      : word;              { valeur de Timeout sélectionnée }
    TO_Count     : word;                          { Compteur de Timeout }
    Branchement  : JMPBUF;          { Adresse de retour pour terminer }
    BlocBuf      : DBLOC;                { Buffer pour mémoriser un bloc }
    Fichier      : file;    { Variable fichier pour traiter un fichier }

{-- Déclaration des fonctions en assembleur -------------------------}

{$L plinkpa.obj }    { Intègre le module en assembleur }

function  getb : byte; external;
procedure putb( Wert : byte ); external;
procedure intr_install(escape_flag, timeout_count : pointer); external;
procedure intr_remove; external;
procedure EscapeDirect( Declenche : boolean ); external;

{*********************************************************************}
{ SetJmp : Détermine l'emplacement du programme où doit se brancher   }
{          LongJmp                                                    }
{---------------------------------------------------------------------}
{ Entrée  : JB : Structure de données du type JumpBuf, qui contient   }
{                les informations nécessaires au branchement          }
{ Sortie  : NOJMP au retour de cette fonction ; toute autre valeur    }
{           lorsque le retour suit un appel à LongJmp                 }
{*********************************************************************}

{$F+} { SetJmp et LongJmp doivent être de type FAR }

function SetJmp( var JB : JMPBUF ) : integer;

type WordP  = ^word;                            { Pointeur sur un mot }

begin
  JB.BP := WordP( ptr( SSeg, Sptr+2) )^;
  JB.IP := WordP( ptr( SSeg, Sptr+4) )^;
  JB.CS := WordP( ptr( SSeg, Sptr+6) )^;

      { SP doit référencer la position à partir de laquelle LongJmp va }
    { ultérieurement mémoriser le nouveau contenu de BP et l'adresse de }
                                                      { retour sur la pile }

  JB.SP := Sptr + 12 - 6 - 6;

  SetJmp := -1;      { Indique qu'il ne s'agit pas d'un appel à LongJmp }
end;

{*********************************************************************}
{ LongJmp : GOTO interprocédural, qui poursuit l'exécution du         }
{           programme à la ligne où SetJmp a été appelé               }
{---------------------------------------------------------------------}
{ Entrée  : JB      : Buffer de branchement rempli par SetJmp         }
{           RetCode : Résultat à renvoyer par SetJmp                  }
{*********************************************************************}

procedure LongJmp( JB : JMPBUF; RetCode : integer );

type WordP  = ^word;                            { Pointeur sur un mot }

begin
  WordP( ptr( SSeg, JB.SP   ) )^ := JB.BP;
  WordP( ptr( SSeg, JB.SP+2 ) )^ := JB.IP;
  WordP( ptr( SSeg, JB.SP+4 ) )^ := JB.CS;

                  (-- Charge en AX le code de retour transmis ce qui --)
                                  (-- simule le résultat de SetJmp --)

  inline( $8b / $46 / $06 );                          { mov ax,[bp+6] }

  inline( $8b / $6e / $fa / $ff );                    { mov bp,[bp-6] }

        ( mov   sp,bp          ;Ces instructions sont automatiquement }
                pop   bp              ;générées par le compilateur }
                ret   6               ;restaurant ainsi la pile }
end;

{$F-}

{*********************************************************************}
{ GetPortAdr : Initialise les adresses des ports d'une interface      }
{              parallèle c'est-à-dire les variables InPort et OutPort }
{ Entrée     : NUMERO = Numéro de l'interface parallèle (1-4)         }
{ Sortie     : TRUE, si interface valide                              }
{ Var. globales : InPort/W, OutPort/W                                 }
{ Info       : Les adresses de base des interfaces parallèles         }
{              (en nombre de 1 à 4)  se trouvent dans les mots        }
{              mémoire commençant en 0040:0008                        }
{*********************************************************************}

function GetPortAdr( Numero : integer ) : boolean;

begin
  {-- Lit els adresses des ports dans le segment des variables du BIOS--}
  OutPort := MemW[ $0040: 6 + Numero * 2 ];
  if ( OutPort <> 0 ) then                    { Interface disponible ? }
    begin                                                    { Oui }
      InPort := OutPort + 1;          { Adresse pour registre d'entrée }
      GetPortAdr := TRUE;                          { Retour sans erreur }
    end
  else
  GetPortAdr := FALSE;                         { Erreur: interface absente }
end;

{*********************************************************************}
{ Port_Init     : Initialise les registres nécessaires à la           }
```

```
{                transmission                                          }
{ Entrées      : EMETTEUR = TRUE, si émetteur, FALSE, si récepteur     }
{ Sortie       : TRUE, si registres correctement initialisés           }
{ Var. globales : InPort/R, OutPort/R                                  }
{ Info         : La dissymétrie: envoie 00010000 , attend 00000000     }
{                est rendue nécessaire par l'inversion du signal       }
{                Normalement les registres d'entrée et de sortie       }
{                contiennent les valeurs souhaitées mais               }
{                l'initialisation est nécessaire lorsqu'on reprend une }
{                transmission interrompue.                             }
{*************************************************************}

function Port_Init( Emetteur : boolean ) : boolean;

begin
  EscapeDirect( TRUE );                    { Déclenche un Time out si Escape }
  if ( Emetteur ) then                       { L'ordinateur est-il émetteur? }
    begin
      TO_Count := Timeout * 5;     { Initialise le compteur de Time Out }
      PutB( $10 );                                  { Envoie : 00010000b }
      while ( ( GetB <> $00 ) and ( TO_Count > 0 ) ) do     { Attend 0 }
        :
    end
  else                                          { L'ordinateur est récepteur }
    begin
      TO_Count := Timeout * 5;     { Initialise le compteur de Time Out }
      while ( ( GetB <> $00 ) and ( TO_Count > 0 ) ) do     { Attend 0 }
        :
      PutB( $10 );                                  { Envoie: 00010000b }
    end;
  EscapeDirect( FALSE );                    { Si Escape pas de Timeout }
  Port_Init := ( TO_Count > 0 );                 { Initialisation terminée }
end;

{*************************************************************}
{ EmetOctet     : Envoie un octet en deux parties à l'ordinateur      }
{                 distant et teste le résultat                        }
{ Entrées       : VALEUR = octet à émettre                            }
{ Sortie        : Transmission correcte ? ( 0 = Erreur , -1 = ok )    }
{ Var. globales : Timeout/R, InPort/R, OutPort/R (dans Macros)        }
{*************************************************************}

function EmetOctet( Wert : byte ) : boolean;

var Retour : byte;                                  { Octet réceptionné }

begin
        {-- Emet le quartet inférieur ------------------------------}

  TO_Count := Timeout;              { Initialise le compteur de Time Out }
  PutB( Wert and $0F );                    { Envoi avec mise à 0 de BUSY }
  while ( ( ( GetB and 128 ) = 0 ) and ( TO_Count > 0 ) ) do
    :
  if ( TO_Count = 0 ) then                          { Erreur de Time Out ? }
    longJmp( Branchement, LJ_TIMEOUT );   { Interrompt la transmission }

  Retour := ( GetB shr 3 ) and $0F;                      { Bits 3-6 en 0-3 }

  {-- Emet le quartet supérieur ------------------------------------}

  TO_Count := Timeout;              { Initialise le compteur de Time Out }
  PutB( ( Wert shr 4 ) or $10 );          { Envoie avec mise à 1 de BUSY }
  while ( ( ( GetB and 128 ) <> 0 ) and ( TO_Count > 0 ) ) do

  if ( TO_Count = 0 ) then                          { Erreur de Timeout }
    longJmp( Branchement, LJ_TIMEOUT );   { Interrompt la transmission }

  Retour := Retour or ( ( GetB shl 1 ) and $F0 );     { Bits 3-6 en 4-7 }
  EmetOctet :=  ( Wert = Retour );       { Octet correctement transmis ? }
end;

{*************************************************************}
{ RecOctet      : Réceptionne un octet en deux parties de la part d'un }
{                 ordinateur distant et renvoie les parties pour      }
{                 vérification                                        }
{ Entrées       : aucune                                              }
{ Sortie        : octet reçu                                          }
{ Var. globales : Timeout/R, InPort/R, OutPort/R ( Macros )           }
{*************************************************************}

function RecOctet : byte;

var LoNib,
    HiNib : byte;                                   { Quartets reçus }

begin
  {-- Réceptionne et renvoie le quartet inférieur ------------------}

  TO_Count := Timeout;              { Initialise le compteur de Time Out }
  while ( ( ( GetB and 128 ) = 0 ) and ( TO_Count > 0 ) ) do
    :

  if ( TO_Count = 0 ) then                          { Erreur de Time Out ? }
    longJmp( Branchement, LJ_TIMEOUT );   { Interrompt la transmission }

  LoNib := ( GetB shr 3 ) and $0F;                       { Bits 3-6 en 0-3 }
  PutB( LoNib );                                  { Retour à l'expéditeur }

  {-- Réceptionne et renvoie le quartet supérieur ------------------}

  TO_Count := Timeout;              { Initialise le compteur de Time Out }
  while ( ( ( GetB and 128 ) <> 0 ) and ( TO_Count > 0 ) ) do
    :

  if ( TO_Count = 0 ) then                          { Erreur de Time Out ? }
    longJmp( Branchement, LJ_TIMEOUT );   { Interrompt la transmission }

  HiNib := ( GetB shl 1 ) and $F0;                       { Bits 3-6 en 4-7 }
  PutB( ( HiNib shr 4 ) or $10 );      { Renvoie en let
      Busy à 1 )

  RecOctet := ( LoNib or HiNib );                          { Octet reçu }
end;

{*************************************************************}
{ EmetBloc : Emet un bloc de données                                   }
{ Entrée   : TOKEN  = Commande pour le récepteur                       }
{            NOMBRE = Nombre d'octets à transmettre                    }
{            DPTR   = Pointeur sur le buffer des données               }
{ Sortie   : néant, en cas d'erreur branchement par LongJmp à la       }
{            routine de traitement d'erreur                            }
{*************************************************************}

procedure EmetBloc( Token  : byte;
                    Nombre : word;
                    Dptr   : pointer );

var header     : BHEADER;     { En-tête pour mémorise Token et  Nombre }
    RecEscape : byte;{ Entrée la touche ESC sur l'ordinateur distant ? }
    ok         : boolean;                          { Indicateur d'erreur }
    i          : word;                          { Compteur d'itérations }
    try        : word;                  { Nombre de tentatives restantes }
    Donnees    : ^DBloc;                 { Pointeur sur bloc de données }

begin
  if ( Escape ) then          { A-t-on tapé Escape sur cet ordinateur ? }
    begin
      Token := TOK_ESCAPE;                { Oui, émettre  le token Escape }
      Nombre := 0;
    end;

      {-- Emission de l'en-tête ----------------------------------------}

  header.Token := Token;                                { Construit l'en-tête }
  header.Len := Nombre;

  try := MAX_TRY;
  repeat                                          { AU maximum MAX_TRY tentatives }
    ok := TRUE;                          { A priori la transmission est bonne }
    for i := 0 to 2 do
      ok := ok and EmetOctet( Header.Champ[ i ] );      { Emet un octet }
    if ( ok ) then
      ok := ok and EmetOctet( ACK )                          { Confirmation }
    else
      ok := ok and EmetOctet( NAK );                         { Confirmation }
    if ( not ok ) then                                            { Erreur ? }
      dec( try );                                    { Oui, autre tentative }
  until ( ( ok ) or ( try = 0 ) );

  if ( try = 0 ) then           { L'en-tête a-t-il pu être transmis ? }
    longJmp( Branchement, LJ_DATA ); { Non, interrompt la transmission }

  if ( Token = TOK_ESCAPE ) then     { A-t-on envoyé l'avis d'ESCAPE ? }
    longJmp( Branchement, LJ_ESCAPE );  { Oui, interrompt transmission }

    {-- Emission du bloc de données proprement dit-------------------}

  if ( Nombre > 0 ) then                                { Taille diff de 0  ? }
    begin
      Donnees := DPTR;
      try := MAX_TRY;
      repeat
    ok := TRUE;                          { A priori la transmission est bonne }
        for i := 1 to Nombre do
          ok := ok and EmetOctet( Donnees^[ i ] );
        if ( ok ) then
      ok := ok and EmetOctet( ACK )                          { Confirmation }
        else
      ok := ok and EmetOctet( NAK );
    if ( not ok ) then                                            { Erreur ? }
      dec( try );                                    { Oui, nouvelle tentative }
      until ( ( ok ) or ( try = 0 ) );
       if ( try = 0 ) then { Le bloc de données transmis correctement ? }
    longJmp( Branchement, LJ_DATA ); { Non, interrompt la transmission }
    end;
```

```
  {-- Teste l'octet ESCAPE du récepteur ----------------------------}

  try := MAX_TRY;
  repeat
    RecEscape:=RecOctet;                          { Détecte un Escape distant }
    dec( try );
  until ( ( RecEscape = byte( true ) ) or
          ( RecEscape = byte( false ) ) );

  if ( try = 0 ) then                          { Etat de la touche Escape reçu ? }
    longjmp( Branchement, LJ_DATA ); { Non, interrompt la transmission }

  if ( RecEscape = byte( true ) ) then ( Esc. sur ordinateur distant ? )
    longjmp( Branchement, LJ_REMESCAPE );{ Oui interrompt transmission }
end;

{*****************************************************************}
{ RecBloc : Réceptionne un bloc de données                        }
{ Entrées : TOKEN  = Pointeur sur la variable qui mémorise le token }
{           LEN    = Pointeur sur la variable qui mémorise la longueur }
{           DONNEES = Pointeur sur le buffer des données          }
{ Sortie  : néant, en cas d'erreur branchement sur routine d'erreur }
{           par LongJmp                                           }
{ Info    : Le buffer translis doit prévoir de la place pour      }
{           MAXBLOC octets, la taille du bloc ne pouvant          }
{           être anticipée                                        }
{*****************************************************************}

procedure RecBloc( var Token : byte;
                       var Len   : word;
                           Dptr  : pointer );

var header       : BHEADER;                  { En-tête pour Token et Nombre }
    ok           : boolean;                        { Indicateur d'erreur }
    i            : word;                       { Compteur d'itérations }
    try          : word;                   { Nombre de tentatives restantes }
    EscapeStatus : boolean;
    ByteBuffer   : byte;
    Donnees      : ^DBloc;                 { Pointeur sur bloc de données }

begin
      {-- Réceptionne d'abord l'en-tête ------------------------------}

  try := MAX_TRY;
  repeat
    for i:= 0 to 2 do
      Header.Champ[ i ] := RecOctet;

    ByteBuffer := RecOctet;
    if ( ByteBuffer <> ACK ) then      { Tous les octets bien reçus ? }
      dec( try );                                { Oui, pas de nouvelle tentative }
  until ( ( try = 0 ) or ( ByteBuffer = ACK ) );

  if ( try = 0 ) then                          { En-tête correctement reçu ? }
    longjmp( Branchement, LJ_DATA );  { Non Interrompt la transmission }

  Token := Header.Token;
  Len := Header.Len;
  if ( Token = TOK_ESCAPE ) then                    { Emetteur ESCAPE? }
    longjmp( Branchement, LJ_REMESCAPE );{ Oui interrompt transmission }

        {-- L'en-tête est bon, il faut passer au bloc des données. --}

  if ( Len > 0 ) then                          { Pas de bloc de données ? }
    begin                                                        { si}
      Donnees := Dptr;
      try := MAX_TRY;

      repeat                          { Réceptionne le bloc octet par octet}
        for i := 1 to len do
          Donnees^[ i ] := RecOctet;

        ByteBuffer := RecOctet;
      if ( ByteBuffer <> ACK ) then      { Tous les octets bien reçus ? }
        dec( try );                              { Oui, pas de nouvelle tentative }
      until ( ( try = 0 ) or ( ByteBuffer = ACK ) );

      if ( try = 0 ) then                  { Le bloc a-t-il été bien reçu? }
    longjmp( Branchement, LJ_DATA ); { Non, interrompt la transmission }
    end;

{-- Envoie à l'ordinateur distant l'état actuel de la touche Escape --}

  EscapeStatus := Escape;                                { Mémorise l'état }

  try := MAX_TRY;
  repeat
    dec( try );
  until ( EmetOctet( byte( EscapeStatus ) ) or ( try = 0 ) );

  if ( try = 0 ) then                  { L'état ESC a-t-il pu être envoyé ? }
    longjmp( Branchement, LJ_DATA ); { Non, interrompt la transmission }

  if ( EscapeStatus ) then {A-t-on actionné Escape sur cet ordinateur ?}
    longjmp( Branchement, LJ_ESCAPE );  { Oui, interrompt transmission }
end;

{*****************************************************************}
{ EmetFichier : Emet un fichier                                   }
{ Entrées     : NAME = Nom de fichier                             }
{ Sortie      : néant                                             }
{*****************************************************************}

procedure EmetFichier( Name : string );

var Status  : word;                                { Etat d'émission }
    Lus : word;                                  { Nombre d'octets lus }
    Taille : longint;                        { Nombre d'octets envoyés }

begin
  write( copy( Name + '                    ', 1, 13 ) );
  assign( Fichier, Name );
  reset( Fichier, 1 );
  EmetBloc( TOK_DATSTART, length( Name ) + 1, @Name ); { Envoie le nom }

  {-- Transmet le contenu du fichier -----------------------------}

  Taille := 0;
  repeat
    Blockread( Fichier, BlocBuf, MAXBLOC, Lus );          { Lit un bloc }
    if ( Lus > 0 ) then                                    { Terminé ? }
      begin                                                        { Non }
    EmetBloc( TOK_DATNEXT, Lus, @BlocBuf );             { Envoie le bloc }
        inc( Taille, Lus );
        write( #13, copy( Name + '                ', 1, 13 ),
               '(', Taille, ')' );
      end;
  until ( Lus = 0 );
  writeln;

  EmetBloc( TOK_DONNEESD, 0, NIL );             { Clôture la transmission }

  close( Fichier );                                { Referme le fichier }
end;

{*****************************************************************}
{ RecFichier : Réceptionne un fichier                             }
{ Entrées    : néant                                              }
{ Sortie     : Dernier token reçu                                 }
{*****************************************************************}

function RecFichier : word;

var Status      : word;                              { Etat de réception }
    AEnregistrer : word;                        { Taille du dernier bloc }
    Taille      : longint;
    Token       : byte;                                  { Token reçu }
    Len         : word;                                { Longueur reçue }
    i           : word;                        { Compteur d'itérations }
    Name        : string[ 13 ];                          { Nom du fichier }

begin
  RecBloc( Token, Len, @BlocBuf );
  if ( Token = TOK_DATSTART ) then
    begin
      for i := 0 to BlocBuf[ 1 ] do
        Name[ i ] := chr( BlocBuf[ i + 1 ] );
      assign( Fichier, Name );
      rewrite( Fichier, 1 );
      write( copy( Name + '                    ', 1, 13 ) );

    {-- Réceptionne le contenu du fichier --------------------------}

      Taille := 0;
      repeat
    RecBloc( Token, Len, @BlocBuf );              { Réceptionne un bloc }
    if ( Token = TOK_DATNEXT ) then      { Bloc de données consécutif ?}
          begin                                                  { Oui }
        BlockWrite( Fichier, BlocBuf, Len );              { Enregistre }
            inc( Taille, Len );
            write( #13, copy( Name + '                ', 1, 13 ),
                   '(', Taille, ')' );
          end;
      until ( TOKEN <> TOK_DATNEXT );
      close( Fichier );                              { Referme le fichier }
      writeln;
    end;
  RecFichier := Token;                          { Retourne l'état d'erreur }
end;

{*****************************************************************}
{                        PROGRAMME PRINCIPAL                      }
{*****************************************************************}

const Avis : array[ 0..8 ] of string =
         ( 'FIN: Tous les fichiers ont été correctement émis.',
```

```
        'FIN: Tous les fichiers ont été correctement reçus.',
        'ERREUR: Time-Out, le système distant ne répond pas.',
        'FIN: Interruption par Escape.',
        'FIN: Interruption par Escapr sur l''ordinateur distant.',
        'ERREUR: Interface ou câble défectueux !',
        'ERREUR: Pas de contact avec l''ordinateur distant.',
        'ERREUR: L''interface indiquée n''existe pas !',
        'ERREUR: Paramètre inconnu ou invalide !' );

var SRec     : SearchRec;{ Structure pour recherche dans un répertoire }
    Emetteur : boolean;{ Mode de transmission (Emetteur, récepteur ) }
    sjStatus : integer;                              { Code de longjump }
    Numero.                                        { Numéro de l'interface }
    i.                                                  { Compteur }
    Trouve : byte;                          { Pour la recherche des fichiers }
    dummy    : integer;
    argv     : array[ 1..10 ] of string;                  { Paramètres }


begin
  write( #13#10'Transmission de données par l''interface parallèle' );
  writeln( '   (c) 1991 by Michael Tischer' );
  write( '-------------------------------------' );
  writeln( '-------------------------' );

  Escape := false;
  Timeout := TO_DEFAULT;

  if ( paramstr( 1 ) = '?' ) then              { Affiche juste la syntaxe }
    begin                                                      { Oui }
      writeln( 'Appel: plinkc [/Pn] [/Tnn] [Nom de fichier]' );
      halt( 0 );
    end;

  sjStatus := setjmp( Branchement ); { Adresse de retour pour terminer }
  if ( sjStatus > 0 ) then                          { Longjmp appelé ? }
    begin                                                      { Oui }
      Intr_remove;    { désactive le gestionnaire
     d'interruption }
      writeln( #13#10@13#10, Avis[ sjStatus - 1 ] );
      halt( 0 );
    end;

  Intr_Install( @Escape, @TO_Count );{Initialise le driver d'interrupt.}

{-- Fixe les paramètres par défaut et exploite la ligne de commande --}

  Emetteur := FALSE;              { Par défaut l'ordinateur est récepteur }
  Numero := 1;                                    { et l'interface LPT1 }

  for i := 1 to paramcount do
    begin
      argv[ i ] := paramstr( i );              { mémorise les paramètres }
      if ( argv[ 1, 1 ] = '/' ) then
        begin
          case ( upcase( argv[ 1, 2 ] ) ) of
            'T' : begin
                    delete( argv[ i ], 1, 2 );
                    val( argv[ i ], Timeout, dummy );
                    Timeout := ( Timeout * DIXSEC ) div 10;
                    if ( Timeout = 0 ) then
                      longjmp( Branchement, LJ_PARA );          { Incorrect }
                  end;
            'P' : begin                                         { Interface }
                    Numero := ord( argv[ 1, 3 ] ) - 48;
                    if ( ( Numero = 0 ) or ( Numero > 4 ) ) then
                      longjmp( Branchement, LJ_PARA );          { Incorrect }
                  end;
            else longjmp( Branchement, LJ_PARA );               { Inconnu }
          end;
        argv[ i ] := '';                                { Efface l'argument }
      end
    else                                          { Doit être un nom de fichier }
      Emetteur := TRUE;                                        { Emetteur }
  end;

{-- Démarre la transmission ------------------------------------------}

  if ( not GetPortAdr( Numero ) ) then     { L'interface existe-t-elle ? }
    longjmp( Branchement, LJ_NOPAR );                      { Non, erreur }

  if ( not Port_Init( Emetteur ) ) then                { Etablit la liaison }
    longjmp( Branchement, LJ_NOLINK );                 { Erreur, impossible }

  if ( Emetteur ) then                                          { Emetteur? }
    begin
      writeln( 'Emission vers LPT', Numero, #13#10 );

      {-- Transmet tous les fichiers ---------------------------------}

      for i := 1 to paramcount do          { Parcourt la ligne de commande }
        begin
      if ( argv[ i ] <> '' ) then                          { Nom de fichier ? }
            begin                                                  { Oui }
              findfirst( argv[ i ], AnyFile, SRec);
            while ( DosError = 0 ) do                  { Tant qu'il y en a }
                begin
                  if ( SRec.Attr <> Directory ) then
                EmetFichier( SRec.Name );              { Transmet le fichier }
                  findnext( SRec );
                end;
            end;
        end;
      EmetBloc( TOK_FIN, 0 , NIL );                { Tous les fichiers émis }
      longjmp( Branchement, LJ_OKEMETTEUR );
    end
  else                                                    { Non, Récepteur }
    begin
      writeln( 'Réception sur LPT', Numero, #13#10 );
      while ( RecFichier <> TOK_FIN ) do       { Réceptionne les fichiers }
        ;                                                { jusqu'au token de FIN }
      longjmp( Branchement, LJ_OKRECEPT );
    end;
end.
```

Listing : PLINKPA.ASM

```
;*****************************************************************;
;*                        P L I N K P A . A S M                  *;
;*---------------------------------------------------------------*;
;*   Fonction        : Complément en assembleur du programme Pascal *;
;*                     PLINKP - Contient en plus des gestionnaires *;
;*                     d'interruption des routines d'accès rapide  *;
;*                     aux ports                                   *;
;*---------------------------------------------------------------*;
;*   Auteur          : MICHAEL TISCHER                           *;
;*   Développé le    : 10.10.1991                                *;
;*   Dernière MAJ    : 11.10.1991                                *;
;*---------------------------------------------------------------*;
;*   Assemblage      : TASM PLINKPA                              *;
;*                     ... puis lier à PLINKP                    *;
;*****************************************************************;

;=== Constantes ================================================

KB_PORT   = 60h                                  ;Port du clavier
INT_CTR   = 20h                  ;Port du contrôleur d'interruption
EOI       = 20h                      ;Commande fin d'interruption
ESCAPE    = 1                       ;Scan-Code de la touche Escape

;=== Segment de données ========================================

DATA   segment word public                       ;Segment de données TP

extrn    InPort                                  ;Port d'entrée comme variable TP
extrn    OutPort                                 ;Port de sortie comme variable TP

DATA   ends                                      ;Fin du segment de données

;=== Programme =================================================

CODE       segment byte public                   ;Segment de code TP

           assume cs:CODE, ds:DATA, es:nothing, ss:nothing

;-- Déclarations publiques de fonctions internes ---------------

public     intr_install          ;Permet l'appel à partir du programme TP
public     intr_remove
public     escapedirect
public     getb
public     putb

;-- Variables pour les gestionnaires d'interruption------------------
;-- (accessibles uniquement par le segment de code ------------

key_ptr    dd 0                          ;Pointeur sur la var pour ESCAPE
```

```
tout_ptr   dd 0                        ;Pointeur sur compteur de Time Out
escdirect  db 0                        ;Pas de Time Out en cas d'Escape

;-- Les variables suivantes mémorisent les anciennes adresses des  ---
;-- interruptions avant la mise en service des nouvelles           ---

int9_ptr   equ this dword              ;ancien vecteur d'interruption 9h
int9_ofs   dw 0                                          ;Offset
int9_seg   dw 0                                          ;Segment

int1C_ptr  equ this dword              ;Ancien vecteur d'unterruption 1Ch
int1C_ofs  dw 0                                          ;Offset
int1C_seg  dw 0                                          ;Segment

;-------------------------------------------------------------------
;-- GETB : lit un octet sur le port d'entrée
;-- Appel depuis TP: getb : BYTE;

getb       proc near

       mov  dx,InPort                          ;Adresse du port en DX
       in   al,dx                                    ;Lit le port
       and  al,0FBh                          ;Enlève les bits 0-2

       ret                                  ;Renvoie le résultat en AL

getb       endp

;-------------------------------------------------------------------
;-- PUTB : émet un octet sur le port de sortie
;-- Appel depuis TP : putb( Wert : BYTE );

putb       proc near

valeur equ byte ptr [bp+4]                 ;Variable transmise par TP

       push bp                             ;Permet d'accéder aux arguments
       mov  bp, sp

       mov al, valeur                          ; Charge la valeur en AL
       mov  dx, OutPort          ;Charge en DX l'adresse du port de sortie
       out  dx, al                           ;Ecrit la valeur dans le port

       pop  bp                                                ;restaure BP
       ret  2                          ;Nettoie la pile et rend la main

putb       endp

;-------------------------------------------------------------------
;-- INTR_INSTALL: Installe les gestionnaires d'interruption
;-- Appel depuis TP : intr_install( escape_flag, timeout_count : ptr );

intr_install  proc near

sframe0    struc                       ;Structure pour accéder à la pile
bp0        dw ?                                        ;mémorise BP
ret_adr0   dw ?                                  ;Adresse de retour
toptr      dd ?                 ;Pointeur FAR sur le compteur de Timeout
escptr     dd ?                   ;Pointeur FAR sur l'indicateur ESCAPE
sframe0    ends                                  ;Fin de la structure

frame  equ [ bp - bp0 ]

       push bp                                 ;mémorise BP sur la pile
       mov  bp,sp                                   ;Transfère SP en BP
       push es                                  ;Sauve ES sur la pile

          ;-- Cherche les arguments sur la pile et les traite ---------

       les  si,frame.escptr      ;Charge un pointeur sur l'indic. ESCAPE
       mov  word ptr key_ptr,si       ;et le transfère dans les variables
       mov  word ptr key_ptr+2,es                ;du segment de code

       les  si,frame.toptr;Charge un pointeur sur le compteur de Timeout
       mov  word ptr tout_ptr,si      ;et le transfère dans les variables
       mov  word ptr tout_ptr+2,es               ;du segment de code

                 ;-- Lit les adresses des gestionnaires à détourner-

       mov  ax,3509h          ;LIT l'adresse du vecteur d'interruption 9h
       int  21h                                                    ;
       mov  int9_ofs,bx                                 ;et la mémorise
       mov  int9_seg,es                                            ;

       mov  ax,351Ch          ;Lit l'adresse du vecteur d'interruption 1Ch
       int  21h                                                    ;
       mov  int1C_ofs,bx                                ;et la mémorise
       mov  int1C_seg,es                                           ;

          ;-- Installe les nouveaux gestionnaires d'interruption ------

       push ds                                ;Stocke le segment de données
       mov  ax,cs                                    ;Rend DS égal à  CS
       mov  ds,ax

       mov  ax,2509h ;Fonction: "Fixer le driver pour l'interruption 9h"
       mov  dx,offset int09   ;DS:DX recueille l'adresse du gestionnaire
       int  21h                              ;déclenche l'interruption de DOS

       mov  ax,251Ch ;Fonction:"Fixer le driver pour l'interruption 1Ch"
       mov  dx,offset int1C   ;DS:DX recueille l'adresse du gestionnaire
       int  21h                              ;déclenche l'interruption de DOS

       pop  ds                                          ;Reprend DS sur la pile
       pop  es                                                    ;Récupère BP
       pop  bp                                  ;et rend la main à l'appelant
       ret  8

intr_install endp

;-------------------------------------------------------------------
;-- INTR_REMOVE: Désactive les gestionnaires d'interruption
;-- Appel depuis TP: intr_remove;

intr_remove proc near

       cli                                             ;Inhibe les interruptions
       push ds                                                 ;Sauvegarde DS

       mov  ax,2509h      ;Fonction: "Fixer le gestionnaire pour INT 9 "
       mov  ds,int9_seg                ;Segment de l'ancien gestionnaire
       mov  dx,int9_ofs                 ;Offset de l'ancien gestionnaire
       int  21h                          ;Rétablit l'ancien gestionnaire

       mov  ax,251Ch      ;Fonction: "Fixer le gestionnaire pour INT 1C"
       mov  ds,int1C_seg               ;Segment de l'ancien gestionnaire
       mov  dx,int1C_ofs                ;Offset de l'ancien gestionnaire
       int  21h                          ;Rétablit l'ancien gestionnaire

       pop  ds                                                 ;Reprend DS
       sti                                 ;Autorise à nouveau les interrruptions

       ret                                              ;retourné à l'appelant

intr_remove endp                                        ;Fin de la procédure

;-------------------------------------------------------------------
;-- EscapeDirect: Détermine si en cas d'Escape il faut déclencher
;--               un Time Out
;-- Appel depuis TP: procedure EscapeDirect( Declenche : boolean );

EscapeDirect proc near

sframe1    struc                            ;Structure d'accès à la pile
bp1        dw ?                                          ;Mémorise BP
ret_adr1   dw ?                                     ;Adresse de retour
escflag    dw ?                                        ;TRUE ou FALSE
sframe1    ends                                     ;Fin de la structure

frame  equ [ bp - bp1 ]

       push bp                                                 ;Empile BP
       mov  bp,sp                                   ;et tranfsère SP en BP

       mov  al,byte ptr frame.escflag                ;Charge l'indicateur
       mov  escdirect.al                ;et le dépose dans une variable CS

       pop  bp
       ret  2                  ;Retourne à l'appelant , enlève l'argument
                                                          ; de la pile

EscapeDirect endp

;-------------------------------------------------------------------
;--          Voici les nouveaux gestionnaires d'interruption          --
;-------------------------------------------------------------------

       assume CS:CODE, DS:nothing, ES:nothing, SS:nothing

;-- nouveau gestionnaire de l'interruption 09h ----------------------

int09      proc far

       push ax                                    ;Sauvegarde AX sur la pile
       in   al,KB_PORT            ;Lit le scan code sur le port du clavier

       cmp  al,128                                       ;Code release ?
       jae  i9_end                                     ;Oui, ne pas tester

       cmp  al,ESCAPE                                  ;Non, est-ce ESCAPE?
       jne  i9_end                          ;Non, passe à l'ancien gestionnaire

          ;-- La touche ESCAPE a été actionnée-----------------------

       push ds                                          ;Met de côté DS et SI
       push si
```

```
;       lds  si,key_ptr         ;Charge un pointeur sur la variable Escape
;       mov  word ptr [si],1                    ;Met à 1 l'indic.  ESCAPE
;       cmp  escdirect,0            ;Faut-il effacer l'indic. Time Out ?
;       je   i9_1                                       ;Non ---> I9_1
;
;       lds  si,tout_ptr  ;Oui, charge un ptr sur le compteur de Time Out
;       mov  word ptr [si],0                    ;Met le compteur à 0
;
;i9_1:  pop  si                                  ;Restaure DS et SI
;       pop  ds
;
;       mov  al,EOI                     ;Indique la fin de l'interruption
;       out  INT_CTR,al
;
;       pop  ax                                         ;Reprend AX
;       iret                            ;et retourne au programme interrompu
;
;i9_end: pop  ax                                        ;Récupère AX
;       jmp  cs:[int9_ptr]          ;Se branche sur l'ancien gestionnaire
;
;int09      endp
;
;;-- Nouveau gestionnaire de l'interruption 1Ch ------------------------
;
;int1C      proc far
;
;       push ds                                         ;Mémorise DS et SI
;       push si
;       lds  si,tout_ptr  ;Charge un pointeur sur le compteur de Time Out
;       cmp  word ptr [si],0                            ;Compteur à 0?
;       je   no_dekr                        ;OUI ---> ne plus décrémenter
;
;       dec  word ptr [si]                              ;Non, décrémenter
;
;no_dekr: pop  si                                ;Restaure DS et SI
;        pop  ds
;
;        jmp  cs:[int1C_ptr]                ;Passe à l'ancien gestionnaire
;
;int1C      endp
;
;;-------------------------------------------------------------------------
;
;CODE   ends                                     ;Fin du segment de code
;       end                                              ;Fin du programme
```

Listing : PLINKC.C

```
/*****************************************************************/
/*                          P L I N K C . C                      */
/*---------------------------------------------------------------*/
/*    Fonction       : transmet des fichiers par le port parallèle  */
/*---------------------------------------------------------------*/
/*    Auteur         : Michael Tischer                          */
/*    Développé le   : 27.09.1991                               */
/*    Dernière MAJ   : 28.11.1991                               */
/*---------------------------------------------------------------*/
/*    Modèle mémoire : SMALL                                     */
/*****************************************************************/

/*-- Fichiers d'inclusion ------------------------------------------*/

#include <stdlib.h>
#include <stdio.h>
#include <conio.h>
#include <dos.h>
#include <setjmp.h>
#include <string.h>
#ifdef __TURBOC__                               /* Compilateur Turbo C ? */
  #include <dir.h>                  /* En-têtes des fonctions de traitement */
  #include <ctype.h>                                /* des répertoires */
#endif

/*-- Définitions de types ------------------------------------------*/

typedef unsigned char BYTE;                             /* Définit un octet */
typedef unsigned int WORD;                          /* Bricolage du type WORD */
typedef struct {                        /* En-tête de transmission des blocs */
                BYTE        Token;
                unsigned int Len;
              } BHEADER;

/*-- Fonctions du module en assembleur -----------------------------*/

extern void IntrInstall( int far  * escape_flag,
                         WORD far * timeout_count );
extern void IntrRemove( void );
extern void EscapeDirect( int ausloesen );

/*-- Constantes ----------------------------------------------------*/

#define UNESEC          18                              /* Une seconde */
#define DIXSEC          182                             /* Dix secondes */
#define TO_DEFAULT      DIXSEC          /* Valeur de Time-Out par défaut */

#define TRUE            ( 0 == 0 )
#define FALSE           ( 0 == 1 )
#define MAXBLOC         4096        /* Taille des blocs 4 Ko ( Cluster ) */

/*-- Constantes pour le protocole de transmission-------------------*/

#define ACK             0x00                    /* Accusé de réception */
#define NAK             0xFF                /* Non-Accusé de réception */
#define MAX_TRY         5           /* Nombre de tentatives avant erreur */

/*-- Tokens pour la communication entre émetteur et récepteur ------*/

#define TOK_DATSTART    0                       /* Début de fichier */
#define TOK_DATNEXT     1                   /* Bloc suivant d'un fichier */
#define TOK_DATEND      2               /* Transmission du fichier terminée */
#define TOK_FIN         3                       /* Terminer le programme */
#define TOK_ESCAPE      4/* Interruption par ESC sur ordinateur distant */

/*-- Codes pour les appels LongJump --------------------------------*/

#define LJ_OKEMET       1       /* Tous les fichiers correctement émis */
#define LJ_OKRECEPT     2       /* Tous les fichiers correctement reçus */
#define LJ_TIMEOUT      3   /* Time-out: le correspondant ne répond pas */
#define LJ_ESCAPE       4/*Interruption par Escape sur ordinateur local */
#define LJ_REMESCAPE    5 /* Interruption Escape sur ordinateur distant */
#define LJ_DATA         6                   /* Erreur de communication */
#define LJ_NOLINK       7                           /* Pas de liaison */
#define LJ_NOPAR        8                           /* Pas d'interface */
#define LJ_PARA         9               /* Paramètres d'appel invalides */

/*-- Macros --------------------------------------------------------*/
/*-- Les trois bits inférieurs du registre d'entrée ne sont pas    --*/
/*-- utilisés selon les systèmes ils peuvent être à 1 ou 0 , ici   --*/
/*-- ils sont éliminés par GetB() .                                --*/

#ifdef __TURBOC__                       /* Compilateur Turbo C ? */
  #define GetB()                        ( inportb( PortdEntree ) & 0xF8 )
  #define PutB( QuelqueChose )          outportb( PortdeSortie, Quelque-
 Chose )
  #define DIRSTRUCT                     struct ffblk
  #define FINDFIRST( path, buf, attr )  findfirst( path, buf, attr )
  #define FINDNEXT( buf )               findnext( buf )
  #define NomFichier                    ff_name
#else                                   /* Non , Microsoft C */
  #define GetB()                        ( inp( PortdEntree ) & 0xF8 )
  #define PutB( QuelqueChose )                  outp( PortdeSortie,
 QuelqueChose )
  #define DIRSTRUCT                     struct        find_t
  #define FINDFIRST( path, buf, attr )  _dos_findfirst(path, attr, buf)
  #define FINDNEXT( buf )               _dos_findnext( buf )
  #define NomFichier                    name
#endif

#ifdef MK_FP                            /* Macro MK_FP déjà définie ? */
  #undef MK_FP                                  /* Oui on l'efface */
#endif

#define MK_FP(seg,ofs) ((void far *) ((unsigned long) (seg)<<16|(ofs)))

/*-- Variables globales --------------------------------------------*/

int     PortdEntree;                    /* Adresse du port d'entrée */
int     PortdeSortie;                   /* Adresse du port de sortie */
int     Escape = 0;                 /* Pas de touche ESCAPE enfoncée */
WORD    Timeout = TO_DEFAULT;               /* Valeur de Time Out */
WORD    TO_Count;                       /* Compteur de Timeout */
jmp_buf Branchement;                /* Adresse de retour pour terminer */
```

```
!BYTE    *BlocBuf;                    /* Buffer de mémorisation d'un bloc */
!FILE    *Fichier = NULL;  /* Variable fichier pour traiter un fichier */
!
!/*****************************************************************/
!/* GetPortAdr    : Initialise les adresses des ports d'une interface */
!/*                 parallèle à l'aide des variables PortdEntrée et   */
!/*                 Portde Sortie                                     */
!/* Entrée        : NUMMER = Numéo de l'interface parallèle (1-4)     */
!/* Sortie        : TRUE, si l'interface est valable                  */
!/* Var. globales : PortdEntree/W, PortdeSortie/W                     */
!/* Info          : Les adresses de base des interfaces parallèles    */
!/*                 (en nombre de 1 à 4) se trouvent dans les mots    */
!/*                 mémoires commençant en 0040:0008                  */
!/*****************************************************************/
!
!int GetPortAdr( int Numero )
!{  /* Lit les adresses des ports dans le segment de variables du BIOS */
! PortdeSortie = *( WORD far * ) MK_FP( 0x0040, 6 + Numero * 2 );
! if ( PortdeSortie != 0 )                  /* Interface disponible ? */
! {                                                          /* Oui */
!   PortdEntree = PortdeSortie + 1;/* Adresse pour le registre d'entrée*/
!   return TRUE;                                  /* Pas d'erreur */
! }
! else
!   return FALSE;                          /* Erreur: interface absente */
!}
!
!/*****************************************************************/
!/* Port_Init     : Initialise les registres nécessaires à la         */
!/*                 transmission                                      */
!/* Entrée        : EMETTEUR=TRUE, si émetteur, FALSE, si récepteur   */
!/* Sortie        : TRUE, si les reg. ont été correctement initialisés */
!/* Var. globales : PortdEntree/R, PortdeSortie/R                     */
!/* Info          : La dissymétrie: envoie 00010000, attend 00000000  */
!/*                 est rendue nécessaire par l'inversion du signal.  */
!/*                 Normalement les registres d'entrée et de sortie   */
!/*                 contiennent les valeurs souhaitées mais           */
!/*                 l'initialisation est nécessaire lorsqu'on reprend */
!/*                 une transmission interrompue                      */
!/*****************************************************************/
!
!int Port_Init( int Emetteur )
!{
! EscapeDirect( TRUE );              /* Déclenche un Time Out si Escape */
! if ( Emetteur )                      /* L'appareil est-il émetteur ? */
! {
!   TO_Count = Timeout * 5;        /* Initialise le compteur de Time Out */
!   PutB( 0x10 );                                /* Envoie : 00010000b */
!   while ( ( GetB() != 0x00 ) && TO_Count )     /* Attend : 00000000b */
!     ;
! }
! else                                      /* L'appareil est récepteur */
!  {
!   TO_Count = Timeout * 5;        /* Initialise le compteur de Time Out */
!   while ( ( GetB() != 0x00 ) && TO_Count )     /* Attend : 00000000b */
!     ;
!   PutB( 0x10 );                                /* Envoie : 00010000b */
!  }
! EscapeDirect( FALSE );                 /* Si Escape pas de Timeout */
! return ( TO_Count != 0 );               /* Initialisation terminée */
!}
!
!/*****************************************************************/
!/* EmetOctet     : Envoie un octet en deux parties à l'ordinateur    */
!/*                 distant et teste le résultat                      */
!/* Entrée        : VALEUR: octet à émettre                           */
!/* Sortie        : Transmission correcte ? ( 0 = Erreur , -1 = ok )  */
!/* Var. globales : Timeout/R, PortdEntrer/R, PortdeSortie/R (macros) */
!/*****************************************************************/
!
!int EmetOctet( BYTE Valeur )
!{
! BYTE Retour;                                    /* Octet réceptionné */
!
!/*-- Emet le quartet inférieur ---------------------------------------*/
!
! TO_Count = Timeout;                /* Initialise le compteur de Timeout */
! PutB( Valeur & 0x0F );                 /* Envoi avec mise à 0 de BUSY */
! while ( ( ( GetB() & 128 ) == 0 ) && TO_Count )  /* Attend le retour */
!   ;
! if ( TO_Count == 0 )                             /* Erreur de Timeout ?*/
!   longJmp( Branchement, LJ_TIMEOUT );  /* Interrompt la transmission */
!
! Retour = ( GetB() >> 3 ) & 0x0F;                 /* Bits 3-6 en  0-3 */
!
!/*-- Emet le quartet supérieur ---------------------------------------*/
!
! TO_Count = Timeout;                /* Initialise le compteur de Timeout */
! PutB( ( Valeur >> 4 ) | 0x10 );        /* Envoi avec mise à 1 de BUSY */
! while ( ( ( GetB() & 128 ) != 0 ) && TO_Count )  /* Attend le retour */
!   ;
!
! if ( TO_Count == 0 )                             /* Erreur de Timeout ? */
!   longJmp( Branchement, LJ_TIMEOUT );  /* Interrompt la transmission */
!
! Retour = Retour | ( ( GetB() << 1 ) & 0xF0 );    /* Bits 3-6 en 4-7 */
! return ( Valeur == Retour );          /* Octet correctement transmis ? */
!}
!
!/*****************************************************************/
!/* RecOctet      : Réceptionne un octet en deux parties de la part   */
!/*                 d'un ordinateur distant et renvoie les parties    */
!/*                 pour vérification                                 */
!/* Entrée        : néant                                             */
!/* Sortie        : Octet reçu                                        */
!/* Var. globales : Timeout/R, PortdEntree/R, PortdeSortie/R (Macros) */
!/*****************************************************************/
!
!BYTE RecOctet( void )
!{
! BYTE LoNib, HiNib;                                 /* Quartets reçus */
!
!     /*-- Réceptionne le quartet inférieur et le renvoie -------------*/
!
! TO_Count = Timeout;                /* Initialise le compteur de Timeout */
! while ( ( ( GetB() & 128 ) == 0 ) && TO_Count )/* Attend que BUSY =1 */
!   ;
!
! if ( TO_Count == 0 )                             /* Erreur de Timeout ?*/
!   longJmp( Branchement, LJ_TIMEOUT );  /* Interrompt la transmission */
!
! LoNib = ( GetB() >> 3 ) & 0x0F;                  /* Bits 3-6 dans 0-3 */
! PutB( LoNib );                                          /* Renvoie */
!
!/*-- Réceptionne le quartet supérieur et le renvoie -------------------*/
!
! TO_Count = Timeout;                /* Initialise le compteur de Time Out */
! while ( ( ( GetB() & 128 ) != 0 ) && TO_Count ) /* Attend que BUSY 0 */
!   ;
!
! if ( TO_Count == 0 )                             /* Erreur de Timeout ? */
!   longJmp( Branchement, LJ_TIMEOUT );  /* Interrompt la transmission */
!
! HiNib = ( GetB() << 1 ) & 0xF0;                  /* Bits 3-6 en 4-7 */
! PutB( ( HiNib >> 4 ) | 0x10 );          /* Renvoie et met Busy à 1 */
!
! return( LoNib | HiNib );                         /* Octet renvoyé */
!}
!
!/*****************************************************************/
!/* EmetBloc  : Envoi un bloc de données                              */
!/* Entrées   : TOKEN  = Commande pour le récepteur                   */
!/*             NOMBRE = Nombre d'octets  à transmettre               */
!/*             DONNEES= Pointeur sur le buffer des données           */
!/* Sortie    : néant, en cas d'erreur branchement par                */
!/*             LongJmp à la routine de traitement d'erreur           */
!/*****************************************************************/
!
!void EmetBloc( BYTE Token, int Nombre, void *Donnees )
!{
! BHEADER header;              /* En-tête pour mémoriser Token et Nombre */
! BYTE    *bptr,               /* Pointe sur l'octet courant à émettre */
!     RecEscape;       /* A-t-on tapé ESCAPE sur l'ordinateur distant ? */
! int     ok,                                    /* Indicateur d'erreur */
!     i,                                      /* Compteur d'itérations */
!     try;                                  /* Nombre d'essais restants */
!
! if ( Escape )                /* A-t-on tapé Escape sur cet ordinateur ? */
! {
!  Token = TOK_ESCAPE;                        /* Oui envoie le token Escape */
!  Nombre = 0;
! }
!
!     /*-- Emission de l'en-tête -------------------------------------*/
!
! header.Token = Token;                            /* Construit l'en-tête */
! header.Len = Nombre;
!
! for ( try = MAX_TRY; try; --try )   /* Au maximum MAX_TRY tentatives */
! {
!  ok = TRUE;                          /* A priori la transmission est bonne */
!  for ( bptr = (BYTE *) &header, i = sizeof( header); i; --i )
!    ok = ok & EmetOctet( *bptr++ );                 /* Emet un octet */
!
!  ok = ok & EmetOctet( (BYTE) (ok ? ACK : NAK) );     /*Confirmation */
!  if ( ok )                                        /* Transmission correcte ? */
!    break;                                  /* Oui, pas d'autre tentative */
! }
!
! if ( try == 0 )                  /* L'en-tête a-t-il pu être transmis ? */
!   longJmp( Branchement, LJ_DATA );     /* Non, arrêter la transmission */
!
! if ( Token == TOK_ESCAPE )           /* A-t-on envoyé l'avis d'ESCAPE ? */
!   longJmp( Branchement, LJ_ESCAPE );  /* Oui, arrêter la transmission */
!
!   /*-- Emission du bloc de données proprement dit ------------------*/
```

```
 if ( Nombre )                                   /* Taille diff de 07 */
 {
  for ( try = MAX_TRY; try; -- try )/* MAX_TRY tentatives au maximum */
  {
   ok = TRUE;                  /* A priori la transmission est bonne */
   for ( bptr = (BYTE *) Donnees, i = Nombre; i; --i )
    ok = ok & EmetOctet( *bptr++ );/*Envoie octet et interroge état*/

   ok = ok & EmetOctet( (BYTE) (ok ? ACK : NAK) );  /* Confirmation */
   if ( ok )                                  /* Transmission correcte ? */
    break;                                  /* Oui pas d'autre tentative */
  }
  if ( try == 0 )          /* Les données ont-elle été transmises ? */
   longjmp( Branchement, LJ_DATA );  /* Non, interrompt transmission */
 }

 /*-- Teste l'octet ESCAPE du récepteur ----------------------------*/

 for ( try = MAX_TRY; try; -- try )                /* Nombre de tentatives */
 {
  RecEscape = RecOctet();                    /* Détecte un escape distant */
  if ( RecEscape == (BYTE) TRUE  ||  RecEscape == (BYTE) FALSE )
   break;                                   /* Etat de la touche Escape reçu */
 }
 if ( try == 0 )      /* L'état de la touche Escape a-t-il été reçu ? */
  longjmp( Branchement, LJ_DATA ); /* Non, interrompt la transmission */

 if ( RecEscape )                        /* Escape sur ordinateur distant ? */
  longjmp( Branchement, LJ_REMESCAPE );/* Oui interrompt transmission */
}

/*********************************************************************/
/* RecBloc : Réceptionne un bloc de données                          */
/* Entrée  : TOKEN  = Pointeur sur la variable qui mémorise le token */
/*           LEN    = Pointeur sur la variable qui mémorise la long. */
/*           DONNEES = Pointeur sur le buffer des données            */
/* Sortie  : néant, en cas d'erreur branchement sur routine d'erreur */
/*           par LongJmp                                             */
/* Info    : Le buffer transmis doit prévoir de la place pour MAXBLOC */
/*           pour MAXBLOC octets, la taille du bloc ne pouvant pas   */
/*           être anticipée                                          */
/*********************************************************************/

void RecBloc( BYTE *Token, int *Len, void *Donnees )
{
 BHEADER header;                                 /* Mémorise l'en-tête */
 BYTE    *bptr;       /* Pointeur courant dans le buffer de réception */
 int     ok,                                      /* Indicateur d'erreur */
     i,                                        /* Compteur d'itérations */
     try,                                /* Nombre de tentatives restantes */
     EscapeStatus;     /* Mémorise l'état courant de la touche Escape */

 /*-- Réceptionne d'abord l'en-tête --------------------------------*/

 for ( try = MAX_TRY; try; -- try )  /* MAX_TRY tentatives au maximum */
 {
  for ( bptr = (BYTE *) &header, i = sizeof(header); i; --i )
    *bptr++ = RecOctet( );

  if ( RecOctet() == ACK );           /* Tous les octets bien reçus ? */
    break;                                  /* Oui pluss de tentative */
 }

 if ( try == 0 )                          /* En-tête correctement reçu ? */
  longjmp( Branchement, LJ_DATA ); /* Non, interrompre la transmission*/

 if ( ( *Token = header.Token ) == TOK_ESCAPE )   /* Emetteur ESCAPE? */
  longjmp( Branchement, LJ_REMESCAPE );/* Oui interrompt transmission */

     /*-- L'en-tête est bon, il faut passer au bloc des données . ---*/

 if ( ( *Len = header.Len ) != 0 )       /* Pas de bloc des données ? */
 {                                                          /* si */
  for ( try = MAX_TRY; try; -- try )/* MAX_TRY tentatives au maximum */
  {
   for ( bptr = (BYTE *) Donnees, i = header.Len; i; --i )
     *bptr++ = RecOctet( );

   if ( RecOctet() == ACK );         /* Tous les octets bien reçus ? */
    break;                                  /* Oui, plus de tentative */
  }

  if ( try == 0 )          /* Le bloc a-t-il été correctement reçu ? */
   longjmp( Branchement, LJ_DATA );/* Non, interrompt la transmission */
 }

 /*-- Envoie l'état actuel de la touche Escape à l'ordinateur distant */

 EscapeStatus = Escape;                          /* Mémorise l'état */
 for ( try = MAX_TRY; try; -- try )        /* Nombre de tentatives   */
 {
  if ( EmetOctet( (BYTE) (EscapeStatus != 0) ) )          /* arrivé ? */
   break;                          /* Oui, plus de tentative nécessaire */
 }

 if ( try == 0 )                  /* L'état ESC a-t-il pu être envoyé ? */
  longjmp( Branchement, LJ_DATA );/* Non, interrompt la transmission */

 if ( EscapeStatus )      /* A-t-on actionné Esc sur cet ordinateur ? */
  longjmp( Branchement, LJ_ESCAPE );/*Oui, interrompt la transmission */
}

/*********************************************************************/
/* EmetFichier : Envoie un fichier                                   */
/* Entrée      : NOM = Ptr sur un buffer contenant le nom du fichier */
/* Sortie      : néant                                               */
/*********************************************************************/

void EmetFichier( char *Nom )
{
 int           Status;                          /* Etat d'émission */
 WORD          Lus;                           /* Nombre d'octets lus */
 unsigned long Taille;                      /* Nombre d'octets envoyés */

 printf( "%-13s", Nom);
 Fichier = fopen( Nom,"rb" );                      /* Ouvre le fichier */
 EmetBloc( TOK_DATSTART, strlen(Nom)+1, Nom );      /* envoie son nom*/

        /*-- Transfère le contenu du fichier ----------------------*/

 Taille = 0;
 do
 {
  Lus = fread( BlocBuf, 1, MAXBLOC, Fichier );       /* Lit un bloc  */
  if ( Lus > 0 )                              /* Est-on à la fin ? */
  {                                                        /* Non */
   EmetBloc( TOK_DATNEXT, Lus, BlocBuf );          /* Emet le bloc */
   Taille += Lus;
   printf( "\r%-13s (%ld)", Nom, Taille );
  }
 }
 while (  Lus > 0 );
 printf( "\n" );

 EmetBloc( TOK_DATEND, 0, NULL );          /* Clôture la transmission */

 fclose( Fichier );                          /* Referme le fichier */
 Fichier = NULL;                               /* Fichier fermé */
}

/*********************************************************************/
/* RecFichier  : Réceptionne un fichier                              */
/* Entrée      : néant                                               */
/* Sortie      : Dernier token reçu                                  */
/*********************************************************************/

int RecFichier( void )
{
 int           Status;                          /* Etat de réception */
 WORD          AEnregistrer;                /* Taille du dernier bloc */
 unsigned long Taille;
 BYTE          Token;                           /* Token réceptionné */
 int           Len;                          /* Longueur du bloc reçu */
 char          Name[13];                        /* Nom du fichier */

 RecBloc( &Token, &Len, BlocBuf );
 if ( Token == TOK_DATSTART )
 {
  strcpy( Name, BlocBuf );
  Fichier = fopen( Name, "wb" );          /* Ouvre (crée) le fichier */
  printf( "%-13s", Name );

  /*-- Réceptionne le contenu du fichier ----------------------------*/

  Taille = 0;
  do
  {
   RecBloc( &Token, &Len, BlocBuf );         /* Réceptionne un bloc */
   if ( Token == TOK_DATNEXT )      /* Bloc de données consécutif ? */
   {                                                       /* Je */
    fwrite( BlocBuf, 1, Len, Fichier );             /* Enregistre */
    Taille += Len;
    printf( "\r%-13s (%ld)", Name, Taille );
   }
  }
  while ( Token == TOK_DATNEXT );
  fclose( Fichier );                          /* Referme le fichier */
  Fichier = NULL;                            /* Le fichier est fermé */
  printf( "\n" );
 }
 return Token;                             /* Retourne l'état d'erreur */
}
```

```
/*****************************************************************/
/*                        PROGRAMME PRINCIPAL                    */
/*****************************************************************/

void main( int argc, char *argv[] )
{
 DIRSTRUCT SRec;    /* Structure pour la recherche dans un répertoire */
 BYTE      Emetteur;  /* Mode de transmission (Emission, Réception ) */
 int       sjStatus,                              /* Code de longjmup */
           Numero,                              /* Numéro de l'interface */
           i,                                   /* Compteur d'itérations */
           Trouve;                        /* Pour la recherche des fichiers */

 static char *Avis[ 9 ] =
  { "FIN: Tous les fichiers ont été correctement émis",
    "FIN: Tous les fichiers ont été correctement reçus",
    "ERREUR: Time-Out, le système distant ne répond pas",
    "FIN: Interruption par Escape.",
    "FIN: Interruption par Escape sur l'ordinateur distant.",
    "ERREUR: Interface ou câble défectueux",
    "ERREUR: pas de contact avec l'ordinateur distant",
    "ERREUR: L'interface indiquée n'existe pas !",
    "ERREUR: Paramètre inconnu ou invalide !" };

 printf( "\n\nTransmission de données par l'interface parallèle    " );
 printf( "(c) 1991 by Michael Tischer\n" );
 printf( "                                                    " );
 printf( "                                \n\n" );

 if ( strcmp( argv[ 1 ], "?" ) == 0 )          /* Affiche la syntaxe ? */
 {                                                          /* oui*/
  printf( "Appel: plinkc [/Pn] [/Tnn] [Nom de fichier]\n" );
  exit ( 0 );
 }

 sjStatus = setjmp( Branchement );                  /* Adresse de retour */
 if ( sjStatus )                              /* Longjmp a-t-il été appelé ? */
 {                                                          /* Oui */
  IntrRemove( );            /* Désactive le gestionnaire d'interruption */
  if ( Fichier )                    /* Reste-t-il un fichier ouvert ? */
    fclose( Fichier );
  free( BlocBuf );                  /* Libère la place mémoire allouée */
  printf( "\n\r%s\n", Avis[ sjStatus - 1 ] );
  exit( 0 );
 }

 BlocBuf = malloc( MAXBLOC );       /* Crée un buffer pour les données */
 IntrInstall( &Escape, &TO_Count );/* Initialise driver interruption*/
 /*-- Fixe les paramètres par défaut et exploite la ligne de commande */

 Emetteur = FALSE;              /* Par défaut l'ordinateur est récepteur */
 Numero = 1;                                /* L'interface est LPT1 */

 for ( i = 1; i < argc; i++ )
 {
  if ( argv[i][0] == '/' )                                /* Paramètres */
  {
   switch ( toupper( argv[i][1] ) )
   {
    case 'T' : Timeout = (atol( &argv[i][2] ) * DIXSEC) / 10;
               if ( Timeout == 0 )
          longjmp( Branchement, LJ_PARA );              /* Incorrect */
               break;
    case 'P' : Numero = argv[i][2] - 48;                 /* Interface */
               if ( Numero == 0   ||   Numero > 4 )
          longjmp( Branchement, LJ_PARA );              /* Incorrect */
               break;
    default  : longjmp( Branchement, LJ_PARA );/* paramètre inconnu*/
               break;
   }
   argv[i][0] = '\0';                           /* Efface l'argument */
  }
  else         /* Pas de paramètre, il doit s'agir du nom du fichier */
   Emetteur = TRUE;                                    /* Emetteur */
 }

/*-- Lance la transmission ---------------------------------------*/

 if ( GetPortAdr(Numero) == FALSE )    /* L'interface existe-t-elle ? */
  longjmp( Branchement, LJ_NOPAR );                    /* Non, erreur */

 if ( Port_Init( Emetteur ) == FALSE )          /* Etablit la liaison */
  longjmp( Branchement, LJ_NOLINK );          /* Erreur, impossibilité */

 if ( Emetteur )                                          /* Emetteur? */
 {
  printf( "Emission vers LPT%d:\n\n", Numero );

       /*-- Transmet tous les fichiers ----------------------------*/

  for ( i = 1; i < argc; i++ )       /* Parcourt la ligne de commande */
  {
   if ( argv[i][0] != '\0' )                        /* Nom du fichier ? */
   {                                                          /* Oui*/
    Trouve = FINDFIRST( argv[i], &SRec, 0);
    while ( !Trouve )
    {
     if ( SRec.NomFichier[0] != '.' )
     EmetFichier( SRec.NomFichier );              /*Fichier transmis */
      Trouve = FINDNEXT( &SRec );
    }
   }
  }
  EmetBloc( TOK_FIN, 0 , NULL );           /* Tous les fichiers émis  */
  longjmp( Branchement, LJ_OKEMET );
 }
 else                                                     /* Non, récepteur  */
 {
  printf( "Réception sur LPT%d:\n\n", Numero );
  while ( RecFichier() != TOK_FIN )       /* Réceptionne les fichiers */
   ;                                        /* jusqu'au token final */
  longjmp( Branchement, LJ_OKRECEPT );
 }
}
```

Listing : PLINKCA.ASM

```
;*****************************************************************;
;*                      P L I N K C A . A S M                    *;
;*---------------------------------------------------------------*;
;*   Fonction        : Complément en assembleur du programme en C  *;
;*                     PLINKC. Contient les gestionnaires d'interrup *;
;*                     de ce programme                           *;
;*---------------------------------------------------------------*;
;*   Auteur          : MICHAEL TISCHER                           *;
;*   Développé le    : 10.10.1991                                *;
;*   Dernière MAJ    : 11.10.1991                                *;
;*---------------------------------------------------------------*;
;*   Modèle mémoire : SMALL                                      *;
;*---------------------------------------------------------------*;
;*   Assemblage      : MASM PLINKCA;  ou TASM PLINKCA            *;
;*                     ... puis lier à PLINKC                    *;
;*****************************************************************;

IGROUP group _text                  ;Regroupe les segments de programme
DGROUP group _bss, _data            ;Regroupe les segments de données
       assume CS:IGROUP, DS:DGROUP, ES:DGROUP, SS:DGROUP

_BSS   segment word public 'BSS';Segment réservé aux variables statiques
_BSS   ends                                          ;non initialisées

_DATA  segment word public 'DATA';Segment réservé aux variables globales
                                                     ;et statiques initialisées

_DATA  ends

;=== Constantes ================================================

KB_PORT   = 60h                                      ;Port du clavier
INT_CTR   = 20h                          ;Port du contrôleur d'interruption
EOI       = 20h                              ;Commande fin d'interruption
ESCAPE    = 1                               ;Scan code de la touche Escape

;=== Programme ==================================================

_TEXT  segment byte public 'CODE'                    ;Segment de code

;-- Déclarations publiques de fonctions internes ----------------

public    _IntrInstall    ;Permet des appels à partir du programme en C
public    _IntrRemove
public    _EscapeDirect

;-- Variables pour les gestionnaires d'interruption-------------
;-- (accessibles uniquelent par le segment de code )------------
```

```
;
key_ptr    dd 0                    ;Pointeur sur la variable pour ESCAPE
tout_ptr   dd 0                    ;Pointeur sur compteur de Time Out
escdirect  db 0                    ;Pas de Time Out en cas d'Escape

;-- Les variables suivantes mémorisent les anciennes adresses des - ---
;-- gestionnaires d'interruption remplacés par les nouveaux    ---

int9_ptr   equ this dword          ;Ancien vecteur d'interruption 9h
int9_ofs   dw 0                                          ;Offset
int9_seg   dw 0                                          ;Segment

int1C_ptr  equ this dword          ;ancien vecteur d'interruption 1Ch
int1C_ofs  dw 0                                          ;Offset
int1C_seg  dw 0                                          ;Segment

;-------------------------------------------------------------------
;-- IntrInstall : Installe les gestionnaires d'interruption      ----
;-- Appel depuis C: void IntrInstall( int far * escape_flag,
;--                                   word far * timeout_count ):

_IntrInstall  proc near

sframe0    struc                        ;Structure d'accès à la pile
bp0        dw ?                                    ;Mémorise BP
ret_adr0   dw ?                                 ;ADresse de retour
escptr     dd ?                 ;Pointeur FAR sur l'indic ESCAPE
toptr      dd ?              ;Pointeur FAR sur le compteur de Time Out
sframe0    ends                                  ;Fin de la structure

frame  equ [ bp - bp0 ]

       push bp                                           ;Empile BP
       mov  bp,sp                                ;Transfère SP  en BP

       push si

              ;-- Prend les arguments sur la pile et les traite-------

       les  si,frame.escptr       ;Charge un pointeur sur l'indic ESCAPE
       mov  word ptr key_ptr,si       ;et le transfère en segment de code
       mov  word ptr key_ptr+2,es                                    ;

       les  si,frame.toptr    ;Charge un ptr sur le compteur de Time Out
       mov  word ptr tout_ptr,si      ;et le transfère en segment de code
       mov  word ptr tout_ptr+2,es                                   ;

          ;-- Lit les adresses des gestionnaires à détourner holen --

       mov  ax,3509h                    ;Lit le vecteur d'interruption 9h
       int  21h                                                      ;
       mov  int9_ofs,bx           ;et mémorise l'adresse du gestionnaire
       mov  int9_seg,es                                              ;

       mov  ax,351Ch                    ;Lit le vecteur d'interruption 1Ch
       int  21h                                                      ;
       mov  int1C_ofs,bx          ;et mémorise l'adresse du gestionnaire
       mov  int1C_seg,es                                             ;

              ;-- Installe les nouveaux gestionnaires d'interruption

       push ds                             ;Mémorise le segment de données
       mov  ax,cs                                  ;Rend DS égal à CS
       mov  ds,ax

       mov  ax,2509h      ;Fonction: "Fixer le gestionnaire pour INT 9h"
       mov  dx,offset int09   ;DS:DX recueille l'adresse du gestionnaire
       int  21h                          ;Déclenche l'interruption de DOS

       mov  ax,251Ch         ;Fonction: "Fixer le gestionnaire pour 1Ch"
       mov  dx,offset int1C   ;DS:DX recueille l'adresse du gestionnaire
       int  21h                          ;Déclenche l'interruption de DOS

       pop  ds                                     ;Reprend DS sur la pile

       pop  si
       pop  bp
       ret                                         ;Retourne à l'appelant

_IntrInstall  endp

;-------------------------------------------------------------------
;-- IntrRemove : Désactive le gestionnaire d'interruption
;-- Appel depuis C: void IntrRemove( void );

_IntrRemove  proc near

       cli                                         ;Inhibe les interruptions
       push ds                                          ;Met de côté DS

       mov  ax,2509h         ;Fonction"Fixer le gestionnaire pour INT 9"
       mov  ds,int9_seg              ;Segment de l'ancien gestionnaire
       mov  dx,int9_ofs                 ;Offset de l'ancien gestionnaire
       int  21h                                  ;Rétablit l'ancien gestionnaire

       mov  ax,251Ch          ;Fonction:"Fixer le gestionnaire pour 1Ch"
       mov  ds,int1C_seg             ;Segment de l'ancien gestionnaire
       mov  dx,int1C_ofs                ;Offset de l'ancien gestionnaire
       int  21h                          ;Ratablit l'ancien gestionnaire

       pop  ds                                     ;Reprend DS sur la pile
       sti                                ;Autorise à nouveau les interruptions

       ret                                         ;Retourne à l'appelant

_IntrRemove  endp                                  ;Fin de la procédure

;-------------------------------------------------------------------
;-- EscapeDirect: Détermine si en cas d'Escape il faut déclencher  ----
;--               un Time Out
;-- Appel depuis C: void EscapeDirect( int Declenche ):

_EscapeDirect proc near

sframe1    struc                        ;Structure pour accéder à la pile
bp1        dw ?                                    ;mémorise BP
ret_adr1   dw ?                                 ;ADresse de retour
escflag    dw ?                                  ;TRUE ou FALSE
sframe1    ends                                  ;Fin de la structure

frame  equ [ bp - bp1 ]

       push bp                                           ;Empile BP
       mov  bp,sp                                ;Transfère SP en BP

       mov  al,byte ptr frame.escflag          ;Charge l'indicateur
       mov  escdirect,al                 ;et le transfère en variable CS

       pop  bp
       ret                                         ;Retourne à l'appelant

_EscapeDirect endp

;-------------------------------------------------------------------
;-- Voici les nouveaux gestionnaires d'interruption-----------------
;-------------------------------------------------------------------

       assume CS:IGROUP, DS:nothing, ES:nothing, SS:nothing

;-- Nouveau gestionnaire de l'interruption 9h ----------------------

int09  proc far

       push ax                                          ;Sauvegarde AX
       in   al,KB_PORT           ;Lit le scan code sur le port du clavier

       cmp  al,128                                      ;Code release?
       jae  i9_end                                     ;Oui, ne pas tester

       cmp  al,ESCAPE                                   ;Non, est-ce ESCAPE?
       jne  i9_end                          ;Non, passe à l'ancien gestionnaire

  ;-- La touche ESCAPE a été actionnée ----------------------------

       push ds                                          ;Mémorise DS et SI
       push si
       lds  si,key_ptr          ;Charge un pointeur sur la variable ESCAPE
       mov  word ptr [si],1                    ;Met à 1 l'indicateur ESCAPE
       cmp  escdirect,0               ;Effacer l'indicateur de Time-Out ?
       je   i9_1                                              ;Non ---> i9_1

       lds  si,tout_ptr  ;Oui, charge un ptr sur le compteur de Time Out
       mov  word ptr [si],0                             ;Met le compteur à 0

i9_1:  pop  si                                          ;Restaure DS et SI
       pop  ds

       mov  al,EOI                        ;Indique la fin  de l'interruption
       out  INT_CTR,al

       pop  ax                                          ;Récupère AX
       iret                                  ;Retourne au programme interrompu

i9_end: pop  ax                                         ;Récupère AX
       jmp  cs:[int9_ptr]              ;Se branche sur l'ancien gestionnaire

int09      endp

;-- Nouveau gestionnaire de l'interruption 1Ch ----------------------

int1C      proc far

       push ds                                          ;mémorise DS et SI
       push si
```

```
        lds  si,tout_ptr  ;Charge un pointeur sur le compteur de Time Out
        cmp  word ptr [si],0                    ;Compteur déjà à 0?
        je   no_dekr                    ;OUI ---> ne plus décrémenter

        dec  word ptr [si]                      ;Non, décrémenter

no_dekr: pop  si                        ;Restaure DS et SI
        pop  ds

        jmp  cs:[int1C_ptr]             ;Passe à l'ancien gestionnaire

int1C   endp

;-------------------------------------------------------------------------

_text   ends                            ;Fin du segment de code
        end                             ;Fin du programme
```

8. L'interface série

Vous savez certainement que les ordinateurs peuvent communiquer entre eux et échanger des données à travers le monde entier. Ils utilisent généralement à cet effet le réseau téléphonique ordinaire, qui ne permet pas une transmission des données très rapide mais qui présente l'avantage considérable de permettre d'atteindre pratiquement n'importe quel coin du globe. Les données sont transférées de façon série, c'est-à-dire bit par bit. Un protocole de transmission bien précis doit être respecté pour que l'émetteur et le récepteur se comprennent.

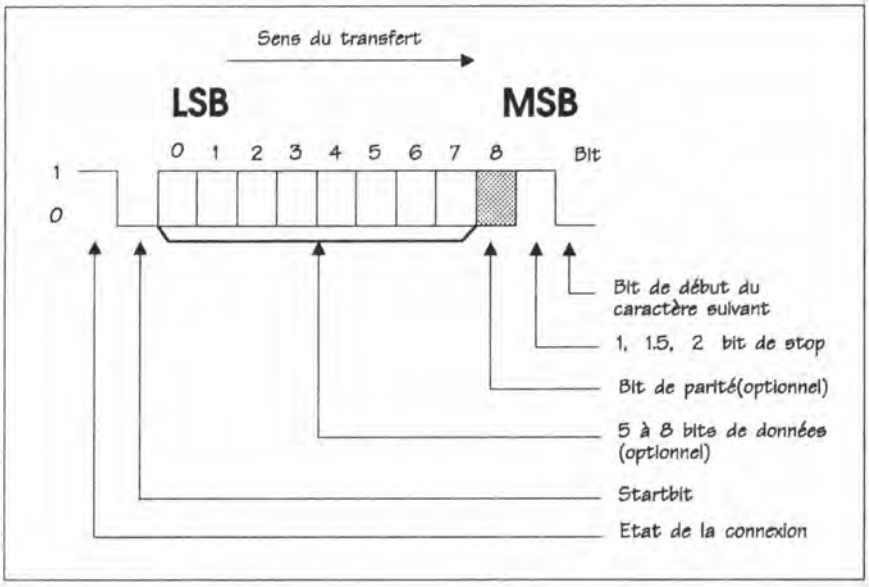
Carte série

Comme le PC n'est pas prévu pour ce type de transmission des données dans sa configuration de base, cette transmission n'est possible qu'en implantant une carte RS232.

Cette carte est désignée, dans la terminologie officielle d'IBM, sous le nom d'adaptateur pour la transmission asynchrone de données.

Avec cette carte, il est possible de réaliser une transmission de données directe entre deux ordinateurs reliés par un câble ou bien une transmission indirecte à travers le câble du téléphone. Dans ce dernier cas cependant, on aura besoin aussi bien du côté émetteur que du côté récepteur de ce qu'on appelle un modem ou coupleur acoustique, c'est-à-dire d'un appareil permettant de convertir les signaux électroniques de l'ordinateur en signaux acoustiques qui puissent être transmis par téléphone.

La communication des données exige non seulement un certain matériel mais aussi un logiciel approprié, qui décharge l'utilisateur du travail complexe de gestion de la carte RS232. Ce logiciel est fourni par le BIOS sous forme de quatre fonctions appelées à travers l'interruption 14h. Avant toutefois de décrire en détail ces fonctions, il convient que nous examinions de plus près le protocole précis employé pour la transmission des données.



Le protocole de transmission asynchrone

Longueur de mot

Comme vous le montre la figure précédente, ce protocole ne prévoit que deux états significatifs, 0 et 1 ou encore low et high. Lorsqu'aucun caractère n'est transmis, le canal est high. Si ce canal passe ensuite à low, le récepteur sait alors que des données vont maintenant être transmises. Suivant la convention appliquée, 5 à 8 bits sont alors envoyés à travers le canal. Les fonctions du BIOS soutiennent malheureusement uniquement une largeur de données de 7 ou 8 bits. Si le canal passe à low lors de la transmission, cela signifie que le bit transmis vaut 0, alors que l'état high signale un bit mis. C'est toujours le bit de plus faible poids de chaque caractère qui est transmis en premier et le bit de plus fort poids en dernier.

Parité

Le caractère peut être suivi de ce qu'on appelle un bit de parité et qui sert à détecter des erreurs éventuelles lors de la transmission des données. On distingue à cet égard entre parité paire et parité impaire. La parité paire signifie que le mot de données transmis est complété par le bit de parité de telle façon que le nombre de bits soit toujours pair. Si le mot transmis comporte donc, par exemple, trois bits valant 1, le bit de parité vaudra également 1 pour que le nombre de bits 1 passe à 4 et soit bien un nombre pair. Si le mot transmis contenait par contre un nombre pair de bits 1, le bit de parité vaudrait

0. Inversement, si c'est la parité impaire qui est convenue, le bit de parité sera fixé de telle façon que le nombre de bits 1 soit toujours impair.

Bits de stop

Viennent en dernier lieu ce qu'on appelle les bits de stop, qui signalent la fin de la transmission des données correspondant à un caractère. Le protocole de transmission des données peut prévoir 1, 1,5 ou 2 bits de stop. Les utilisateurs sont généralement surpris qu'il soit possible de travailler avec 1,5 bits de stop. Comment un bit peut-il donc être divisé ? Il s'agit en fait d'un paradoxe en apparence seulement et qui s'explique très bien si on étudie le protocole de transmission des données.

Vitesse de transmission

Les anciens standards indiquaient que la transmission s'effectuait à 300 bauds, c'est-à-dire de 300 bits par seconde, avec un bit de stop. Comme les bits de stop valent toujours 1, cela voudrait dire que la ligne serait sur high pendant un trois-centième de seconde. Mais si l'on veut transmettre non pas 1 mais 1,5 bits de stop, la ligne devra rester high pendant non pas un trois-centième de seconde mais pendant 1,5 fois un trois-centième de seconde (autrement dit pendant un deux-centième de seconde). Le mystère est donc déjà résolu.

Notez qu'il existe aussi des interfaces qui travaillent avec une logique négative. Dans ce cas, tous les états 0 et 1 de la description précédente, qui correspond à une logique positive, doivent être intervertis. Cela ne change toutefois rien au principe de base de la transmission série.

Protocoles de transmission

La transmission des données ne peut cependant fonctionner qu'à condition que les différents paramètres variables de ce protocole soient connus aussi bien de l'émetteur que du récepteur. Au premier rang de ces paramètres à définir figure naturellement la vitesse de transmission, exprimée en bauds, c'est-à-dire en bits par seconde. Lorsqu'on utilise une ligne téléphonique ordinaire, on emploie habituellement une vitesse de transmission de 300 ou 1200 bauds, suivant que l'on travaille avec un coupleur acoustique ou bien avec un modem. Sur une ligne privée ou pour un transfert de données direct à l'aide d'un câble, des vitesses de transmission allant jusqu'à 9600 bauds sont possibles. A une telle vitesse, 80 octets par seconde ou 4800 par minute peuvent être transmis.

Le nombre de bits de données transmis chaque fois dépend des données à transmettre. Pour transmettre des données en véritable standard ASCII, 7 bits de données suffisent

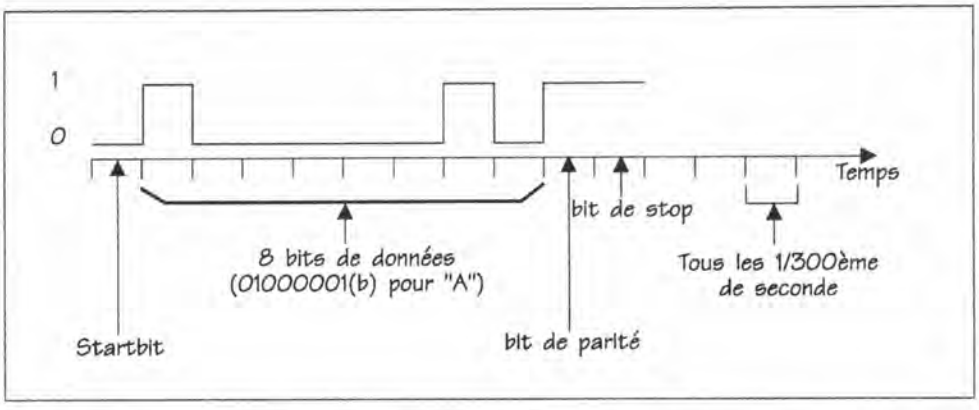
puisque le jeu de caractères ASCII ne comporte que 128 caractères. Mais si l'on veut exploiter pleinement le jeu plus complet des 256 caractères du PC, il faut transmettre 8 bits à la fois.

Il convient également de définir si un contrôle de parité doit être effectué et, si oui, s'il doit s'agir de la parité paire ou de la parité impaire. Il est généralement préférable de travailler avec un contrôle de parité car les lignes téléphoniques ne permettent pas toujours une transmission correcte des données. Peu importe, par contre, quel type de contrôle de parité est appliqué, les deux types garantissant la même sécurité de transmission.

Il faut enfin définir le nombre de bits de stop. Si on utilise un seul bit de stop, le caractère suivant peut être transmis plus rapidement qu'avec 2 bits de stop mais la transmission est moins sûre.

Exemple de protocole

La figure suivante vous montre comment se présenterait, par exemple, la transmission du caractère "A" avec un protocole prévoyant 8 bits de données, un contrôle de parité impaire et un bit de stop. Cet exemple repose sur une logique positive de l'interface et sur une vitesse de transmission de 300 bauds. Comme le code ASCII de la lettre A est 65 (01000001b) et qu'il contient par conséquent 2 bits 1, le contenu du bit de parité sera 1 dans ce cas, de façon à ce que le nombre de bits 1 soit impair.



Transmission du caractère "A" avec 8 bits de données, contrôle de parité impaire et 300 bauds

UART

Le cerveau d'une carte RS232 est constitué par un processeur appelé UART (Universal Asynchronous Receiver Transmitter = émetteur-récepteur asynchrone universel). Pour

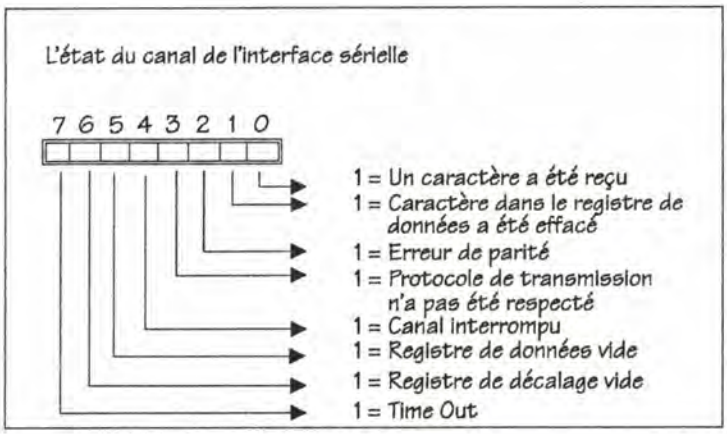
pouvoir réagir comme il convient aux messages d'erreur transmis par les différentes fonctions du BIOS, il est utile de bien comprendre la structure et le mode de fonctionnement de ce processeur.

Registres de transfert

Lorsqu'un caractère doit être transmis à travers le canal de données, il est tout d'abord transféré dans un registre appelé "transmission holding register" (Registre d'attente de l'émetteur). Il y reste ensuite jusqu'à ce que le caractère transmis précédemment ait été complètement traité. Il est ensuite transféré dans le registre appelé "transmission shift register" (Registre de décalage de l'émetteur), à partir duquel il est transféré bit par bit par l'UART à travers le canal de données. Suivant la convention appliquée, il insère le cas échéant les bits de parité ou les bits de stop voulus dans le flux de données. Lorsque les fonctions du BIOS renvoient l'état du canal de données dans le registre AH, les bits 5 et 6 indiquent si ces deux registres sont vides.

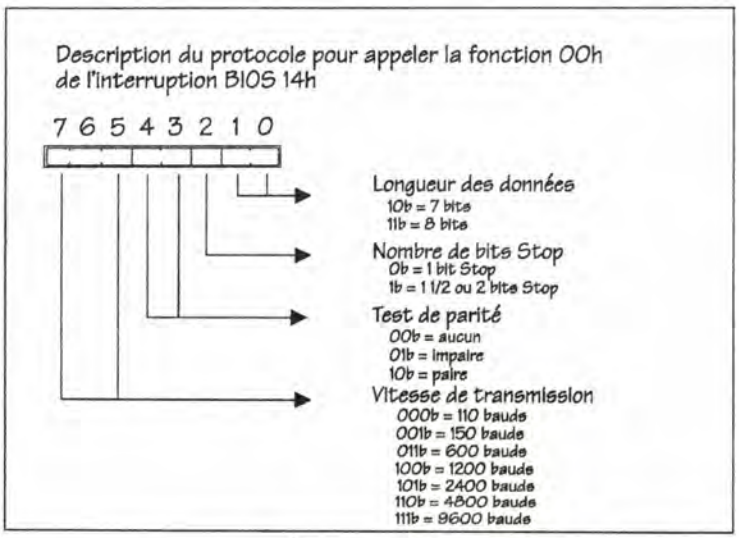
Registres de réception

Lorsqu'il s'agit de recevoir des données, celles-ci sont tout d'abord chargées dans le "receiver shift register" (Registre de décalage du récepteur), d'où elles sont ensuite transférées dans le "receiver data register" (Registre de données du récepteur). Les bits de parité et les bits de stop sont éliminés par l'UART à cette occasion. Si un caractère reçu précédemment figurait encore dans le registre de données, et s'il vient donc d'être effacé par l'opération, le bit 1 de l'état du canal est fixé sur 1. Le bit 0 indique qu'un caractère a été reçu. Si l'UART détecte une erreur de transmission (parce qu'il constate une erreur de parité) lors du traitement du caractère reçu, il fixe le bit 2 de l'état du canal. Si le protocole convenu (Nombre de bits de parité et de bits de stop) n'a pas été respecté, c'est le bit 3 qui est fixé. Le bit 4 est fixé par l'UART chaque fois qu'un canal de données est utilisé plus longtemps que nécessaire pour la transmission d'un caractère. Le bit 7 signale une erreur "time out", c'est-à-dire un dépassement du temps imparti. Cette erreur apparaît parfois lorsqu'on veut transférer des données à travers une ligne téléphonique et que la communication entre la carte RS232 et le modem ne peut être établie correctement.



Fonction 0 : Réglage du protocole

Avant que des données puissent être reçues ou envoyées, encore faut-il toutefois communiquer à l'UART le nombre de bits de stop, etc... On appelle à cet effet la fonction 00h de l'interruption 14h avec la valeur 0 dans le registre AH et le protocole dans le registre AL. Les différents bits du registre AL spécifient les différents paramètres :



Après l'initialisation, cette fonction renvoie l'état du canal dans le registre AH.

Fonction 1 : Envoi de caractère

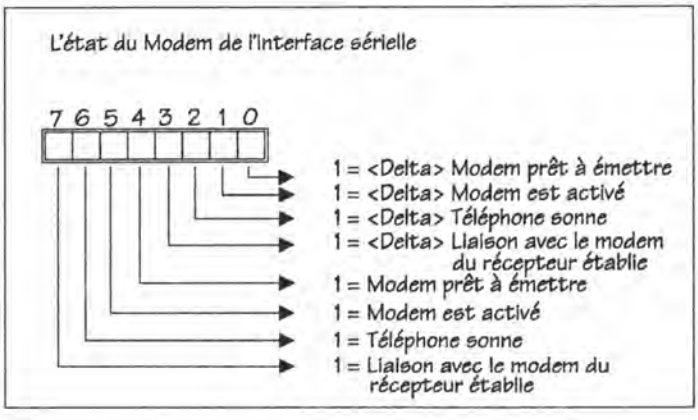
La fonction 01h est utilisée pour l'envoi de caractères. Lorsqu'elle est appelée, le registre AH doit contenir 01h et le registre AL le caractère à transmettre. Une fois le caractère transmis, le bit 7 du registre AH est fixé sur 0. Un 1 dans ce bit signalerait que le caractère n'a pu être transmis. Les autres bits correspondent à l'état du canal.

Fonction 2 : Réception de caractère

C'est la fonction 02h qui sert à la réception de caractères. Si un caractère a été reçu, le registre AL contiendra le caractère reçu après appel de cette fonction. Si AH contient la valeur 0, c'est qu'aucune erreur n'est à signaler, sinon il contient la valeur de l'état du canal.

Fonction 3 : Etat de la ligne/modem

La fonction 03h permet de connaître non seulement l'état du canal mais aussi l'état du modem. Elle fournit l'état du canal dans le registre AH et l'état du modem dans le registre AL :



Les bits 4 à 7 sont une réplique des bits 0 à 3, qui sont cependant chaque fois précédés du mot delta. Ce nom de lettre grecque est employé comme symbole d'une modification. Il signifie en l'occurrence que les bits 0 à 3 indiquent chaque fois si le contenu des bits 4 à 7 s'est modifié depuis la dernière lecture de l'état du modem. S'il en est ainsi, le bit delta correspondant contiendra 1. Si le bit 2 contient donc, par exemple, la valeur 1, cela signifie que le contenu du bit 6 s'est modifié depuis la dernière lecture. Concrètement, cela signifie que le téléphone vient juste de se mettre à sonner ou bien qu'il vient au contraire juste de s'arrêter de sonner, suivant l'état du bit 6.

9. Programmation de la souris

Si elles faisaient encore, il y a quelques années, figures d'animaux exotiques dans l'univers du PC, les souris ont maintenant trouvé leur place à côté de presque tous les claviers PC. Elles doivent ce succès à la diffusion toujours croissante des standards vidéo puissants tels que EGA et VGA, qui, à leur tour, permettent la réalisation d'environnements utilisateur graphique, qui doivent une grande partie de leur attrait à l'utilisation de la souris.

L'emploi de la souris ne se limite d'ailleurs pas aux applications fonctionnant en mode graphique. Depuis qu'est apparu le standard SAA et l'environnement utilisateur DOS (4.0, 5.0), les utilisateurs attendent aussi des applications orientées texte qu'elles acceptent la souris, parallèlement au clavier, comme un périphérique de saisie à part entière. Des programmes tels que Microsoft Works, Ventura Publisher, démontrent que la souris permet d'organiser le mode d'emploi d'un programme de façon beaucoup plus simple et commode.

L'interface qui a été définie par Microsoft pour ses diverses souris a été acceptée comme standard par les autres fabricants. Cette interface est généralement installée à l'aide d'un driver de périphérique, qui est chargé lors du chargement du système, ou à travers un programme TSR, comme le programme MOUSE.COM des souris Microsoft.

9.1. Les fonctions souris

La technique servant à appeler les différentes fonctions est la même que pour l'interface de programmation du DOS et du BIOS : les différentes fonctions sont appelées à travers un interruption spéciale, l'interruption 33h, le numéro de code de la fonction appelée devant être spécifié dans le registre AX. Les autres registres du processeur sont également utilisés, avec des combinaisons différentes, pour transmettre des informations aux diverses fonctions.

Appel des fonctions souris

Même si 53 fonctions différentes peuvent être ainsi appelées, seules quelques-unes seront réellement utilisées dans une application. Avant d'en venir à la description de chaque fonction, un exposé rapide des principes sur lesquels repose l'interface souris nous aidera à mieux comprendre les différentes fonctions. Nous nous intéresserons ici essentiellement aux problèmes qui se posent dans une application orientée texte. Si vous souhaitez en effet développer une application orientée texte, vous auriez de toute façon plutôt intérêt à recourir à un environnement graphique, comme les API Windows, qui offre pour le travail avec la souris des fonctions beaucoup plus pratiques que celles que l'interface de programmation que nous allons décrire ici peut offrir.

La table suivante fournit la liste des fonctions supportées par les gestionnaires de souris jusqu'à la version 8.0 incluse. Ces fonctions sont plus que suffisantes pour gérer la souris dans une application orientée texte.

Fonction	Utilisation	Version
00H*	Réinitialise le driver	01
01H*	Affiche le curseur souris	01
02H*	Cache le curseur souris	01
03H*	Lit la position du curseur/l'état du bouton	01
04H*	Déplace le curseur souris	01
05H*	Détermine le nombre de fois le bouton est appuyé	01
06H*	Détermine le nombre de fois le bouton est relâché	01
07H*	Règle la zone horizontale de déplacement	01
08H*	Règle la zone verticale de déplacement	01
09H	Définit le curseur souris (mode graphique)	01
0AH*	Définit le curseur souris (mode texte)	01
0BH*	Détermine les distances de déplacement	01
0CH*	Définit le handler d'événement	01
0DH	Active l'émulation du crayon optique	01
0EH	Désactive l'émulation du crayon optique	01
0FH	Définit la vitesse du curseur	01
10H*	Zone d'exclusion	01
11H	Non documenté	01
12H	Non documenté	01
13H*	Définit la vitesse maximale du double-clic	01
14H	Echange les handlers d'événement	01
15H	Détermine la taille du buffer de l'état de la souris	01
16H*	Sauve l'état de la souris	01
17H*	Restaure l'étt de la souris	01
18H*	Installe un handler d'événement alterné	01
19H	Détermine l'adresse du handler d'événement alterné	01
1AH	Règle la sensibilité de la souris	01
1BH	Détermine la sensibilité de la souris	01

Fonction	Utilisation	Version
1CH	Définit le taux d'interruption hardware de la souris	01
1DH*	Définit la page d'affichage	01
1EH*	Détermine la page d'affichage	01
1FH	Désactive le driver souris	01
20H	Active le driver souris	01
21H	Réinitialise le driver souris	01
22H	Définit la langue pour les messages	01
23H	Lit le numéro de la langue	01
24H	Détermine le type de souris	01
25H	Lit les informations du driver	06
26H	Lit les coordonnées virtuelles maxi	06
27H*	Lit les masques et les compteurs mickey	7A
28H	Définit le mode vidéo	07
29H	Compte les modes vidéo	07
2AH	Lit le hotspot du curseur	7B
2BH	Définit les courbes d'accélération	07
2CH	Lit les courbes d'accélération	07
2DH	Définit/lit les courbes d'accélération	07
2EH	Non documenté	01
2FH	Réinitialise le hardware souris	7B
30H	Définit/lit les informations de la ballpoint	7C
31H	Lit les coordonnées virtuelles mini/maxi	7D
32H	Lit les fonctions actives avancées	7D
33H	Lit les réglages des switch	7D
34H	Lit la localisation de MOUSE.INI	08

Légende :
*** = fonction couramment utilisée (décrite dans ce chapitre)**
01 = Version >= 1.0
06 = Version >= 6.26
07 = Version >= 7.0
7A = Version >= 7.01
7B = Version >= 7.02
7C = Version >= 7.04
7D = Version >= 7.05
08 = Version >= 8.0

Les boutons de la souris

Contrairement à un clavier avec ses nombreuses touches et le nombre de codes différents qui en résulte, une souris PC ne dispose généralement que de deux boutons, parfois trois, pour permettre à l'utilisateur de transmettre des instructions à un programme.

Une autre information, à savoir la position de la souris ou plutôt de curseur de la souris sur l'écran, joue cependant également un rôle décisif. C'est cette information qui fait de la souris une sorte de baguette permettant de désigner n'importe quel endroit de l'écran. Une information telle que le fait que le bouton gauche de la souris a été appuyé doit donc toujours être interprétée en fonction de la position actuelle de la souris. La position de la souris permet de désigner l'objet (de l'écran) sur lequel figure le curseur de la souris, et qui a ainsi été sélectionné par l'utilisateur.

L'écran souris virtuel

A l'intérieur du driver de la souris, les positions de la souris sont toujours interprétées par référence à un écran graphique virtuel, dont la résolution dépend du mode vidéo sélectionné et de la carte vidéo utilisée. Cet écran graphique virtuel est utilisée y compris dans les différents modes de texte pour déterminer la position de la souris, et il sert de base à la communication avec l'interface souris, de sorte que ces coordonnées graphiques doivent toujours être converties en coordonnées de colonne et de ligne du curseur de la souris. Chaque colonne ou chaque ligne correspondant à 8 points, les coordonnées graphiques doivent donc être divisées par 8, ou bien être décalées de trois chiffres binaires sur la droite, ce qui revient exactement au même arithmétiquement, mais peut être réalisé par le processeur beaucoup plus rapidement qu'une division.

Le curseur de la souris

La souris est représentée sur l'écran par un curseur, qui suit les déplacements de la souris sur le bureau. En mode texte, ce curseur peut être représenté sous forme d'un curseur électronique clignotant, ou bien sous forme d'un curseur logiciel dont la forme peut être définie par une application. Dans le premier cas, l'application peut seulement fixer les lignes de départ et de fin de ce curseur sur le caractère recouvert, en tenant compte de la taille de la matrice de caractère actuelle dans le mode vidéo activé. Le curseur logiciel offre des possibilités de configuration beaucoup plus larges, puisque son apparence peut être définie par deux valeurs 16 bits appelées masque écran (Screen Mask) et masque curseur (Cursor Mask).

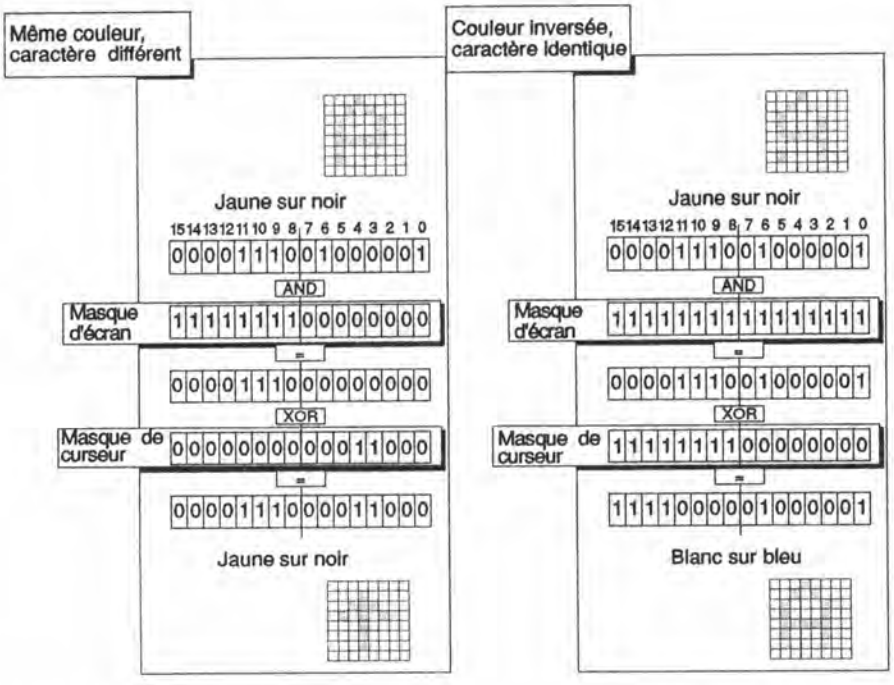
L'apparence du curseur logiciel est redéfinie par le driver de la souris sur chaque position de l'écran, en combinant le masques curseur et écran avec les 2 octets définissant, à l'intérieur de la RAM vidéo, le code caractère et la couleur d'un caractère. Cette combinaison s'effectue en deux étapes, le code caractère et l'octet d'attribut étant tout

d'abord combinés avec le masque écran par un ET binaire. Le résultat de cette combinaison est alors, à son tour, combiné avec le masque curseur par un OU exclusif, et le résultat est ensuite envoyé sur l'écran.

Bien que cette méthode n'ait rien d'évident au premier abord, ce mode de combinaison offre toute une série de possibilités pour définir l'apparence du curseur de la souris, parmi lesquelles les quatre suivantes sont les plus fréquemment employées :

- ✓ Le curseur de la souris est toujours affiché sous forme d'un caractère déterminé, avec une couleur constante
- ✓ Le curseur de la souris est toujours affiché sous forme d'un caractère déterminé, mais sa couleur dépend de la couleur du caractère recouvert (qui sera par exemple inversée)
- ✓ Le caractère sous le curseur de la souris ne change pas mais reçoit une couleur déterminée
- ✓ Le caractère sous le curseur de la souris ne change pas mais reçoit une couleur obtenue par combinaison avec la couleur du caractère actuel

La figure suivante vous montre des exemples de ces possibilités, avec les masques écran et curseur correspondants.



Le curseur de la souris résultant de la combinaison du caractère actuel avec les masques curseur et écran.

L'interface souris rend hommage à la souris certainement la plus célèbre : Mickey. Le mickey est en effet le nom qui a été donné à l'unité mesurant un parcours de 1/200 de pouce (1,27 mm). L'électronique de la souris a été conçue de façon à ce que toutes les distances correspondent à un multiple entier de cette longueur. C'est pourquoi nous aurons de nombreuses occasions d'évoquer cette unité de mesure lors de la description des différentes fonctions.

Fonction 00H : Réinitialise le driver souris

Avant de travailler avec les fonctions de l'interface souris, un programme doit de préférence appeler tout d'abord la fonction 00h, pour réinitialiser le driver de la souris. Au vu du contenu du registre AX après appel de cette fonction, le programme pourra, en outre, déterminer si une souris et un driver de souris approprié sont bien effectivement installés. Si le registre AX contient encore la valeur 0000h après appel de cette fonction, cela signifie qu'aucun driver de souris n'est en tout cas installé (même si une souris peut être présente). Le programme pourra donc se dispenser de tenir compte de la souris au cours de son travail.

Si par contre un driver de souris, et donc aussi une souris, est installé, la fonction 00h renvoie la valeur FFFFh dans le registre AX. Comme nous l'avons déjà indiqué plus haut, les souris PC disposent généralement de deux boutons, bien que certains fabricants dotent leurs souris de trois boutons. Sous peine de devoir fournir un travail de programmation beaucoup plus important, il est donc préférable de renoncer à une définition individuelle et de n'utiliser que deux boutons de la souris.

La "réinitialisation" du driver de la souris consiste, pour la fonction 00h, à réinitialiser de nombreux paramètres de la souris que l'interface souris permet de fixer, en rétablissant les valeurs par défaut de ces paramètres.

Fonction 01H : Affiche le curseur de la souris

La fonction 01H affiche le curseur à l'écran. Elle n'attend pas d'autre paramètre que le numéro de fonction dans le registre AX. Comme le driver de la souris suit les déplacements de la souris même lorsque le curseur de la souris est désactivé, il se peut que le curseur de la souris n'apparaisse d'ailleurs déjà plus au centre de l'écran où à l'endroit où il se trouvait la dernière fois qu'il a été caché.

Fonction 02H : Cache le curseur de la souris

Le curseur peut à nouveau être caché à l'aide de la fonction 02h, qui n'attend pas non plus d'autre paramètre que le numéro de fonction. Il existe une relation très particulière entre les fonctions 01h et 02h, car les appels des deux fonctions doivent être équilibrés

pour produire un effet. Si vous appelez par exemple la fonction 02h par deux fois consécutives, la fonction 01h devra également être appelée deux fois pour que le curseur de la souris réapparaisse sur l'écran. Il en va naturellement de même dans le sens contraire.

Vous n'aurez d'ailleurs, en principe, pas à employer ces deux fonctions très souvent, car il suffit de rendre le curseur de la souris visible après avoir appelé la fonction 00h, puis de le cacher à nouveau à la fin du programme. Vous ne serez contraint d'avoir fréquemment recours à ces fonctions que si votre programme écrit directement dans la RAM vidéo, pour contourner les routines de sortie assez lentes du DOS et du BIOS. Dans ce cas, en effet, vous devrez veiller à ce que le curseur de la souris ne soit pas recouvert par vos sorties, ce qui aurait deux conséquences fâcheuses :

① le curseur de la souris disparaîtrait de l'écran parce qu'il aurait été effacé par un autre caractère

② le driver de la souris enverrait sur l'écran un caractère erroné lorsque le curseur de la souris serait déplacé. Cela est dû au mode de travail du driver de la souris lorsque le curseur de la souris est déplacé sur l'écran. Avant de placer le curseur de la souris dans un emplacement donné de l'écran, le driver de la souris enregistre en effet tout d'abord le caractère qui figurait jusqu'ici dans cette position, pour pouvoir le replacer sur l'écran dès que le curseur de la souris sera déplacé vers une autre position de l'écran. Or, en cas d'écriture directe dans la RAM vidéo, le driver de la souris ne peut détecter qu'un nouveau caractère a été sorti dans l'emplacement du curseur de la souris. Lorsque le curseur de la souris sera déplacé, il ramènera donc naturellement l'ancien caractère, c'est-à-dire un caractère qui ne convient plus, sur l'écran.

Cette source d'erreur peut cependant être évitée en cachant le curseur de la souris avant la sortie de caractères, ce qui oblige le driver de la souris à ramener l'ancien caractère sur l'écran, puis en incrustant le curseur de la souris à nouveau, après la sortie de caractères, pour faire enregistrer le nouveau caractère. Il est cependant déconseillé d'effectuer ces opérations lors de la sortie de chaque caractère, car elles prennent beaucoup de temps et contrebalanceraient le temps gagné par une écriture directe dans la RAM vidéo. Il est donc recommandé de cacher le curseur de la souris avant une opération de sortie d'envergure, par exemple lorsqu'il s'agit de mettre en place une fenêtre de l'écran, et de ne l'incruster sur l'écran à nouveau qu'une fois cette opération achevée.

Les fonctions DOS et BIOS de sortie de caractères envoyant également, au bout du compte, leurs sorties directement dans la RAM vidéo, le même problème devrait se poser. En réalité, le programmeur n'a pas à se préoccuper du curseur de la souris lorsqu'il travaille avec ces fonctions, car, lors de son installation, le driver de la souris détourne vers une routine personnelle le vecteur d'interruption 10h, à travers lequel s'effectuent les sorties sur écran du BIOS et du DOS. A travers ces fonctions, le driver de la souris peut ainsi cacher le curseur de la souris avant toute sortie de caractères, puis l'incruster à nouveau après la sortie, chaque fois que cela est nécessaire.

Fonction 04H : Déplace le curseur de la souris

La fonction 04h permet d'amener le curseur de la souris sur un emplacement donné de l'écran, sans que la souris ait été déplacée. Elle attend le numéro de fonction dans le registre AX, la nouvelle ordonnée horizontale (la colonne) dans le registre CX et l'ordonnée verticale (la ligne) dans le registre DX. Les coordonnées de texte doivent donc être multipliées par huit (ou bien être décalées de trois chiffres binaires sur la gauche) avant d'être transmises à la fonction 04h. Les coordonnées fournies doivent se situer à l'intérieur de la zone de l'écran définie comme zone de déplacement de la souris

Après appel de la fonction 00h (réinitialisation du driver de la souris), cette zone comprend tout d'abord la totalité de l'écran, mais elle peut être limitée avec les fonctions 07h et 08h.

Fonctions 07H & 08H : Définissent la zone de déplacement

La fonction 07h définit la zone de déplacement horizontale, l'ordonnée X minimale devant lui être spécifiée dans le registre CX, l'ordonnée X maximale dans le registre DX. Il en va de même pour la fonction 08h, qui définit la zone de déplacement verticale, et qui attend dans les registres CX et DX les ordonnées Y minimale et maximale respectivement. N'oubliez pas de convertir les coordonnées de texte en coordonnées de l'écran virtuel de la souris, en multipliant les coordonnées par huit.

Après appel de ces fonctions, le driver de la souris ramène automatiquement le curseur de la souris dans la zone spécifiée, si toutefois il figurait en dehors de ces limites. L'utilisateur n'a alors plus aucune possibilité de déplacer le curseur de la souris au-delà de ces limites, puisque tout déplacement est stoppé aux limites de cette zone.

Fonction 10H : Zone d'exclusion

A la zone de déplacement de la souris s'oppose la zone dite d'exclusion, qui fonctionne comme une sorte de trou noir et engloutit le curseur de la souris (c'est-à-dire le rend invisible) dès qu'il pénètre dans cette zone. Le curseur de la souris redevient toutefois visible dès qu'il quitte cette zone. A la suite d'une réinitialisation du driver de la souris, aucune zone d'exclusion n'est encore définie, mais la fonction 10h permet d'en définir une à tout moment. Le driver de la souris ne peut cependant gérer qu'une seule zone d'exclusion à la fois. Les coordonnées de la zone d'exclusion doivent être transmises à la fonction 10h dans les registres CX et DX (coin supérieur gauche) ainsi que SI et DI (coin inférieur droit). CX et SI reçoivent l'ordonnée X, DX et SI l'ordonnée Y.

Comme la fonction 02h, la zone d'exclusion joue un rôle important en cas d'accès direct à la RAM vidéo. Alors que la fonction 02h élimine le curseur de la souris de l'écran dans tous les cas, la fonction 10h ne le fait que s'il figure déjà dans la zone d'exclusion ou

bien s'il y est amené. La fonction 10h est donc particulièrement utile lorsqu'il s'agit de reconstruire une zone de l'écran assez vaste (comme une fenêtre par exemple), car elle permet de laisser le curseur de la souris sur l'écran tant qu'il ne figure pas à l'intérieur de cette zone.

La zone d'exclusion peut être annulée en appelant la fonction 01h ou la fonction 00h. La fonction 01h fait automatiquement réapparaître le curseur s'il se trouvait à l'intérieur de la zone d'exclusion.

Fonction 1DH : Définit la page d'affichage

La fonction 1Dh permet de fixer dans quelle page écran le curseur de la souris doit être sorti (le numéro de page écran doit être placé dans le registre BX). Cela n'est cependant nécessaire que lorsque le programme amène au premier plan, par programmation directe de l'électronique d'une carte vidéo, une autre page écran que celle qui était jusqu'ici affichée. Lorsqu'une page écran est activée à l'aide de la fonction appropriée de l'interruption BIOS 10h, il est superflu d'appeler cette fonction car le driver de la souris en tient automatiquement compte.

Fonction 0FH : Définit la vitesse du curseur

Deux paramètres déterminent la "vitesse" à laquelle le curseur de la souris se déplace sur l'écran. Ils décrivent le rapport entre la longueur d'un déplacement physique de la souris et le nombre de points parcourus parallèlement dans l'écran virtuel de la souris. La fonction 0Fh permet de fixer ces paramètres séparément pour les déplacements horizontal et vertical. Les deux paramètres transmis dans les registres CX (déplacement horizontal) et DX (déplacement vertical) indiquent le nombre de mickeys équivalant à 8 points de l'écran virtuel de la souris, c'est-à-dire à une ligne ou à une colonne de l'écran de texte.

Les valeurs par défaut fixées par la fonction 00h sont de 8 mickeys horizontalement et de 16 mickeys verticalement. Dans l'écran de texte, le curseur de la souris se déplacera donc d'une colonne lorsque la souris sera déplacée de 8 mickeys (environ 10 mm) horizontalement. Pour sauter à la ligne suivante, il faudra cependant que la souris parcoure 16 mickeys verticalement, c'est-à-dire 20 mm.

Il n'est pas conseillé de modifier ces réglages, car ils tiennent compte des différences de résolution de l'écran entre la largeur et la longueur et donnent de bons résultats. Si toutefois vous souhaitez une souris "plus rapide" ou "plus lente", cette fonction vous permet de modifier la vitesse entièrement à votre gré.

Fonction 0AH : Définit la forme de la souris

La fonction 0AH détermine l'apparence du curseur en mode texte. Le masque du curseur et le masque écran peuvent également être définis à l'aide d'une fonction de l'interface souris, la fonction 0Ah. Le programme appelant la fonction 0Ah indique au driver de la souris, dans le registre BX, si le driver de la souris doit utiliser le curseur électronique ou bien un curseur logiciel.

Curseur défini par logiciel

Si le registre BX contient la valeur 0, le driver de la souris sélectionnera le curseur logiciel. Dans ce cas, il attend dans le registre CX le masque écran et dans le registre DX le masque curseur qui définiront désormais l'apparence du curseur de la souris.

Curseur défini par hardware

Le driver de la souris sélectionne le curseur électronique si la valeur 1 lui a été transmise dans le registre BX, lors de l'appel de la fonction 0Ah. Il interprète alors le contenu du registre CX comme ligne de départ du curseur électronique, le contenu du registre DX comme sa ligne finale.

Mode vidéo et taille du curseur

Notez bien à ce propos que les valeurs admises pour les lignes de départ et de fin dépendent chaque fois du mode vidéo sélectionné. En mode d'écran monochrome, les valeurs entre 0 et 13 sont autorisées, en mode couleur seules les valeurs de 0 à 7. La matrice de caractère des cartes EGA et VGA comprend en effet jusqu'à 14 ou 16 lignes de balayage, mais il ne faut pas en tenir compte car les valeurs entre 0 et 7 spécifiées sont automatiquement converties par le BIOS EGA ou VGA en fonction de la taille de la matrice de caractère actuelle.

Les fonctions présentées jusqu'ici servaient à fixer les différents paramètres affectant le fonctionnement du driver de la souris. Le driver de la souris soutient naturellement toute une série de fonctions servant à tester la position de la souris et l'état des boutons de la souris. Ces diverses fonctions peuvent être classées en deux catégories correspondant à des approches différentes du test de périphériques externes tels que la souris, le clavier, l'imprimante ou le lecteur de disquette. On oppose en effet les méthodes dites Polling ou Interrupt.

Méthode Polling

La méthode polling signifie qu'un périphérique est testé en permanence dans le cadre d'une boucle, qui ne se termine qu'une fois que l'événement attendu s'est produit. L'exécution de cette boucle absorbant totalement l'unité centrale, celle-ci n'a généralement plus de temps pour accomplir d'autres tâches.

Méthode Interrupt

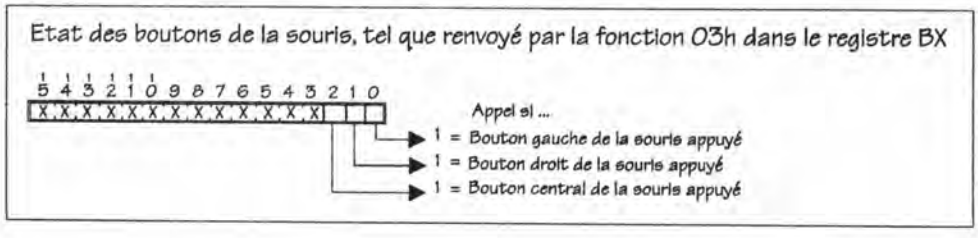
La méthode d'interruption (Interrupt) permet de remédier à cet inconvénient car l'unité centrale peut vaquer à d'autres tâches jusqu'à ce que l'événement ("event" en anglais) se soit produit. Ce n'est qu'alors que le driver de la souris appellera une routine, qui aura été déclarée comme routine d'interruption pour réagir à l'événement attendu, puis exécutera les autres opérations nécessaires pour réagir à l'événement.

Fonction 03H : Lire la position du curseur/l'état du bouton

Quelle que soit la méthode que vous préférez, les deux sont soutenues par le driver de la souris. L'interface souris offre quatre fonctions relevant de la méthode du Polling, mais il s'agit en réalité de variantes et une seule de ces fonctions, la fonction 03h, suffit généralement tout à fait à la tâche.

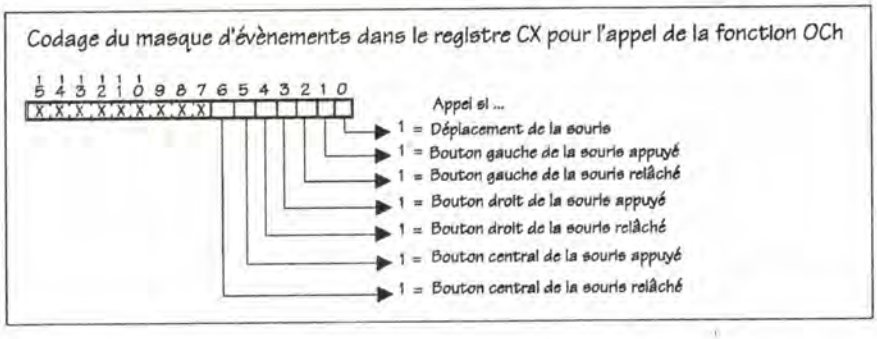
Cette fonction est l'une des rares de l'interface souris à renvoyer des informations au programme d'appel. Ces informations indiquent non seulement la position actuelle du curseur de la souris, mais aussi l'état des boutons de la souris. La position horizontale de la souris est fournie dans le registre CX, la position verticale dans le registre DX. Notez bien que ces coordonnées se réfèrent également à l'écran virtuel de la souris et qu'elles doivent donc être divisées par 8 pour être converties en coordonnées de l'écran de texte.

Comme le montre la figure suivante, l'état des boutons de la souris est renvoyé dans le registre BX, seuls les trois bits inférieurs étant cependant significatifs. Ils représentent chacun l'état de l'un des deux ou trois boutons de la souris, et contiennent 1 si le bouton correspondant est appuyé au moment de l'appel de la fonction.



Fonction 0CH : Définit le handler d'événement

La fonction 0Ch permet d'installer une routine d'interruption, qu'on appelle aussi gestionnaire d'événement (Event Handler). Elle attend, outre le numéro de fonction, les adresses de segment et d'offset du gestionnaire d'événement dans la paire de registres ES:DX ainsi que ce qu'on appelle un masque d'événements dans le registre CX. Les différents bits de ce flag décident dans quelles situations le gestionnaire d'événement doit être appelé. Examinez à ce propos la figure suivante :



Après exécution de cette fonction, le gestionnaire d'événements sera appelé par le driver de la souris dès qu'un au moins des événements spécifiés sera intervenu. Contrairement à ce qui se passe avec une routine d'interruption, l'appel ne s'effectue toutefois pas à travers l'instruction INT du langage machine mais avec une instruction FAR CALL.

Cette distinction est très importante pour la configuration du gestionnaire d'événements, car elle implique qu'il doit se terminer par une instruction langage machine FAR RET, et non par IRET. Comme une routine d'interruption, il ne doit renvoyer avec une valeur différente aucun des divers registres du processeur. Immédiatement après avoir été appelé, le gestionnaire d'événements doit donc sauvegarder sur la pile les registres risquant d'être modifiés, de façon à pouvoir, en fin d'exécution, retirer les valeurs sauvegardées de la pile, pour les charger à nouveau dans les registres appropriés.

Le driver de la souris transmet au gestionnaire d'événements différentes informations dans les divers registres du processeur, par exemple le type d'événement ayant provoqué l'appel du gestionnaire. Cette information figure par exemple dans le registre AX, les différents bits revêtant la même signification que dans le masque d'événements pour l'appel de la fonction 0Ch (voyez la figure présentée plus haut). Certains bits peuvent cependant également être fixés bien qu'ils ne représentent pas l'intervention d'un quelconque événement.

Lorsque le gestionnaire d'événements est par exemple censé être appelé uniquement lorsque le bouton gauche de la souris est appuyé (bit 1), les bits 0 et 4 peuvent alors aussi être mis si la souris a été déplacée en même temps que le bouton droit était relâché.

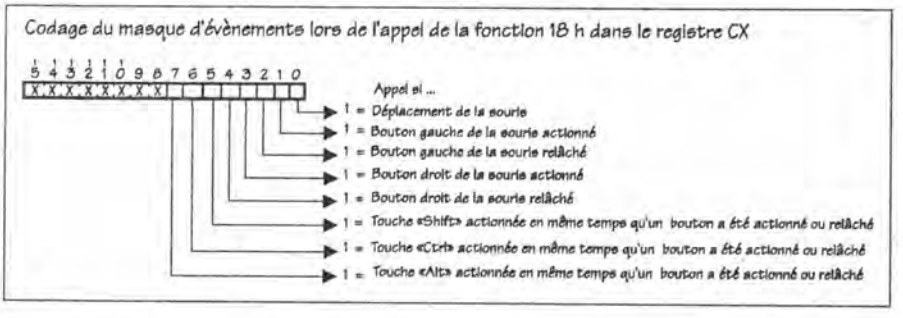
Lorsque le gestionnaire d'événements est appelé, il peut lire l'état des boutons de la souris dans le registre CX. La valeur dans ce registre est codée exactement comme pour l'appel de la fonction 03h, les bits 0 à 2 représentant donc les différents boutons de la souris. La position actuelle du curseur de la souris est indiquée dans les registres CX (horizontale) et DX (verticale). Cette position doit cependant être convertie avant de pouvoir être interprétée dans le cadre du système de coordonnées de l'écran texte.

Lorsqu'on développe un gestionnaire d'événements, il convient également de tenir compte du fait que le registre DS, lors de l'appel du gestionnaire, ne désigne pas le segment de données du programme interrompu mais celui du driver de la souris. Pour accéder à son segment de données propre, le gestionnaire d'événements doit donc d'abord en charger l'adresse dans le registre DS.

Fonction 18H : Installe un handler d'événement alterné

La fonction 18h permet d'installer un gestionnaire d'événements capable de réagir non seulement aux événements concernant la souris mais aussi (dans une mesure limitée) aux événements du clavier. La combinaison avec un événement du clavier permet, lorsqu'on le souhaite, que le gestionnaire d'événements ne soit appelé que lorsqu'une des trois touches de contrôle »Alt«, »Ctrl« ou »Shift« est appuyée en même temps qu'un bouton de la souris est appuyé ou relâché.

La définition des registres est tout à fait identique à celle lors de l'appel de la fonction 0Ch, mais le masque d'événements dans le registre CX a été complété de ces trois événements.



Lorsqu'on appelle un gestionnaire d'événements "alternatif", peu de choses changent également par rapport aux gestionnaires d'événements installés avec la fonction 0Ch. Seul le contenu du registre AX doit être interprété un peu différemment car sa structure correspond à celle du masque d'événements ci-dessus.

Une autre différence par rapport aux gestionnaires d'événements "normaux" est que la fonction 18h permet d'installer jusqu'à trois gestionnaires d'événements "alternatifs". Alors que tout appel de la fonction 0Ch a pour effet de remplacer le gestionnaire

d'événements déjà installé par celui spécifié, trois gestionnaires d'événements différents peuvent être installés en appelant à trois reprises la fonction 18h, à condition toutefois que chaque gestionnaire d'événements soit doté d'un masque d'événements différent. Lorsqu'est spécifié à la fonction 18h un masque d'événements déjà associé à un gestionnaire installé, ce gestionnaire est remplacé par le nouveau gestionnaire d'événements.

9.2. Programmes de démonstration

Nous vous proposons dans ce chapitre, à titre d'exemples du travail avec les fonctions de l'interface souris, un programme C et un programme Turbo Pascal se chargeant de la partie certainement la plus complexe du test de la souris, la réalisation et l'installation d'un gestionnaire d'événements. Les deux programmes comportent en outre quelques fonctions ou procédures servant à appeler les différentes fonctions de la souris. Ces routines ne posent d'ailleurs pas de problèmes trop compliqués pour les programmeurs puisqu'il s'agit essentiellement de charger les valeurs voulues dans les registres du processeur puis d'appeler l'interruption 33h. Ce type de séquences d'instructions ayant été décrites de façon suffisamment détaillée dans les chapitres précédents, nous nous dispenserons de décrire ici le fonctionnement de ces routines. Concentrons-nous donc sur la partie la plus complexe des programmes : le gestionnaire d'événements.

L'installation d'un gestionnaire d'événements dans un programme en langage évolué n'est pas très simple car elle suppose la réalisation d'un certain nombre de conditions qui échappent normalement au contrôle du programmeur dans un langage évolué :

- ✓ le gestionnaire d'événements doit être une procédure FAR, c'est-à-dire se terminer par une instruction FAR RET,
- ✓ il doit sauvegarder les différents registres du processeur lorsqu'il est appelé, et les restaurer avant de se terminer et
- ✓ il doit charger l'adresse de segment du segment de données en langage évolué dans le registre DS pour pouvoir accéder aux variables globales du programme.

Grâce à l'extension qu'ont connue les versions les plus récentes de Turbo Pascal, QuickC, Turbo C et MSC, ces exigences pourraient être remplies directement en langage évolué, mais cela nécessiterait un peu de bricolage, de sorte que nous avons préféré opter pour une solution plus traditionnelle. Le véritable gestionnaire d'événements a donc été programmé en assembleur, le module objet correspondant étant ensuite intégré dans les programmes en langage évolué.

Dans les deux programmes, cette routine assembleur s'appelle AssHand. Elle se contente en fait, une fois appelée, de sauvegarder sur la pile les différents registres du processeur, puis d'appeler une fonction C ou une procédure Pascal appelée MouEventHandler. Elle transmet comme arguments à cette routine différentes informations qui

lui ont été communiquées par le driver de souris dans les différents registres du processeur. Il s'agit d'abord du Event Flag, qui décrit l'événement ayant provoqué l'appel du gestionnaire d'événements, ensuite de l'état des boutons de la souris, et enfin de la position actuelle du curseur de la souris. Elle ne retransmet cependant pas ces informations telles quelles à MouEventHandler. Pour gagner du temps, elle convertit tout d'abord les coordonnées qui lui ont été transmises du système de coordonnées de l'écran graphique virtuel vers le système de coordonnées de l'écran texte, avec ses 25 lignes et 80 colonnes (normalement).

Comme il est habituel dans les langages évolués, les paramètres sont transmis à travers la pile. Il convient de noter à ce propos, en ce qui concerne la version C de AssHand, que les arguments doivent être placés sur la pile dans l'ordre inverse de leurs déclarations (dans la fonction C), comme le veut l'usage en C. Après appel de la routine en langage évolué (avant quoi le registre DS doit avoir été défini), ces arguments doivent à nouveau être retirés de la pile en augmentant le pointeur de pile à raison de la place mémoire occupée par les arguments (8 octets). Cela ne vaut cependant que pour la version C de la routine. En Turbo Pascal, c'est la procédure Pascal appelée qui se charge de cette tâche.

Après appel de cette routine, la routine assembleur a déjà terminé son travail et il ne lui reste plus qu'à préparer le retour vers le driver de la souris. Elle retire pour cela de la pile les registres du processeur sauvegardés auparavant et rend alors le contrôle au programme d'appel, à l'aide d'une instruction FAR RET.

Alors que les différentes instructions de AssHand sont vite exécutées, l'exécution du gestionnaire en langage évolué peut prendre beaucoup de temps dans certains cas. Cela pose le problème de la récursion, car un événement lié à la souris peut à nouveau intervenir pendant l'exécution de ce gestionnaire. Le driver de la souris AssHand risque alors d'être à nouveau appelé avant que l'appel précédent ne se soit conclu. Pour éviter que cette situation ne se produise, avec les complications qui en résulteraient, AssHand gère dans son segment de code une variable appelée »actif«, à laquelle il attribue la valeur 1 lors de son exécution. Il teste cependant auparavant si cette variable ne vaut pas déjà 1, ce qui signifierait tout simplement que le dernier appel n'est pas encore terminé. Dans ce cas, l'exécution du gestionnaire s'interromprait aussitôt pour éviter une récursion.

Si cette méthode permet d'éviter les problèmes inhérents à la récursion, elle ne va pas sans certains inconvénients. L'appel du gestionnaire en langage évolué étant en effet interdit, celui-ci ignorera l'événement à cause duquel il avait été appelé par le driver de la souris. Malgré le "piège de la récursion", il est donc conseillé de programmer le gestionnaire d'événements en langage évolué de façon aussi efficace que possible, pour que son exécution prenne le moins de temps possible et qu'il ne soit pas nécessaire d'interdire des appels.

Il ne suffit cependant pas que AssHand soit présent pour qu'il soit appelé par le driver de la souris. Ce gestionnaire d'événements doit en effet être d'abord installé à l'aide de la fonction 0Ch de l'interface souris, ce dont se charge la procédure/fonction MouISe-

tEventHandler. Cette procédure/fonction est de son côté appelée par la procédure/fonction MouInit, qui initialise le module de la souris et devrait donc être la première procédure/fonction de ce module appelée par tout programme d'application. Elle attend comme arguments le nombre de lignes et de colonnes de l'écran, dont elle a besoin pour déterminer la taille d'un buffer interne qui joue un rôle central pour le fonctionnement des différentes procédures/fonctions du module.

A l'aide de ce buffer, l'écran peut être divisé en différentes zones de la souris, dont chacune sera dotée d'un code et de masques curseur et écran propres. Ces zones de souris jouent un rôle très important pour le travail avec la souris, car elles permettent de décrire différents objets de l'écran, comme un slider, un champ OK ou un point de menu, qui conduisent le programme à déclencher une action déterminée dès que le curseur de la souris est amené sur ces objets et qu'un bouton de la souris est appuyé.

Ces zones peuvent être enregistrées à l'aide de la procédure/fonction MouDefZone, à laquelle doivent être transmis un pointeur sur un vecteur ou sur un tableau ainsi que le nombre d'éléments de ce tableau. Ces éléments du type ZONE décrivent chacun une zone de l'écran ainsi que les masques curseur et écran qui doivent être employés pour le curseur de la souris dès qu'il pénètre dans la zone correspondante. Une zone peut se limiter à un seul caractère, mais peut aussi englober l'écran tout entier, car cela est entièrement de la responsabilité du programme d'appel qui met en place le tableau avec les différents descripteurs de zone. Le code de chaque zone est fonction de la position du descripteur correspondant à l'intérieur du tableau. Il est automatiquement attribué par la procédure/fonction MouDefZone. La première zone reçoit le code 0, la deuxième le code 1, etc. Le code AUCUNE_ZONE est attribué aux zones de l'écran qui ne sont couvertes par aucun des descripteurs de zone.

Pour mettre en place un tableau de description de zone, et notamment pour définir les masques curseur et écran dans le tableau PtrMask, vous pouvez vous aider dans le programme C de diverses macros et constantes, qui vous sont également offertes par le programme Pascal, mais sous forme de fonctions et constantes. La macro ou fonction MouPtrMask se charge de produire une variable du type PTRVIEW, telle que le tableau PtrMask l'attend à l'intérieur d'un descripteur de zone. Elle attend comme premier paramètre le masque curseur et le masque écran pour le caractère sous la forme duquel le curseur de la souris devra apparaître sur l'écran.

Si vous spécifiez ici PtrSameChar, le curseur de la souris apparaîtra toujours sous la forme du caractère qu'il recouvre. Si vous souhaitez un autre curseur, vous pouvez disposer, avec PtrDifChar, que le curseur de la souris apparaîtra toujours sous forme d'un caractère déterminé. Lors de l'appel de PtrDifChar, vous spécifiez pour cela le code ASCII du caractère voulu.

MouPtrMask attend comme second paramètre les masques curseur et écran pour la couleur du curseur de la souris. Ici aussi, différentes options sont possibles :

✓ Avec PtrSameCol, le curseur de la souris prendra la couleur du caractère recouvert.

- ✓ PtrSameColB produira également un curseur de la souris prenant la couleur du caractère recouvert, mais le bit 7 de l'octet d'attribut sera en outre fixé sur 1, de façon à ce que le caractère clignote ou soit affiché avec une couleur de fond plus intense.

- ✓ PtrInvCol fait apparaître le curseur de la souris dans la couleur inverse du caractère recouvert.

- ✓ PtrDifCol fera prendre au curseur la couleur dont le code est spécifiée à la suite de PtrDifCol.

En dehors des différentes zones de la souris enregistrées à travers la procédure/fonction MouDefZone, un curseur de la souris déterminé peut aussi être attribué au reste de l'écran, autrement dit à la zone portant le code AUCUNE_ZONE. Un programme peut recourir à cet effet à la procédure/fonction MouSetDefaultPtr, à laquelle les masques curseur et écran du curseur de la souris doivent être transmis sous forme d'un paramètre du type PTRVIEW. Pour composer ce paramètre, vous pouvez également vous servir des constantes et macros ou fonctions décrites plus haut.

C'est le gestionnaire d'événements en langage évolué, MouEventHandler, qui se charge de commuter sur les masques curseur et écran de la zone correspondante. Comme il est appelé chaque fois que survient un événement souris, y compris donc lorsque la souris est déplacée, il peut déterminer chaque fois dans quelle zone souris le curseur de la souris se trouve au moment précis. Pour que cette opération se déroule le plus vite possible, il n'examine pas le tableau de zones tout entier pour vérifier chaque fois si la position du curseur de la souris se situe à l'intérieur de la zone de la souris considérée. Il utilise en effet un buffer de zones interne, que MouInit met en place lorsqu'elle est appelée. Ce buffer reflète exactement la structure de la RAM vidéo et contient pour chaque position de l'écran un octet contenant le code de la zone dont dépend cette position. Le gestionnaire d'événements peut ainsi utiliser la position actuelle du curseur de la souris comme index sur ce buffer de zones et déterminer ainsi, avec un seul accès à la mémoire, dans quelle zone de la souris le curseur de la souris se trouve actuellement. Il stocke alors le code de zone ainsi obtenu dans la variable globale MouZone, et l'utilise ensuite comme index pour le tableau des descripteurs de la souris, dont il tire ainsi les masques curseur et écran définis pour cette zone.

Une autre tâche, peut-être encore plus importante, incombe au gestionnaire d'événements en langage évolué : la gestion de la variable MouEvent, dans laquelle il retrace les événements souris actuels. Il ne peut se contenter pour cela de copier purement et simplement le Mouse Event qui lui est transmis par le driver de la souris, à travers le gestionnaire d'événements en assembleur AssHand, sous forme de la variable EvFlags. Cette variable reflète en effet uniquement l'événement actuel, mais pas les événements qui étaient déjà intervenus auparavant. Lorsque, par exemple, on appuie d'abord sur le bouton gauche de la souris, qu'on le tient enfoncé et qu'on n'appuie qu'ensuite sur le bouton droit de la souris, on provoque deux appels du gestionnaire d'événements, qui signalent chacun qu'un bouton de la souris a été appuyé. Rien n'indique cependant lors

du second appel que le bouton gauche est maintenu enfoncé, car seul l'événement »bouton droit de la souris appuyé« est annoncé.

C'est pourquoi le gestionnaire d'événements doit absolument isoler les différents événements retracés par EvFlags, et ne reprendre dans la variable MouEvent que les événements nouvellement intervenus. Cette variable reflétera ainsi à tout moment l'état des boutons de la souris et les déplacements ou l'immobilité du curseur de la souris. MouEvent convient ainsi parfaitement à l'une des fonctions principales du test de la souris, à savoir l'attente de l'intervention d'un événement déterminé. Ce n'est d'ailleurs pas tellement le déplacement de la souris, mais plutôt le fait d'appuyer ou de relâcher les boutons de la souris qui devra être interprété, dans le contexte du moment, comme un message adressé au programme par l'utilisateur.

Sur ce plan aussi, le module de la souris vous soutient avec une fonction appropriée. MouEventWait est le nom d'une fonction qui attend que surviennent les événements spécifiés par le masque de bits transmis. Ce masque de bits peut être défini à l'aide d'une combinaison OU des constantes suivantes :

```
EV_MOU_MOVE       Déplacement de la souris
EV_LEFT_PRESS     Bouton gauche de la souris appuyé
EV_LEFT_REL       Bouton gauche de la souris relâché
EV_RIGHT_PRESS    Bouton droit de la souris appuyé
EV_RIGHT_REL      Bouton droit de la souris relâché
```

Vous pouvez indiquer à la procédure/fonction, à l'aide d'un autre paramètre si vous voulez attendre que tous les événements soient survenus ou si vous vous contentez qu'intervienne au moins un des événements spécifiés. Vous pouvez spécifier ici les constantes ET ou OU, qui correspondent aux combinaisons logiques ainsi dénommées. Le résultat de la fonction vous indique lesquels des événements attendus sont intervenus, sous forme d'un masque de bits dans lequel chaque bit représente un événement. Ce résultat peut être testé à l'aide des constantes indiquées plus haut.

Pour que le travail avec ces procédures/fonctions ne se limite pas à la théorie, le programme principal Turbo Pascal ainsi que celui du module C comportent une petite démonstration des différentes procédures/fonctions. Pour plus de détails, nous vous invitons à vous reporter aux différents listings que voici.

Listing : SOURISP.PAS

```
{*****************************************************************}
{*                        S O U R I S . P A S                    *}
{*---------------------------------------------------------------*}
{*  Fonction      : Fournit diverses fonctions pour le travail   *}
{*                  avec la souris                               *}
{*---------------------------------------------------------------*}
{*  Auteur        : MICHAEL TISCHER                              *}
{*  Développé le  : 21/04/1989                                   *}
{*  Dernière modif.: 23/04/1989                                  *}
{*---------------------------------------------------------------*}
{*  Le module assembleur sourisp.obj doit se trouver dans le     *}
{*  répertoire courant sinon le chemin complet                   *}
{*  doit être spécifier lors de son intégration.                 *}
{*****************************************************************}

uses Dos;                                       { Intégrer unité DOS }

{$L sourispa} { Intégrer le module assembleur }

{-- Déclaration des fonctions externes ---------------------------}

{$F+}                                           { C'est une fonction FAR }
procedure AssHand ; external;                   { Le gestionnaire d'événements en }
{$F-}                                           { assembleur ne plus obliger des fonctions FAR }

{-- Constantes ---------------------------------------------------}

const

{-- Codes d'événement --------------------------------------------}

    EV_MOU_MOVE     =  1;                       { Souris déplacée }
    EV_LEFT_PRESS   =  2;                       { Bouton gauche de la souris appuyé }
    EV_LEFT_REL     =  4;                       { Bouton gauche de la souris relâché }
    EV_RIGHT_PRESS  =  8;                       { Bouton droit de la souris appuyé }
    EV_RIGHT_REL    = 16;                       { Bouton droit de la souris relâché }
    EV_MOU_ALL      = 31;                       { Tous les événements souris }

    LBITS           =  6;                       { EV_LEFT_PRESS or EV_LEFT_REL }
    RBITS           = 24;                       { EV_RIGHT_PRESS or EV_RIGHT_REL }

    AUCUNE_ZONE     = 255;                      { Curseur de la souris pas dans zone xy }

    PtrSameChar     = $00ff;                    { Même caractère }
    PtrSameCol      = $00ff;                    { Même couleur }
    PtrInvCol       = $7777;                    { Couleur inversée }
    PtrSameColz     = $807f;                    { Même couleur clignotante }
    PtrInvColz      = $f777;                    { Couleur inversée clignotante }

    ET              = 0;{ Combinaisons d'événements pour MouEventWait }
    OU              = 1;

    ZV              = #13#10;                   { Saut de ligne }

{-- Déclarations de type -----------------------------------------}

type  FCTPTR   = longint;                       { Adresse d'une fonction FAR }
      PTRVIEW  = longint;                       { Masque pour le curseur de la souris }
      ZONE     = record                         { Décrit une zone de la souris }
                   x1,                          { Coordonnées du coin supérieur gauche }
                   y1,                          { et du coin inférieur droit de }
                   x2,                          { la zone spécifiée }
                   y2     : byte;
                   PtrMask : PTRVIEW;           { Masque pour le curseur souris }
                 end;
      TABLEAUZONES = array [0..100] of ZONE;
      PTRZONE  = ^TABLEAUZONES;
      PTRREC   = record                         { Sert à accéder aux }
                   Ofs : word;                  { éléments d'un pointeur }
                   Seg : word;                  { quelconque }
                 end;
      PTRVREC  = record                         { Sert à accéder aux }
                   ScreenMask : word;           { éléments d'un PTRVIEW }
                   CursorMask : word;
                 end;
      BUFZONE  = array [0..10000] of byte;      { Buffer de zone }
      PTRBZ    = ^BUFZONE;                      { Pointeur sur un buffer de zone }

{-- Variables globales -------------------------------------------}

var   NmbZones,                                 { Nombre de zones }
      TLine,                                    { Nombre de lignes de texte }
      TCol     : byte;                          { Nombre de colonnes de texte }
      MouAvail : boolean;                       { Est TRUE, si souris disponible }
      OldPtr,                                   { Ancienne apparence du curseur de la souris }
      StdPtr   : PTRVIEW;{ Masque pour le curseur de la souris standard }
      BufPtr   : PTRBZ;  { Pointeur sur buffer pour identification zone }
      PtrZonAct: PTRZONE;                       { Pointeur sur vecteur de zone actuel }
      BLen     : integer;                       { Longueur du buffer de zone en octets }
      ExitOld  : pointer;                       { Pointeur sur l'ancienne procédure Exit }

{ Variables définies lors de chaque appel du gestionnaire de la souris }

      ZonSou,                                   { Zone de souris actuelle }
      MouCol,                                   { Colonne de la souris (écran de texte) }
      MouRow   : byte;                          { Ligne de la souris (écran de texte) }
      MouEvent : integer;                       { Masque d'événements }

{-- Variables qui ne sont définies par le gestionnaire de la souris ---}
{-- que lorsqu'intervient un événement attendu                     ---}

      ZonEv,                                    { Zone dans laquelle figure la souris }
      EvCol,                                    { Colonne de la souris }
      EvRow : byte;                             { Ligne de la souris }

{*****************************************************************}
{*  MouPtrMask: Compose les masques curseur et écran à partir d'un *}
{*              masque de bits pour le caractère et pour la couleur *}
{*---------------------------------------------------------------*}
{*  Entrée : Caractere - masque bits pour les masques curseur et *}
{*                       écran concernant le caractère           *}
{*           Couleur   - masque bits pour les masques curseur et *}
{*                       écran concernant la couleur du caractère *}
{*  Sortie : les masques curseur et écran sous forme d'une valeur du *}
{*           type PtrView                                        *}
{*  Infos :  pour Caractere et Couleur peuvent être transmises les *}
{*           constantes PtrSameChar, PtrSameCol, PtrSameColz,    *}
{*           PtrInvCol et PtrInvColz ainsi que les résultats des *}
{*           fonctions PtrDifChar et PtrDifCol                   *}
{*****************************************************************}

function MouPtrMask( Caractere, Couleur  : word ) : PTRVIEW;

var Mask : PTRVIEW;                             { Les masques curseur et écran à créer }

begin
  PTRVREC( Mask ).ScreenMask := ( ( Couleur and $ff ) shl 8 ) +
                                ( Caractere and $ff );
  PTRVREC( Mask ).CursorMask := ( Couleur and $ff00 ) + ( Caractere shr 8
);
  MouPtrMask := Mask;                           { Renvoyer masque au programme d'appel }
end;

{*****************************************************************}
{*  PtrDifChar: crée la partie caractère des masques curseur et écran *}
{*              concernant le caractère                          *}
{*---------------------------------------------------------------*}
{*  Entrée : Code ASCII du caractère devant être le curseur souris *}
{*  Sortie : Masques curseur et écran pour ce caractère          *}
{*  Infos :  Le résultat de la fonction devra être retraité à l'aide *}
{*           de la fonction MouPtrMask                           *}
{*****************************************************************}

function PtrDifChar( Caractere : byte ) : word;

begin
  PtrDifChar := Caractere shl 8;
end;

{*****************************************************************}
{*  PtrDifCol: Crée la partie caractère des masques curseur et écran *}
{*             concernant la couleur du curseur de la souris     *}
{*---------------------------------------------------------------*}
{*  Entrée : Couleur du caractère devant être le curseur de la souris *}
{*  Sortie : Masques curseur et écran pour cette couleur         *}
{*  Infos  : Le résultat de la fonction devra être retraité à l'aide *}
{*           de la fonction MouPtrMask                           *}
{*****************************************************************}

function PtrDifCol( Couleur : byte ) : word;

begin
  PtrDifCol := Couleur shl 8;
end;

{*****************************************************************}
{*  MouDefinePtr: transmet au driver de la souris les masques curseur *}
{*                et écran qui définissent désormais l'apparence du *}
{*                curseur de la souris                           *}
{*---------------------------------------------------------------*}
{*  Entrée : Masque - les masques curseur et écran sous forme d'un *}
{*                    paramètre du type PTRVIEW                  *}
{*  Infos :  - le paramètre Masque doit avoir été créé à l'aide de la *}
{*             fonction MouPtrMask                               *}
```

```
!(*          - les 16 bits de plus fort poids de Masque représentent  *)
!(*            le masque écran, les 16 bits faibles le masque curseur *)
!(*****************************************************************)

!procedure MouDefinePtr( Mask : PTRVIEW );

!var Regs : Registers;    ( Registres du processeur pour l'appel d'int. )

!begin
!  if OldPtr <> Mask then ( Modification par rapport au dernier appel ? )
!    begin                                                    ( Oui )
!      Regs.AX := $000a;          ( N° fct pour "Set text pointer type" )
!      Regs.BX := 0;                                ( Fixer curseur logiciel )
!      Regs.CX := PTRVREC( Mask ).ScreenMask; ( Mot faible : masque AND )
!      Regs.DX := PTRVREC( Mask ).CursorMask;   ( Mot fort : masque XOR )
!      Intr( $33, Regs);                  ( Appeler le driver de la souris )
!      OldPtr := Mask;                     ( Ranger le nouveau masque bits )
!    end;
!end;

!(*****************************************************************)
!(*  MouEventHandler: Est appelé par le driver de la souris à travers  *)
!(*                   la routine assembleur AssHand dès qu'intervient  *)
!(*                   un événement concernant la souris                *)
!(*-----------------------------------------------------------------*)
!(*  Entrée : EvFlags  = le masque Event                              *)
!(*           ButState = état actuel des boutons de la souris         *)
!(*           X, Y     = position actuelle du curseur de la souris    *)
!(*                      concernant l'écran de texte                  *)
!(*****************************************************************)

!procedure MouEventHandler( EvFlags, ButState, x, y : integer );

!var NouZon : byte;                          ( Numéro de la nouvelle zone )

!begin

!  MouEvent := MouEvent and not(1);                ( Masquer le bit 0 )
!  MouEvent := MouEvent or ( EvFlags and 1 );      ( Copier le bit 0 )

!  if ( EvFlags and LBITS ) <> 0 then (Bouton gauche relâché ou appuyé ?)
!    begin                                                    ( Oui )
!      MouEvent := MouEvent and not( LBITS );      ( Masque état actuel)
!      MouEvent := MouEvent or ( EvFlags and LBITS );  ( Incruster état )
!    end;

!  if ( EvFlags and RBITS ) <> 0 then ( Bouton droit relâché ou appuyé ?)
!    begin                                                    ( Oui )
!      MouEvent := MouEvent and not( RBITS );      ( Masque état actuel)
!      MouEvent := MouEvent or ( EvFlags and RBITS );  ( Incruster état )
!    end;

!  MouCol := x;                  ( Convertir colonne en colonnes de texte )
!  MouRow := y;                    ( Convertir ligne en lignes de texte )

!  {-- Déterminer zone dans laquelle figure la souris et examiner    ----}
!  {-- si la zone a été modifiée depuis le dernier appel du          ----}
!  {-- gestionnaire. Dans ce cas, l'apparence du curseur de la       ----}
!  {-- souris doit être redéfinie.                                   ----}

!  NouZon := BufPtr^[ MouRow * TCol + MouCol ];          ( Retirer zone )
!  if NouZon <> ZonSou then                            ( Nouvelle zone ? )
!    begin                                                    ( Oui )
!      if NouZon = AUCUNE_ZONE then              ( En dehors d'une zone ? )
!        MouDefinePtr( StdPtr )     ( Oui, curseur de la souris standard )
!      else                                                ( Non, zone connue )
!        MouDefinePtr( PtrZonAct^[ NouZon ].PtrMask );
!    end;
!  ZonSou := NouZon;       ( Ranger numéro de zone dans variable globale )
!end;

!(*****************************************************************)
!(*  MouIBuffFill: stocke le code d'une zone de la souris à l'intérieur *)
!(*                de la mémoire de zone interne du module            *)
!(*-----------------------------------------------------------------*)
!(*  Entrée : x1, y1 = coin supérieur gauche de la zone de la souris  *)
!(*           x2, y2 = coin inférieur droit de la zone de la souris   *)
!(*           Code   = le code de zone                                *)
!(*****************************************************************)

!procedure MouIBuffFill( x1, y1, x2, y2, Code : byte );

!var Index   : integer;                        ( Désigne le champ traité )
!    Colonne,                                    ( Compteur de boucle )
!    Ligne   : byte;

!begin
!  for Ligne:=y1 to y2 do              ( Parcourir les différentes lignes )
!    begin
!      Index := Ligne  * TCol + x1;          ( Premier index de la ligne )
!      for Colonne:=x1 to x2 do     ( Parcourir les colonnes de la ligne )
!        begin
!          BufPtr^[ Index ] := Code;                      ( Stocker code )
!          inc( Index );                  ( Fixer index sur champ suivant )
!        end;
!    end;
!end;

!(*****************************************************************)
!(*  MouDefZone : permet d'enregistrer différentes zones de l'écran,  *)
!(*               qui seront identifiées comme zones de souris        *)
!(*               distinctes et pourront être dotées d'un curseur de  *)
!(*               la souris particulier                               *)
!(*-----------------------------------------------------------------*)
!(*  Entrée : Nombre = Nombre de zones de l'écran                     *)
!(*           ZPtr   = Pointeur sur le tableau dans lequel les diffé- *)
!(*                    rentes zones sont chacune décrites par une     *)
!(*                    structure du type ZONE                         *)
!(*  Infos :  - le code AUCUNE_ZONE est attribué aux zones de l'écran *)
!(*             restées libres                                        *)
!(*           - lorsque le curseur de la souris pénètre dans l'une des *)
!(*             zones souris inscrites dans le tableau, le          *)
!(*             gestionnaire d'événements commute immédiatement sur   *)
!(*             la forme du curseur de la souris dans cette zone      *)
!(*****************************************************************)

!procedure ZoneMouDef( Nombre : byte; PtrZ : PTRZONE );

!var ActZon,                                  ( Numéro de la zone actuelle )
!    Zone : byte;                                  ( Compteur de boucle )

!begin
!  PtrZonAct := PtrZ;                        ( Ranger pointeur sur vecteur )
!  NmbZones := Nombre;                                ( et nombre de zones )
!  FillChar( BufPtr^, BLen, AUCUNE_ZONE );(Tous les éléments=AUCUNE_ZONE)
!  for Zone:=0 to nombre-1 do          ( Parcourir les différentes zones )
!    with PtrZ^[ Zone ] do
!      MouIBuffFill( x1, y1, x2, y2, Zone );

!  {-- Redéfinir le curseur de la souris ------------------------------}
!  ActZon := BufPtr^[ MouRow * TCol + MouCol ];          ( Retirer zone )
!  if ActZon = AUCUNE_ZONE then                ( En dehors d'une zone ? )
!    MouDefinePtr( StdPtr )          ( Oui, curseur de la souris standard )
!  else                                                ( Non, zone connue )
!    MouDefinePtr( PtrZ^[ ActZon ].PtrMask );
!end;

!(*****************************************************************)
!(*  MouEventWait: attend l'intervention d'un événement déterminé     *)
!(*                concernant la souris                               *)
!(*-----------------------------------------------------------------*)
!(*  Entrée : TYP        = type de combinaison entre les différents   *)
!(*                        événements                                 *)
!(*           WAIT_EVENT = masque bits spécifiant les événements à    *)
!(*                        attendre                                   *)
!(*  Sortie : masque bits de l'événement intervenu                    *)
!(*  Infos :  - WAIT_EVENT peut être créé par combinaison OU des      *)
!(*             diverses constantes telles que EV_MOU_MOVE,           *)
!(*             EV_LEFT_PRESS, etc.                                   *)
!(*           - les constantes ET et OU peuvent être spécifiées comme *)
!(*             type de combinaison. Si vous optez pour ET, la        *)
!(*             fonction ne rendra la main au programme d'appel qu'une *)
!(*             fois que tous les événements seront intervenus. Avec  *)
!(*             OU, l'intervention d'un au moins des événements       *)
!(*             spécifiés suffira.                                    *)
!(*****************************************************************)

!function MouEventWait( Typ : BYTE; WaitEvent : integer ) : integer;

!var ActEvent : integer;
!    Ligne,
!    Colonne  : byte;
!    Fin      : boolean;

!begin
!  Colonne := MouCol;       ( Ranger la position actuelle de la souris )
!  Ligne   := MouRow;
!  Fin     := false;

!  repeat
!    {-- Attendre jusqu'à ce qu'un des événements se produise ----------}

!    if Typ = ET then                                  ( Combinaison ET ? )
!      repeat                ( Oui, tous les événements doivent se produire )
!        ActEvent := MouEvent;                ( Rechercher événement actuel )
!      until ActEvent = WaitEvent
!    else                                              ( Combinaison OU )
!      repeat                        ( Un événement au moins doit survenir )
!        ActEvent := MouEvent;                ( Rechercher événement actuel )
!      until ( ActEvent and WaitEvent ) <> 0;

!    ActEvent := ActEvent and WaitEvent;{Laisser que les bits événements}

!    {-- Si on attend un déplacement de la souris, l'événement n'est -- }
!    {-- accepté que si le curseur de la souris a été déplacé vers   -- }
!    {-- une autre ligne et/ou une autre colonne de l'écran de texte - }
```

```
{
    if ( ( (WaitEvent and EV_MOU_MOVE) <> 0 )  and
         ( Colonne = MouCol ) and  ( Ligne = MouRow ) ) then
      begin                  { Souris déplacée mais même position écran }
        ActEvent := ActEvent and not( EV_MOU_MOVE ); { Masquer bit move }
        Fin := ( ActEvent <> 0);          { Reste-t-il des événements ? }
      end
    else                                          { Evénement intervenu }
      Fin := TRUE;
  until Fin;

  EvCol := MouCol;                          { Conserver position et zone  }
  EvRow := MouRow;                          { actuelles de la souris dans }
  ZonEv := ZonSou;                          { des variables globales      }

  MouEventWait := ActEvent;
end;

{*****************************************************************}
{* MoulSetEventHandler: Installe un gestionnaire d'événements appelé *}
{*                      par le driver de la souris lorsque survien-  *}
{*                      nent certains événements souris.             *}
{*---------------------------------------------------------------**}
{* Entrée : EVENT = masque bits décrivant les événements dont     *}
{*                  l'intervention doit entraîner un appel du     *}
{*                  gestionnaire d'événements                     *}
{*          FPTR  = Pointeur sur gestionnaire évén. du type FCTPTR *}
{* Infos :  - EVENT peut être créé par combinaison OU des diverses *}
{*            constantes telles que EV_MOU_MOVE, EV_LEFT_PRESS etc. *}
{*          - le gestionnaire d'évén. doit être une procédure FAR et *}
{*            ne modifier aucun des reg. processeur reçus          *}
{*****************************************************************}

procedure MoulSetEventHandler( Event : integer; PtrC : FCTPTR );

var Regs : Registers; { Registres du processeur pour appel d'interrup. }

begin
  Regs.AX := $000C;                  { Nº fct. pour "Set Mouse Handler" }
  Regs.CX := event;                        { Charger masque d'événements }
  Regs.DX := PTRREC( PtrC ).Ofs;     { Adresse d'offset du gestionnaire }
  Regs.ES  := PTRREC( PtrC ).Seg; { Adresse de segment du gestionnaire }
  Intr( $33,  Regs );                  { Appeler le driver de la souris }
end;

{*****************************************************************}
{* MoulGetX: Détermine la colonne de texte dans laquelle figure le  *}
{*           curseur de la souris                                    *}
{*---------------------------------------------------------------**}
{* Sortie : Colonne du curseur de la souris dans l'écran de texte  *}
{*****************************************************************}

function MoulGetX : byte;

var Regs : Registers; { Registres du processeur pour appel d'interrup. }

begin
  Regs.AX := $0003;                { Nº fct.: pour "Get mouse position" }
  Intr( $33,  Regs );                 { Appeler le driver de la souris }
  MoulGetX := Regs.CX shr 3;           { Convertir colonne et renvoyer }
end;

{*****************************************************************}
{* MoulGetY: Déterminer ligne de texte du curseur de la souris     *}
{*---------------------------------------------------------------**}
{* Sortie : Ligne du curseur de la souris dans l'écran de texte    *}
{*****************************************************************}

function MoulGetY : byte;

var Regs : Registers;{ Registres du processeur pour appel d'interrupt. }

begin
  Regs.AX := $0003;                { Nº fct.: pour "Get mouse position" }
  Intr( $33, Regs );                  { Appeler le driver de la souris }
  MoulGetY := Regs.DX shr 3;           { Convertir ligne et renvoyer }
end;

{*****************************************************************}
{* MouShowMouse: Place le curseur de la souris sur l'écran         *}
{*---------------------------------------------------------------**}
{* Infos : Les appels de MouShowMouse et MouHideMouse doivent      *}
{*         s'équilibrer pour avoir un effet                        *}
{*****************************************************************}

procedure MouShowMouse;

var Regs : Registers;{ Registres du processeur pour appel d'interrupt. }

begin
  Regs.AX := $0001;                          { Nº fct.: pour "Show Mouse" }
  Intr( $33, Regs );                        { Appeler le driver de la souris }
end;

{*****************************************************************}
{* MouHideMouse: Elimine le curseur de la souris de l'écran        *}
{*---------------------------------------------------------------**}
{* Infos : Les appels de MouShowMouse et MouHideMouse doivent      *}
{*         s'équilibrer pour avoir un effet                        *}
{*****************************************************************}

procedure MouHideMouse;

var Regs : Registers;{ Registres du processeur pour appel d'interrupt. }

begin
  Regs.AX := $0002;                          { Nº fct. pour "Hide Mouse" }
  Intr( $33, Regs);                        { Appeler le driver de la souris }
end;

{*****************************************************************}
{* MouSetMoveArea: Fixe la zone de déplacement pour le curseur de  *}
{*                 la souris                                       *}
{*---------------------------------------------------------------**}
{* Entrée :  x1, y1 = Coordonnées coin supérieur gauche de la zone *}
{*           x2, y2 = Coordonnées coin inférieur droit de la zone  *}
{* Infos :  - Les ordonnées se rapportent à l'écran de texte et non *}
{*            à l'écran graphique virtuel du driver de la souris   *}
{*****************************************************************}

procedure MouSetMoveArea( x1, y1, x2, y2 : byte );

var Regs : Registers;{ Registres du processeur pour appel d'interrupt. }

begin
  Regs.AX := $0008;                  { Nº fct. pour "Set vertical limits" }
  Regs.CX := integer( y1 ) shl 3;              { Conversion vers l'écran }
  Regs.DX := integer( y2 ) shl 3;                    { virtuel de la souris }
  Intr( $33, Regs );                       { Appeler le driver de la souris }
  Regs.AX := $0007;               { Nº fct. pour "Set horizontal limits" }
  Regs.CX := integer( x1 ) shl 3;              { Conversion vers l'écran }
  Regs.DX := integer( x2 ) shl 3;                    { virtuel de la souris }
  Intr( $33, Regs );                       { Appeler le driver de la souris }
end;

{*****************************************************************}
{* MouSetSpeed: Fixe le rapport entre le déplacement de la souris et *}
{*              le mouvement du curseur de la souris qui doit en   *}
{*              résulter                                           *}
{*---------------------------------------------------------------**}
{* Entrée : XSpeed = vitesse dans le sens des X                    *}
{*          YSpeed = vitesse dans le sens des Y                    *}
{* Infos :  - Les deux paramètres sont exprimés en unités          *}
{*            mickey / 8 points écran                              *}
{*****************************************************************}

procedure MouSetSpeed( XSpeed, YSpeed : integer );

var Regs : Registers;{ Registres du processeur pour appel d'interrupt. }

begin
  Regs.AX := $000f;                { Nº fct. "Set mickeys to pixel ratio" }
  Regs.CX := XSpeed;
  Regs.DX := YSpeed;
  Intr( $33, Regs);                          { Appeler le driver de la souris }
end;

{*****************************************************************}
{* MouMovePtr: Amène le curseur de la souris dans un emplacement   *}
{*             déterminé de l'écran                                *}
{*---------------------------------------------------------------**}
{* Entrée : COL = la nouvelle colonne d'écran du curseur souris    *}
{*          ROW = la nouvelle ligne d'écran du curseur souris      *}
{* Infos :  - Les ordonnées se rapportent à l'écran de texte et non *}
{*            à l'écran graphique virtuel du driver de la souris   *}
{*****************************************************************}

procedure MouMovePtr( Col, Row : byte );

var Regs   : Registers;(Registres du processeur pour appel d'interrupt.)
    NouZon : byte;             { Zone dans laquelle la souris est amenée }

begin
  Regs.AX := $0004;               { Nº fct. "Set mouse pointer position" }
  MouCol := col;                            { Stocker coordonnées dans }
  MouRow := row;                             { des variables globales }
  Regs.CX := integer( col ) shl 3;          { Convertir coordonnées et }
  Regs.DX := integer( row ) shl 3; {stocker dans des variables globales}
  Intr( $33, Regs );                        { Appeler le driver de la souris }

  NouZon := BufPtr^[ Row * TCol + Col ];                  { Retirer zone }
  if NouZon <> ZonSou then                              { Nouvelle zone ? }
    begin                                                       { Oui }
      if NouZon = AUCUNE_ZONE then              { En dehors d'une zone ? }
        MouDefinePtr( StdPtr )     { Oui, curseur de la souris standard }
      else                                                  { Non, zone connue }
```

```
!        MouDefinePtr( PtrZonAct^[ NouZon ],PtrMask );
!    end;
!  ZonSou := NouZon;       { Ranger numéro de zone dans variable globale }
!end;
!
!{*****************************************************************}
!{*  MouSetDefaultPtr: Définit l'apparence du curseur de la souris  *}
!{*                    pour les zones de l'écran qui n'ont pas été  *}
!{*                    désignées explicitement comme des zones souris *}
!{*---------------------------------------------------------------*}
!{*  Entrée : standard = masques curseur et écran pour curseur souris *}
!{*  Infos :  - Le paramètre transmis doit avoir été créé avec la  *}
!{*             fonction MouPtrMask                               *}
!{*****************************************************************}
!
!procedure MouSetDefaultPtr( Standard : PTRVIEW );
!
!begin
!  StdPtr := Standard;       { Ranger masque bits dans variable globale }
!
!  {-- Si la souris ne figure actuellement dans aucune zone, on     ----}
!  {-- commute directement sur nouvelle apparence                   ----}
!
!  if ZonSou = AUCUNE_ZONE then                  { Dans aucune zone ? }
!    MouDefinePtr( Standard );                                { Non }
!end;
!
!{*****************************************************************}
!{*  MouEnd: Est appelée pour terminer le travail avec les fonctions *}
!{*          et procédures du module de la souris                 *}
!{*---------------------------------------------------------------*}
!{*  Infos : - La procédure n'a pas besoin d'être appelée explicite- *}
!{*            ment par le programme d'application car la fonction *}
!{*            MouInit la définit comme procédure Exit            *}
!{*****************************************************************}
!
!{$F+}     { Doit être FAR pour permettre un appel comme procédure EXIT }
!
!procedure  MouEnd;
!
!var Regs : Registers;{ Registres du processeur pour appel d'interrupt. }
!
!begin
!  MouHideMouse;             { Eliminer curseur de la souris de l'écran }
!  Regs.AX := 0;              { Réinitialisation du driver de la souris }
!  Intr( $33, Regs);                  { Appeler le driver de la souris }
!
!  FreeMem( BufPtr, BLen );         { Libérer à nouveau mémoire allouée }
!
!  ExitProc := ExitOld;           { Installer à nouveau procédure Exit }
!end;
!
!{$F-}                                      { Plus de procédures FAR }
!
!{*****************************************************************}
!{*  MouInit: Dirige le travail avec les différentes fonctions et   *}
!{*           procédures du module de la souris et initialise les   *}
!{*           différentes variables                                 *}
!{*---------------------------------------------------------------*}
!{*  Entrée : Colonnes = Nombre de colonnes de l'écran             *}
!{*           Lignes   = Nombre de lignes de l'écran               *}
!{*  Sortie : TRUE si un driver de la souris est installé, sinon FALSE *}
!{*  Infos :  - Cette fonction doit être la première des différentes *}
!{*             procédures et fonctions de ce module à être appelée *}
!{*             par un programme d'application                    *}
!{*****************************************************************}
!
!function MouInit( Colonnes, Lignes : byte ) : boolean;
!
!var Regs : Registers; { Registres du processeur pour appel d'interrup. }
!
!begin
!  TLine := Lignes;                  { Stocker nombre de lignes et de }
!  TCol  := Colonnes;              { colonnes dans des variables globales }
!
!  ExitOld := ExitProc;          { Ranger adresse de la procédure Exit }
!  ExitProc := @MouEnd;          { Définir MouEnd comme procédure Exit }
!
!  {-- Allouer et remplir buffer pour les zones souris ----------------}
!
!  BLen := TLine * TCol;              { Nombre de caractères dans l'écran }
!  GetMem( BufPtr, BLen );             { Allouer buffer de zone interne }
!  MouBuffFill( 0, 0, TCol-1, TLine-1, AUCUNE_ZONE );
!
!  Regs.AX := 0;                      { Initialiser driver de la souris }
!  Intr( $33, Regs );                  { Appeler le driver de la souris }
!  MouInit := ( Regs.AX <> 0 );        { Driver de la souris installé ? }
!
!  MouSetMoveArea( 0, 0, TCol-1, TLine-1 ); { Fixer zone de déplacement }
!
!  MouCol  := MouGetX;                          { Charger position souris }
!  MouRow  := MouGetY;                  { actuelle dans variables globales }
!  ZonSou  := AUCUNE_ZONE;      { Curseur de la souris dans aucune zone }
!  MouEvent := EV_LEFT_REL or EV_RIGHT_REL;{ Pas bouton de souris appuyé}
!  StdPtr  := MouPtrMask( PTRSAMECHAR, PTRINVCOL );   { Curseur souris }
!  OldPtr  := PTRVIEW( 0 );
!
!  {-- Installer le gestionnaire d'événements assembleur "AssHand" ------}
!  MouISetEventHandler( EV_MOU_ALL, FCTPTR(@AssHand) );
!
!end;
!
!{*****************************************************************}
!{*                    P R O G R A M M E   P R I N C I P A L      *}
!{*****************************************************************}
!
!const Zones : array[0..4] of ZONE =          { Les zones de la souris }
!  (
!    ( x1:  0; y1:  0; x2: 79; y2:  0 ),              { Ligne du haut }
!    ( x1:  0; y1:  1; x2:  0; y2: 23 ),          { Colonne de gauche }
!    ( x1:  0; y1: 24; x2: 78; y2: 24 ),               { Ligne du bas }
!    ( x1: 79; y1:  1; x2: 79; y2: 23 ),         { Colonne de droite }
!    ( x1: 79; y1: 24; x2: 79; y2: 24 )          { Coin inférieur droit }
!  );
!
!var Dummy : integer;              { Reçoit le résultat de MouEventWait }
!
!begin
!  {-- Fixer curseur de la souris pour différentes zones de la souris --}
!  Zones[ 0 ].PtrMask := MouPtrMask( PtrDifChar($18), PtrInvCol);
!  Zones[ 1 ].PtrMask := MouPtrMask( PtrDifChar($1b), PtrInvCol);
!  Zones[ 2 ].PtrMask := MouPtrMask( PtrDifChar($19), PtrInvCol);
!  Zones[ 3 ].PtrMask := MouPtrMask( PtrDifChar($1a), PtrInvCol);
!  Zones[ 4 ].PtrMask := MouPtrMask( PtrDifChar($58), PtrDifCol($40));
!
!  writeln(@13@10,'SOURIS - (c) 1989 by MICHAEL TISCHER'@13@10);
!  if MouInit( 80, 25 ) then          { Initialiser module de la souris }
!  begin                { Tout va bien, un driver de la souris est installé }
!    writeln('Si vous déplacez le curseur de la souris sur',ZV,
!            'l''écran, surtout le long des bords, vous constaterez',ZV,
!            'que l''apparence du curseur de la souris se modifie en',ZV,
!            'fonction de sa position.',ZV,ZV,
!            'Pour mettre fin à cette démo, amenez le curseur de la',ZV,
!            'souris dans le coin inférieur droit de l''écran et appuyez'
!          ,ZV,'alors simultanément sur les boutons gauche et droit de',ZV,
!            'la souris.' );
!
!    MouSetDefaultPtr( MouPtrMask( PtrDifChar( $DB ), PtrDifCol( 3 ) ) );
!    ZoneMouDef( 5, @Zones );                              { Déf. zones }
!    MouShowMouse;          { Afficher curseur de la souris sur l'écran }
!
!    {-- Attendre que boutons gauche et droit soient appuyés en       ---}
!    {-- même temps et que le curseur de la souris se trouve à ce     ---}
!    {-- moment dans la zone 4                                        ---}
!
!    repeat                                          { Boucle de test }
!      Dummy := MouEventWait( ET, EV_LEFT_PRESS or EV_RIGHT_PRESS );
!    until ZonEv = 4;
!  end
!  else               { Pas de souris ou pas de driver souris installé }
!    writeln('Aucun driver souris n''est installé !');
!end.
```

Listing : SOURISPA.ASM

```
;*****************************************************************;
;*                          S O U R I S P A                      *;
;*---------------------------------------------------------------*;
;*   Fonction       : Gestionnaire d'événements appelé par le driver *;
;*                    de la souris. A intégrer dans un programme TP *;
;*---------------------------------------------------------------*;
;*   Auteur         : MICHAEL TISCHER                            *;
;*   Développé le   : 21/04/1989                                 *;
;*   Dernière modif.: 21/04/1989                                 *;
;*---------------------------------------------------------------*;
;*   Assemblage     : MASM /MX SOURISPA;   ou                    *;
;*                    TASM -MX SOURISPA;                         *;
;*                    ... combiner avec le programme PASCAL SOURISP *;
;*****************************************************************;

;-- Segment de données ---------------------------------------------

DATA    segment word public
DATA    ends                            ;Pas de variables dans ce programme

;-- Programme ------------------------------------------------------

CODE    segment byte public                             ;Le segment de programme

        assume CS:CODE      ;CS désigne le segment de code, le contenu de
                                                        ;DS, SS et ES est inconnu

public      AssHand                 ;Donne au programme TP la possibilité de
                                      ;déterminer l'adresse du gestionnaire
                                                        ;en assembleur

extrn       MouEventHandler : near ;Le driver d'événements en TP à appeler

actif       db  0              ;Indique si un appel est actuellement en cours
                                                        ;d'exécution

;-----------------------------------------------------------------------
;-- AssHand : Le gestionnaire d'événements qui est d'abord appelé par le
;--           driver de la souris et appelle alors à son tour la
;--           procédure Turbo Pascal MouEventHandler
;-- Appel en TP : Interdit !

AssHand     proc far
;- Sauvegarder tout d'abord tous les registres du processeur sur la pile

        cmp  actif,0                    ;Appel non encore terminé ?
        jne  ffin                       ;Non --> ne pas permettre l'appel

        mov  actif,1                    ;Ne plus autoriser d'appels

        push ax
        push bx
        push cx
        push dx
        push di
        push si
        push bp
        push es
        push ds

      ;-- Placer sur la pile les arguments pour l'appel de la ------
                                                        ;-- fonction TP
                                                          ;-- Appel :
       ;--   MouEventHandler (EvFlags, ButStatus, x , y : integer );

        push ax                 ;Placer flags d'événements sur la pile
        push bx               ;Etat des boutons de la souris sur la pile

        mov  di,cx                ;Ranger ordonnée horizontale dans DI
        mov  cl,3    ;Compteur de décalages pour numéro de coordonnée

        shr  di,cl              ;Diviser DI (ordonnée horizontale) par 8
        push di                                 ;et placer sur la pile

        shr  dx,cl              ;Diviser DX (ordonnée verticale) par 8
        push dx                                 ;et placer sur la pile

        mov  ax,DATA              ;Placer adresse de segment du segment
        mov  ds,ax    ;de données dans AX et de là dans le registre DS

        call  MouEventHandler                 ;Appel de la procédure TP

        ;-- Retirer de la pile les registres sauvegardés ------------

        pop  ds
        pop  es
        pop  bp
        pop  si
        pop  di
        pop  dx
        pop  cx
        pop  bx
        pop  ax

        mov  actif,0                          ;Appel à nouveau autorisé

ffin:   ret                             ;Retour au driver de la souris

AssHand endp

;-----------------------------------------------------------------------

CODE    ends                                    ;Fin du segment de code
        end                                       ;Fin du programme
```

Listing : SOURIS.C

```
/*****************************************************************/
/*                          S O U R I S C . C                    */
/*---------------------------------------------------------------*/
/*   Fonction       : Fournit différentes fonctions pour le travail  */
/*                    avec la souris                             */
/*---------------------------------------------------------------*/
/*   Auteur         : MICHAEL TISCHER                            */
/*   Développé le   : 20/04/1989                                 */
/*   Dernière modif.: 22/04/1989                                 */
/*---------------------------------------------------------------*/
/*   Microsoft C                                                 */
/*   Création       : CL /AS SOURISC.C SOURISCA.OBJ               */
/*   Appel          : SOURISC                                    */
/*---------------------------------------------------------------*/
/*   Turbo C (environnement intégré)                             */
/*   Création       : avec fichier Project de contenu suivant :  */
/*                      SOURISC                                  */
/*                      SOURISCA.OBJ                             */
/*                    avec le modèle de mémoire SMALL.           */
/*                    Le Stack Checking doit être désactivé.     */
/*   Appel          : SOURIS                                     */
/*****************************************************************/

/*-- Intégrer les fichiers Include ------------------------------*/

#include <dos.h>
#include <stdlib.h>
#include <string.h>
#include <stdio.h>

extern void far AssHand( void );            /* Déclaration externe du */
                                          /* gestionnaire en assembleur */
/*-- Typedefs ---------------------------------------------------*/

typedef unsigned char BYTE;             /* Nous nous bricolons un octet */
typedef unsigned long PTRVIEW; /* Masque pour le curseur de la souris */
typedef struct {                                /* Décrit une zone souris */
                BYTE x1,       /* Coordonnées du coin supérieur gauche */
                     y1,             /* et du coin inférieur droit de */
                     x2,                          /* la zone spécifiée */
                     y2;
               PTRVIEW ptr_mask;    /* Masque pour le curseur souris */
              } ZONE;
typedef void (far * MOUHAPTR)( void ); /* Ptr sur driver d'événements */

/*-- Constantes -------------------------------------------------*/
```

```
!
!#define TRUE  ( 1 == 1 )
!#define FALSE ( 1 == 0 )
!
!/*-- Codes Event ---------------------------------------------------*/
!
!#define EV_MOU_MOVE      1                          /* Souris déplacée */
!#define EV_LEFT_PRESS    2       /* Bouton gauche de la souris appuyé */
!#define EV_LEFT_REL      4      /* Bouton gauche de la souris relâché */
!#define EV_RIGHT_PRESS   8       /* Bouton droit de la souris appuyé */
!#define EV_RIGHT_REL    16      /* Bouton droit de la souris relâché */
!#define EV_MOU_ALL      31              /* Tous les événements souris */
!
!#define AUCUNE_ZONE 255      /* Curseur de la souris pas dans zone xy */
!
!/*-- Macros --------------------------------------------------------*/
!
!#define MouGetCol()       (ev_col)    /* Fournissent position et zone */
!#define MouGetRow()       (ev_row)         /* de souris au moment de */
!#define MouGetZone()      (ev_zon)   /* l'intervention de l'événement */
!#define MouAvail()        ( mavail ) /* Renvoie TRUE si souris existe */
!#define MouGetActCol()    ( moucol )          /* Renvoient chaque fois */
!#define MouGetActRow()    ( mourow )    /* position et zone actuelles */
!#define MouGetActZon()    ( mouzon )                 /* de la souris */
!#define MouIsLeftPress()  ( mouevent & EV_LEFT_PRESS )
!#define MouIsLeftRel()    ( mouevent & EV_LEFT_REL )
!#define MouIsRightPress() ( mouevent & EV_RIGHT_PRESS )
!#define MouIsRightRel()   ( mouevent & EV_RIGHT_REL )
!#define MouSetMoveAreaAll() MouSetMoveArea( 0, 0, tcol-1, tline-1 );

!#define ELVEC(x) ( sizeof(x) / sizeof(x[0]) ) /* Nb d'éléments dans X */

!/*-- Macros pour créer le masque bits servant à définir          ---*/
!/*-- l'apparence du curseur de la souris.                        ---*/
!/*-- L'appel de MouPtrMask se présente par exemple ainsi :       ---*/
!/*--   MouPtrMask( PTRDIFCHAR( 'x' ), PTRINVCOL )               ---*/
!/*-- pour que le curseur de la souris apparaisse sous forme d'un  ---*/
!/*-- petit X avec la couleur inverse du caractère qu'il recouvre. ---*/
!
!#define MouPtrMask( z, f )\
!  ( ((PTRVIEW) f) >> 8 << 24) + (((PTRVIEW) z) >> 8 << 16) +\
!    (((f) & 255) << 8) + ((z) & 255) )
!
!#define PTRSAMECHAR   ( 0x00ff )                  /* Même caractère */
!#define PTRDIFCHAR(z) ( (z) << 8 )               /* Autre caractère */
!#define PTRSAMECOL    ( 0x00ff )                    /* Même couleur */
!#define PTRINVCOL     ( 0x7777 )                 /* Couleur inversée */
!#define PTRSAMECOLC   ( 0x807f )          /* Même couleur clignotante */
!#define PTRINVCOLC    ( 0xf777 )       /* Couleur inversée clignotante */
!#define PTRDIFCOL(f)  ( (f) << 8 )                /* Autre couleur */
!#define PTRDIFCOLC(f) ((((f)|0x80) << 8)  /* Autre couleur clignotante */
!
!#define ET  0              /* Combinaisons Event pour MouEventWait() */
!#define OU 1
!
!#define MOUINT(rin, rout) int86(0x33, &rin, &rout)
!#define MOUINTX(rin, rout, sr) int86x(0x33, &rin, &rout, &sr)
!
!/*-- Macros de conversion des coordonnées de la souris entre l'écran */
!/*-- virtuel de la souris et l'écran de texte */
!
!#define XTOCOL(x) ( (x) >> 3 )                    /* X divisé par 8 */
!#define YTOROW(y) ( (y) >> 3 )                 /* Ligne divisée par 8 */
!#define COLTOX(c) ( (c) << 3 )                      /* C fois 8 */
!#define ROWTOY(r) ( (r) << 3 )                    /* Ligne par 8 */
!
!/*-- Variables globales ------------------------------------------*/
!
!BYTE tline,                              /* Nombre de lignes de texte */
!     tcol,                            /* Nombre de colonnes de texte */
!     mavail = FALSE;                /* Est TRUE, si souris disponible */
!
!/*-- Masque pour le curseur standard de la souris ------------------*/
!
!PTRVIEW stdptr = MouPtrMask( PTRSAMECHAR, PTRINVCOL );
!
!BYTE    * bufz,          /* Ptr sur buffer pour identification de zone */
!        nmb_zones = 0;              /* Aucune zone définie jusqu'ici */
!
!ZONE * zone_act;              /* Pointeur sur vecteur de zone actuel */
!int    blen;                 /* Longueur du buffer de zone en octets */
!
!/*-- Variables définies lors de chaque appel du gestionnaire souris */
!
!BYTE mouzon = AUCUNE_ZONE,                /* Zone de souris actuelle */
!     moucol,                /* Colonne de la souris (écran de texte) */
!     mourow;                  /* Ligne de la souris (écran de texte) */
!int  mouevent = EV_LEFT_REL + EV_RIGHT_REL;   /* Masque d'événements */
!
!/*-- Variables qui ne sont définies par le gestionnaire de la    ---*/
!/*-- souris que lorsqu'intervient un événement attendu            ---*/
!
!BYTE ev_zon,                    /* Zone dans laquelle figure la souris */
!     ev_col,                          /* Colonne de la souris */
!     ev_row;                            /* Ligne de la souris */
!
!/*****************************************************************
!*  Fonction        : M o u D e f i n e P t r                     *
!**---------------------------------------------------------------**
!*  Fonction        : Définit les masques curseur et écran définissant *
!*                    l'apparence du curseur de la souris         *
!*  Paramètres en entrée : MASK = les deux masques bits combinés en une *
!*                         valeur 32 bits du type UNSIGNED LONG   *
!*  Valeur Return   : Aucune                                      *
!*  Infos           : - Les 16 bits de plus fort poids de MASK repré- *
!*                      sentent le masque écran, les 16 bits de plus *
!*                      faible poids le masque curseur            *
!*****************************************************************/
!
!#pragma check_stack(off)                  /* Pas de Stack Checking ici */
!
!void MouDefinePtr( PTRVIEW mask )
!{
! static PTRVIEW anciencurseur = (PTRVIEW) 0;  /* Dernière valeur MASK */
! union REGS regs; /* Registres du processeur pour l'appel d'interrup. */
!
! if ( anciencurseur != mask )      /* Modification au dernier appel ? */
!  {                                                        /* Oui */
!   regs.x.ax = 0x000a;          /* N° fct pour "Set text pointer type" */
!   regs.x.bx = 0;                              /* Fixer curseur logiciel */
!   regs.x.cx = mask;              /* Le mot faible est le masque AND */
!   regs.x.dx = mask >> 16;          /* Le mot fort est le masque XOR */
!   MOUINT(regs, regs);             /* Appeler le driver de la souris */
!   anciencurseur = mask;            /* Ranger le nouveau masque bits */
!  }
!}
!
!/*****************************************************************
!*  Fonction        : M o u E v e n t H a n d l e r               *
!**---------------------------------------------------------------**
!*  Fonction        : Est appelé par le driver de la souris à travers *
!*                    la routine assembleur AssHand dès qu'intervient *
!*                    un événement concernant la souris.          *
!*  Paramètres en entrée : EvFlags  = Masque Event de l'événement *
!*                         ButState = Etat des boutons de la souris *
!*                         X, Y     = Position actuelle du curseur de *
!*                                    la souris, déjà convertie vers le *
!*                                    système de coordonnées de texte *
!*  Valeur Return   : Aucune                                      *
!*  Infos           : - Cette fonction est seulement destinée à être *
!*                      appelée par le driver de la souris et ne doit *
!*                      pas être appelée par une autre fonction.  *
!*****************************************************************/
!
!void MouEventHandler( int EvFlags, int ButState, int x, int y )
!{
! #define LBITS ( EV_LEFT_PRESS | EV_LEFT_REL )
! #define RBITS ( EV_RIGHT_PRESS | EV_RIGHT_REL )
!
! unsigned nouzon;                          /* Numéro de la nouvelle zone */
!
! mouevent &= 1;                                    /* Masquer le bit 0 */
! mouevent |= ( EvFlags & 1 );  /* Copier le bit 0 à partir de EvFlags */
!
! if ( EvFlags & LBITS )/* Bouton gauche de la souris relâché/appuyé ? */
!  {                                                        /* Oui */
!   mouevent &= LBITS;                          /* Masquer état antérieur */
!   mouevent |= ( EvFlags & LBITS );             /* Incruster nouvel état */
!  }
!
! if ( EvFlags & RBITS ) /* Bouton droit de la souris relâché/appuyé ? */
!  {                                  /* Oui, masquer et incruster bits */
!   mouevent &= RBITS;                          /* Masquer état antérieur */
!   mouevent |= ( EvFlags & RBITS );             /* Incruster nouvel état */
!  }
!
! moucol = x;                    /* Convertir colonne et colonnes de texte */
! mourow = y;                      /* Convertir ligne en lignes de texte */
!
! /*-- Déterminer zone dans laquelle figure la souris et examiner   ---*/
! /*-- si cette zone s'est modifiée depuis le dernier appel. Dans   ---*/
! /*-- ce cas, l'apparence du curseur de la souris doit être        ---*/
! /*-- redéfinie.                                                   ---*/
!
! nouzon = *(bufz + mourow * tcol + moucol);          /* Retirer zone */
! if ( nouzon != mouzon )                          /* Nouvelle zone ? */
!  MouDefinePtr((nouzon==AUCUNE_ZONE) ? stdptr :
!                                    (zone_act+nouzon)->ptr_mask);
! mouzon = nouzon;       /* Ranger numéro de zone dans variable globale */
!}
!
!#pragma check_stack               /* Restaurer ancien état en ce qui */
!#pragma check_stack              /* concerne le Stack Checking       */
!
```

```
/*****************************************************************
*  Fonction        : MouIBuffFill                                *
******************************************************************
*  Fonction        : Stocke le code de zone pour une zone de l'écran *
*                    déterminée à l'intérieur de la mémoire écran *
*                    interne du module                            *
*  Paramètres en entrée : x1, y1 = coin supérieur gauche de zone écran *
*                         x2, y2 = coin inférieur droit de zone écran *
*                    CODE   = Code de zone                        *
*  Valeur Return   : Aucune                                       *
*  Infos           : Cette fonction ne doit être appelée qu'à     *
*                    l'intérieur de ce module.                    *
*****************************************************************/

static void MouIBuffFill( BYTE x1, BYTE y1,
                          BYTE x2, BYTE y2, BYTE code )
{
 register BYTE * lptr;              /* Pointeur sur la mémoire de zone */
 BYTE i, j;                                 /* Compteur de boucle */

 lptr = bufz + y1 * tcol + x1;      /* Pointeur sur la première ligne */

 /*-- Parcourir les différentes lignes ----------------------------*/
 for (j=x2 - x1 + 1 ; y1 <= y2; ++y1, lptr+=tcol )
  memset( lptr, code, j );                          /* Fixer code */
}

/*****************************************************************
*  Fonction        : MouDefZone                                  *
******************************************************************
*  Fonction        : Permet de définir différentes zones de l'écran *
*                    qui seront dotées de codes respectifs pour le *
*                    travail avec la souris.                      *
*  Params. en entrée: - NOMBRE = Nombre de zones de l'écran       *
*                     - PTR    = Pointeur sur vecteur avec les des- *
*                                cripteurs de zone du type ZONE    *
*  Valeur Return   : Aucune                                       *
*  Infos           : - Le code AUCUNE_ZONE est affecté aux zones de *
*                      l'écran restées libres.                    *
*                    - Lorsque la souris pénètre dans l'une des zones *
*                      de l'écran spécifiées, le gestionnaire de la *
*                      souris commute automatiquement sur l'apparence *
*                      du curseur de la souris définie par le     *
*                      descripteur de zone correspondant.         *
*                    - Comme seul le pointeur transmis est stocké, *
*                      mais que le vecteur transmis n'est pas copié *
*                      dans un buffer séparé, le contenu du vecteur *
*                      ne doit pas être modifié jusqu'au prochain *
*                      appel de cette fonction.                   *
*****************************************************************/

void MouDefZone( BYTE nombre, ZONE * ptr )
{
 register BYTE i,                           /* Compteur de boucle */
              zone;                         /* Zone de la souris */

 zone_act = ptr;                    /* Ranger pointeur sur vecteur */
 nmb_zones = nombre;                        /* et nombre de zones */
 memset( bufz, AUCUNE_ZONE, blen );
 for (i=0 ; i<nombre ; ++ptr )
  MouIBuffFill( ptr->x1, ptr->y1, ptr->x2, ptr->y2, i++);

 /*-- Redéfinir le curseur de la souris ---------------------------*/

 zone = *(bufz + mourow * tcol + moucol);       /*Zone souris actuelle*/
 MouDefinePtr( ( zone == AUCUNE_ZONE ) ? stdptr
               : (zone_act+zone)->ptr_mask );
}

/*****************************************************************
*  Fonction        : MouEventWait                                *
******************************************************************
*  Fonction        : Attend l'intervention d'un événement déterminé *
*                    sur le clavier.                              *
*  Params. en entrée: TYP        = Définit la combinaison entre les *
*                                  différents événements.       *
*                     WAIT_EVENT = Masque bits spécifiant l'événement *
*                                  attendu.                     *
*  Valeur Return   : Masque bits décrivant le ou les événements   *
*                    intervenu(s)                                 *
*  Infos           : - WAIT_EVENT peut être composé par combinaison *
*                      OU entre les diverses constantes, telles que *
*                      EV_MOU_MOVE ou EV_LEFT_PRESS par exemple   *
*                    - ET ou OU peuvent être spécifiés pour le TYP. *
*                      Avec ET, la fonction ne revient au programme *
*                      d'appel qu'une fois que tous les événements *
*                      attendus sont intervenus simultanément, alors *
*                      que OU se contente de l'intervention d'un  *
*                      événement au moins.                        *
*****************************************************************/

int MouEventWait( BYTE typ, int wait_event )
{
 int act_event;                             /* Le masque Event actuel */
 register BYTE colonne = moucol,            /* Dernière position souris */
              ligne    = mourow;
 BYTE fin = FALSE;                  /* Devient TRUE si événement intervenu */

 while ( !fin )         /* Répéter jusqu'à ce qu'événement intervienne */
  {
   /*-- Attendre que l'un des événements intervienne ----------------*/

   if ( typ == ET )   /* ET : tous les événements doivent se produire */
    while ( (act_event = mouevent) != wait_event)
     ;
   else                 /* OU : au moins un événement doit intervenir */
    while ( ( (act_event = mouevent) & wait_event) == 0)
     ;

   act_event &= wait_event;   /* Ne laisser que les bits d'événements */

       /*-- Lorsqu'on attend le déplacement de la souris, l'événement */
        /*-- n'est accepté que si le curseur de la sourîs a été amené */
         /*-- dans une autre ligne et/ou colonne de l'écran de texte */

   if ((wait_event & EV_MOU_MOVE) && colonne==moucol && ligne==mourow)
    {                           /* Souris déplacée mais même position écran */
     act_event &= (EV_MOU_MOVE);                    /* Masquer bit Move */
     fin = (act_event != 0);              /* Reste-t-il des événements ? */
    }
   else                                             /* Evénement intervenu */
    fin = TRUE;
  }
 ev_col = moucol;                           /* Conserver position et zone */
 ev_row = mourow;                            /* souris dans des variables */
 ev_zon = mouzon;                                        /* globales */
 return( act_event );                          /* Renvoyer masque Event */
}

/*****************************************************************
*  Fonction        : MouISetEventHandler                         *
******************************************************************
*  Fonction        : Installe un gestionnaire d'événements qui est *
*                    appelé par le driver de la souris lorsqu'un  *
*                    événement souris déterminé intervient.       *
*  Params en entrée : EVENT = Masque bits spécifiant l'événement  *
*                             dont l'intervention doit entraîner  *
*                             l'appel du gestionnaire de la souris. *
*                     PTR   = Pointeur sur gestionnaire de la souris *
*  Valeur Return   : Aucune                                       *
*  Infos           : - EVENT peut être composé par combinaison Ou *
*                      des différentes constantes telles que      *
*                      EV_MOU_MOVE ou EV_LEFT_PRESS du fichier    *
*****************************************************************/

static void MouISetEventHandler( unsigned event, MOUHAPTR ptr )
{
 union REGS regs;/* Registres du processeur pour l'appel d'interrupt. */
 struct SREGS sregs; /* Registre de segment pour l'appel d'interrupt. */

 regs.x.ax = 0x000C;                /* N° fct. pour "Set Mouse Handler" */
 regs.x.cx = event;                      /* Charger masque d'événements */
 regs.x.dx = FP_OFF( ptr );         /* Adresse d'offset du gestionnaire */
 sregs.es  = FP_SEG( ptr );        /* Adresse de segment du gestionnaire */
 MOUINTX( regs, regs, sregs );       /* Appeler le driver de la souris */
}

/*****************************************************************
*  Fonction        : MouIGetX                                    *
******************************************************************
*  Fonction        : Détermine la colonne (de texte) dans laquelle *
*                    figure le curseur de la souris.              *
*  Params en entrée : Aucun                                       *
*  Valeur Return   : La colonne de souris par rapport à écran texte *
*****************************************************************/

static BYTE MouIGetX( void )
{
 union REGS regs;/* Registres du processeur pour l'appel d'interrupt. */

 regs.x.ax= 0x0003;                 /* N° fct.: pour "Get mouse position" */
 MOUINT( regs, regs );                 /* Appeler le driver de la souris */
 return XTOCOL( regs.x.cx );            /* Convertir colonne et renvoyer */
}

/*****************************************************************
*  Fonction        : MouIGetY                                    *
******************************************************************
*  Fonction        : Détermine la ligne (de texte) dans laquelle  *
*                    figure le curseur de la souris.              *
*  Params en entrée : Aucun                                       *
*  Valeur Return   : La ligne de souris par rapport à écran de texte *
*****************************************************************/

static BYTE MouIGetY( void )
{
```

```
union REGS regs;/* Registres du processeur pour l'appel d'interrupt. */

regs.x.ax= 0x0003;              /* N° fct.: pour "Get mouse position" */
MOUINT(regs, regs);               /* Appeler le driver de la souris */
return YTOROW(regs.x.dx);         /* Convertir ligne et renvoyer */
}

/*****************************************************************
*  Fonction        : M o u S h o w M o u s e                     *
*----------------------------------------------------------------*
*  Fonction        : Afficher curseur souris sur l'écran.        *
*  Params en entrée : Aucun                                      *
*  Valeur Return   : Aucune                                      *
*  Infos           : Les appels de MouHideMouse() et MouShowMouse() *
*                    doivent s'équilibrer pour avoir un effet.   *
*****************************************************************/

void MouShowMouse( void )
{
union REGS regs;/* Registres du processeur pour l'appel d'interrupt. */

regs.x.ax = 0x0001;                   /* N° fct.: pour "Show Mouse" */
MOUINT(regs, regs);               /* Appeler le driver de la souris */
}

/*****************************************************************
*  Fonction        : M o u H i d e M o u s e                     *
*----------------------------------------------------------------*
*  Fonction        : Eliminer curseur de la souris de l'écran.   *
*  Params en entrée : Aucun                                      *
*  Valeur Return   : Aucune                                      *
*  Infos           : Les appels de MouHideMouse() et MouShowMouse() *
*                    doivent s'équilibrer pour avoir un effet.   *
*****************************************************************/

void MouHideMouse( void )
{
union REGS regs;/* Registres du processeur pour l'appel d'interrupt. */

regs.x.ax = 0x0002;                   /* N° fct, pour "Hide Mouse" */
MOUINT(regs, regs);               /* Appeler le driver de la souris */
}

/*****************************************************************
*  Fonction        : M o u S e t M o v e A r e a                 *
*----------------------------------------------------------------*
*  Fonction        : Définit la zone de l'écran à l'intérieur de *
*                    laquelle le curseur de la souris peut se déplacer*
*  Params en entrée : x1, y1 = coordonnées du coin supérieur gauche *
*                     x2, y2 = coordonnées du coin inférieur droit *
*  Valeur Return   : Aucune                                      *
*  Infos           : - Les deux paramètres se réfèrent à l'écran de *
*                      texte et non à l'écran graphique virtuel de *
*                      la souris                                 *
*****************************************************************/

void MouSetMoveArea( BYTE x1, BYTE y1, BYTE x2, BYTE y2 )
{
union REGS regs;/* Registres du processeur pour l'appel d'interrupt. */

regs.x.ax = 0x0008;           /* N° fct. pour "Set vertical limits" */
regs.x.cx = ROWTOY( y1 );             /* Conversion vers l'écran */
regs.x.dx = ROWTOY( y2 );                /* virtuel de la souris */
MOUINT(regs, regs);           /* Appeler le driver de la souris */
regs.x.ax = 0x0007;         /* N° fct. pour "Set horizontal limits" */
regs.x.cx = COLTOX( x1 );             /* Conversion vers l'écran */
regs.x.dx = COLTOX( x2 );                /* virtuel de la souris */
MOUINT(regs, regs);           /* Appeler le driver de la souris */
}

/*****************************************************************
*  Fonction        : M o u S e t S p e e d                       *
*----------------------------------------------------------------*
*  Fonction        : Fixe le rapport entre la longueur d'un déplace- *
*                    ment de la souris et le déplacement du curseur *
*                    de la souris qui doit en résulter.          *
*  Params en entrée : - XSPEED = vitesse horizontalement         *
*                     - YSPEED = vitesse verticalement           *
*  Valeur Return   : Aucune                                      *
*  Infos           : - Les deux paramètres sont exprimés en unités de *
*                      mickey / 8 points écran.                  *
*****************************************************************/

void MouSetSpeed( int xspeed, int yspeed )
{
union REGS regs;/* Registres du processeur pour l'appel d'interrupt. */

regs.x.ax = 0x000f;           /* N° fct. "Set mickeys to pixel ratio" */
regs.x.cx = xspeed;
regs.x.dx = yspeed;
MOUINT(regs, regs);               /* Appeler le driver de la souris */
}

/*****************************************************************
*  Fonction        : M o u M o v e P t r                         *
*----------------------------------------------------------------*
*  Fonction        : Amène le curseur de la souris dans une position *
*                    déterminée de l'écran.                      *
*  Params en entrée : - COL = nouvelle colonne de l'écran        *
*                     - ROW = nouvelle ligne de l'écran          *
*  Valeur Return   : Aucune                                      *
*  Infos           : - Les deux paramètres se réfèrent à l'écran de *
*                      texte et non à l'écran graphique virtuel de *
*                      la souris                                 *
*****************************************************************/

void MouMovePtr( int col, int row )
{
union REGS regs;/* Registres du processeur pour l'appel d'interrupt. */
unsigned nouzon;          /* Zone dans laquelle la souris est amenée */

regs.x.ax = 0x0004;           /* N° fct. "Set mouse pointer position" */
regs.x.cx = COLTOX( moucol = col );  /* Convertir coordonnées et les */
regs.x.dx = ROWTOY( mourow = row );/* stocker dans variables globales*/
MOUINT(regs, regs);               /* Appeler le driver de la souris */

nouzon = *(bufz + mourow * tcol + moucol);          /* Retirer zone */
if ( nouzon != mouzon )                          /* Nouvelle zone ? */
  MouDefinePtr((nouzon==AUCUNE_ZONE) ? stdptr :
                                  (zone_act+nouzon)->ptr_mask);
mouzon = nouzon;              /* Ranger n° de zone dans var. globale */
}

/*****************************************************************
*  Fonction        : M o u S e t D e f a u l t P t r             *
*----------------------------------------------------------------*
*  Fonction        : Définit l'apparence du curseur de la souris pour *
*                    les zones de l'écran qui n'ont pas été définies *
*                    avec MouDefZone.                            *
*  Params en entrée : STANDARD = Masque bits pour curseur souris stand.*
*  Valeur Return   : Aucune                                      *
*****************************************************************/

void MouSetDefaultPtr( PTRVIEW standard )
{
stdptr = standard;        /* Ranger masque bits dans variable globale */

/*-- Si la souris ne figure actuellement dans aucune zone, la    ---*/
/*-- nouvelle apparence du curseur est directement activée       ---*/

if ( MouGetZone() == AUCUNE_ZONE )            /* Dans aucune zone ? */
  MouDefinePtr( standard );                                 /* Non */
}

/*****************************************************************
*  Fonction        : M o u E n d                                 *
*----------------------------------------------------------------*
*  Fonction        : Met fin au travail avec les fonctions du module *
*                    Mousec.                                     *
*  Params en entrée : Aucun                                      *
*  Valeur Return   : Aucune                                      *
*  Infos           : Cette fonction est appelée automatiquement à la *
*                    fin d'un programme, à condition que MouInstall *
*                    ait été appelé auparavant.                  *
*****************************************************************/

void MouEnd( void )
{
union REGS regs;/* Registres du processeur pour l'appel d'interrupt. */

MouHideMouse();           /* Eliminer curseur de la souris de l'écran */
regs.x.ax = 0;             /* Réinitialisation du driver de la souris */
MOUINT(regs, regs);               /* Appeler le driver de la souris */

free( bufz );                     /* Libérer à nouveau mémoire allouée */
}

/*****************************************************************
*  Fonction        : M o u I n i t                               *
*----------------------------------------------------------------*
*  Fonction        : Dirige le travail avec le module Mousec et  *
*                    initialise les différentes variables        *
*  Params en entrée : Colonnes, = la résolution de l'écran de texte *
*                     Lignes                                     *
*  Valeur Return   : TRUE si une souris est installée, sinon FALSE *
*  Infos           : Cette fonction doit être la première fonction de *
*                    ce module à être appelée.                   *
*****************************************************************/

BYTE MouInit( BYTE colonnes, BYTE lignes )
{
union REGS regs;/* Registres du processeur pour l'appel d'interrupt. */

tline = lignes;                   /* Stocker nombre de lignes et de */
tcol  = colonnes;           /* colonnes dans des variables globales */
```

```
atexit( MouEnd );              /* Appeler MouEnd à la fin du programme */

/*-- Allouer et remplir buffer pour zones souris ---------------------*/

bufz = (BYTE *) malloc( bien = tline * tcol );
MouIBuffFill( 0, 0, tcol-1, tline-1, AUCUNE_ZONE );

regs.x.ax = 0;                      /* Initialiser driver de la souris */
MOUINT(regs, regs);                  /* Appeler le driver de la souris */
if ( regs.x.ax |= 0xffff )           /* Driver de la souris installé ? */
 return FALSE;                                          /* Non */

MouSetMoveAreaAll();                       /* Fixer zone de déplacement */

moucol = MouIGetX();                       /* Charger position actuelle */
mourow = MouIGetY();      /* de la souris dans des variables globales */

/*-- Installer gestionnaire d'événements assembleur "AssHand" -------*/
MouISetEventHandler( EV_MOU_ALL, (MOUHAPTR) AssHand );

return mevail = TRUE;                      /* La souris est installée */
}

/*****************************************************************
*                  P R O G R A M M E   P R I N C I P A L         *
*****************************************************************/

int main( void )
{
 static ZONE zones[] =          /* Les différentes zones de la souris */
  {
   ( 0,  0, 79,  0, MouPtrMask( PTRDIFCHAR(0x18), PTRINVCOL)  ),
   ( 0,  1,  0, 23, MouPtrMask( PTRDIFCHAR(0x1b), PTRINVCOL)  ),
   ( 0, 24, 78, 24, MouPtrMask( PTRDIFCHAR(0x19), PTRINVCOL)  ),
   (79,  1, 79, 23, MouPtrMask( PTRDIFCHAR(0x1a), PTRINVCOL)  ),
   (79, 24, 79, 24, MouPtrMask( PTRDIFCHAR('X'),  PTRDIFCOLC(0x40) ) ),
  };

 printf("\nSOURIS - (c) 1989 by MICHAEL TISCHER\n\n");
 if ( MouInit( 80, 25 ) )           /* Initialiser module de la souris */
  {                  /* Tout va bien, un driver de la souris est installé */
   printf("Si vous déplacez le curseur de la souris sur l'écran,\n"\
          "et notamment le long des bords de l'écran, vous\n"\
          "constatez que l'apparence du curseur de la souris se\n"\
          "modifie en fonction de sa position.\n\n"
          "Pour mettre fin à cette démo, amenez le curseur de la\n"\
          "souris dans le coin inférieur droit de l'écran puis\n"\
          "appuyez alors à la fois sur les boutons gauche et\n"\
          "droit de la souris.\n" );

   MouSetDefaultPtr( MouPtrMask( PTRDIFCHAR( '' ), PTRDIFCOL( 3 ) ) );
   MouDefZone( ELVEC( zones ), zones );                  /* Déf. zones */
   MouShowMouse();          /* Afficher curseur de la souris sur l'écran */

   /*-- Attendre que les boutons gauche et droit de la souris       --*/
   /*-- soient appuyés simultanément et que le curseur de la souris --*/
   /*-- figure à ce moment dans la zone 4                           --*/

   do                                                    /* Boucle de test */
    MouEventWait( ET, EV_LEFT_PRESS | EV_RIGHT_PRESS );
   while ( MouGetZone() != 4 );

   return 0;                                  /* Renvoyer code OK au DOS */
  }
 else              /* Pas de souris ou pas de driver souris installé */
  {
   printf("Aucun driver de la souris n'est installé !\n");
   return 1;                                  /* Renvoyer code d'erreur au DOS */
  }
}
```

Listing : SOURISCA.ASM

```
;*****************************************************************;
;*                          S O U R I S C A                      *;
;*-------------------------------------------------------------- *;
;*   Fonction       : Gestionnaire d'événements appelé par le driver *;
;*                    de la souris. A intégrer dans un programme C  *;
;*                    compilé avec le modèle de mémoire SMALL.      *;
;*-------------------------------------------------------------- *;
;*   Auteur         : MICHAEL TISCHER                             *;
;*   Développé le   : 20/04/1989                                  *;
;*   Dernière modif.: 20/04/1989                                  *;
;*-------------------------------------------------------------- *;
;*   Assemblage     : MASM SOURISCA;                              *;
;*                    ... combiner avec le programme C SOURISC.C  *;
;*****************************************************************;

;--- Déclarations de segment pour le programme C ------------------

IGROUP group _text              ;Regroupement des segments de programme
DGROUP group _bss, _data        ;Regroupement des segments de données
       assume CS:IGROUP, DS:DGROUP, ES:DGROUP, SS:DGROUP

_BSS   segment word public 'BSS'  ;Reçoit toutes les variables statiques
_BSS   ends                                          ;non initialisées

_DATA  segment word public 'DATA'       ;Variables globales et statiques
                                         ;initialisées sont logées dans
                                                          ;ce segment
_DATA  ends

;--- Programme ----------------------------------------------------

_TEXT  segment byte public 'CODE'                ;Le segment de programme

public     _AssHand                 ;Donne au programme C la possibilité de
                                     ;déterminer l'adresse du gestionnaire
                                                         ;en assembleur

extrn      _MouEventHandler : near    ;Driver d'événements en C à appeler

actif      db  0                  ;Indique si un appel est encore en cours
                                                         ;d'exécution

;-------------------------------------------------------------------
;-- _AssHand : le gestionnaire d'événements, qui est d'abord appelé par
;--            le driver de la souris, et appelle à son tour la
;--            fonction C MouEventHandler()
;-- Appel en C : interdit !

_AssHand   proc far

;-- Sauvegarder tout d'abord tous les registres du processeur sur la pile

           cmp  actif,0                    ;Appel non encore terminé ?
           jne  fin                      ;Non --> ne pas permettre l'appel

           mov  actif,1                    ;Ne plus autoriser d'appels

           push ax
           push bx
           push cx
           push dx
           push di
           push si
           push bp
           push es
           push ds

           ;-- Placer sur la pile les arguments pour l'appel de la -----
                                                  ;-- fonction FCT_C
               ;-- Appel : MouEventHandler( int EvFlags, int ButStatus,
                    ;--                    int x,       int y );

           mov  di,cx                ;Ranger ordonnée horizontale dans DI
           mov  cl,3    ;Compteur de décalages pour numéro de coordonnée
           shr  dx,cl               ;Diviser DX (ordonnée verticale) par 8
           push dx                            ;et placer sur la pile

           shr  di,cl             ;Diviser DI (ordonnée horizontale) par 8
           push di                            ;et placer sur la pile

           push bx              ;Etat des boutons de la souris sur la pile
           push ax                 ;Placer flags d'événements sur la pile

           mov  ax,DGROUP            ;Adresse de segment du DGROUP dans AX
           mov  ds,ax                      ;et de là dans le registre DS

           call _MouEventHandler                  ;Appel de la fonction C

           add  sp,8                        ;Eliminer arguments de la pile

           ;-- Retirer de la pile les registres sauvegardés ------------

           pop  ds
           pop  es
```

```
        pop  bp
        pop  si
        pop  di
        pop  dx
        pop  cx
        pop  bx
        pop  ax

        mov  actif,0                    ;Appel à nouveau autorisé

fin:    ret                             ;Retour au driver de la souris

_AssHand  endp

;-------------------------------------------------------------------------

_text   ends                            ;Fin du segment de code
        end                             ;Fin du programme
```

9.3. Souris - Communication PC

Etudions brièvement comment les mouvements physiques et les appuis sur les boutons de la souris sont traduites en informations compréhensibles par l'ordinateur.

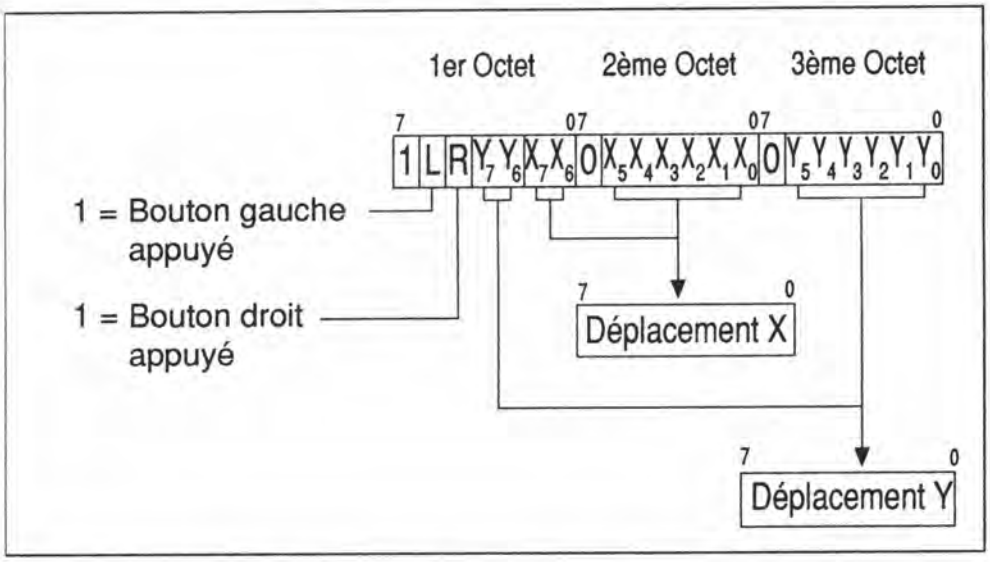
La méthode de transmission de données et la manière dont la souris est interfacée avec l'ordinateur dépendent du constructeur de la souris. Une souris série fonctionne différemment d'une souris bus. Etant donné que les souris série sont largement plus répandues que les souris bus, nous parlerons ici uniquement des premières.

Le driver de la souris détermine durant son initialisation comment la souris est connectée. Le driver de la souris Microsoft règle le signal DTR (Data Transfer Ready) du port série à 1. La souris reconnaît le signal et envoie le code ASCII de la lettre M sur la ligne de données. Le driver n'attend pas très longtemps le message de retour. C'est pourquoi une souris compatible Microsoft peut ne pas fonctionner avec le driver souris Microsoft. Même si la souris est réglée pour envoyer des données selon le protocole Microsoft, le driver peut ne pas reconnaître la souris si le "M" n'est pas envoyé rapidement.

Une fois que la connexion a été établie correctement, la souris utilise la méthode interrupt pour envoyer ses informations au driver. La souris exécute une interruption hardware sur le port série dès qu'elle doit informer le driver d'un mouvement ou du changement de l'état des boutons. Les interruptions utilisées sont 0CH (COM1) ou 0DH (COM2).

Le driver de la souris déroute les interruptions vers ses propres routines pour pouvoir lire directement les données sur le port série. La transmission de données pour une souris Microsoft s'effectue selon les normes suivantes : 1200 baud, 7 bits de données, pas de bit de parité. Chaque message de la souris se constitue de trois octets qui indiquent l'état des boutons de la souris et les mouvements éventuels effectués depuis la dernière transmission. Ces déplacements sont mesurés en mickeys. En fonction des résolutions de la carte vidéo et de la souris, un mickey correspond à 1/200 pouce ou 1/400 pouce.

Comme le montre la figure suivante, seuls 7 bits sont disponibles pour décrire le déplacement effectué sur les axes X et Y. Cela permet de couvrir des distances entre -128 et +127 mickeys. A la limite, la nouvelle position de la souris doit être envoyée quand la souris se déplace hors de ces valeurs. En général, un nouveau message sera envoyé plus souvent que cela.



Comme le montre la figure précédente, le format Microsoft se limite à deux boutons de souris. Le bit le plus fort est réservé pour la synchronisation avec le driver souris. Cela évite que le driver souris ne lise d'abord le second ou troisième octet du message lorsque l'interruption hardware échoue et que le premier octet ou les deux octets sont perdus. Le bit le plus fort du permier octet est toujours à 1 et les bits forts des deux autres octets sont toujours à 0.

D'autres constructeurs de souris qui utilisent trois boutons de souris exploitent un format de communication différent (huit bits de données au lieu de sept). Ce bit supplémentaire permet la transmission de l'état du troisième bouton. Les autres principes de la transmission restent identiques, la souris est toujours branchée sur un port série et l'interruption du port série communique avec le driver.

10. Joystick

Avec l'introduction de cartes graphiques de plus en plus performantes dans le monde du PC, les jeux ont également pris une importance considérable. Qui aime bien piloter son vaisseau spatial avec le clavier à travers les astéroïdes ou diriger sa voiture de course avec la souris ? Certainement personne ! C'est pourquoi, les joysticks ont fait leur apparition dans le monde du PC ces dernières années. Dans ce chapitre, vous :

- ✓ saurez comment les joysticks sont reliés au PC,
- ✓ apprendrez à intégrer les joysticks dans vos programmes, et
- ✓ ferez connaissance avec les services rendus par le BIOS.

Les trois programmes d'exemple en BASIC, Pascal et C fournis à la fin du chapitre vous aideront à intégrer très facilement les joysticks dans vos programmes.

Connexion de joysticks

Encore une fois, l'électronique du PC n'est pas en mesure de recevoir les joysticks et doit, par conséquent, attendre une carte capable d'assurer le pilotage du joystick. Généralement, il s'agit d'une petite carte d'extension implantée dans la place réservée à l'extension et appelée officiellement "Game Control Adapter". En pratique, on parle tout simplement de "carte joystick".

A l'arrière d'une telle carte, on trouve en principe deux prises pour connecter les deux joysticks. Il ne s'agit là que d'une option qui n'est pas toujours utilisée. En fait, un seul joystick suffit amplement.

Celui qui espérait réutiliser ses vieux joysticks adaptés au domaine des ordinateurs personnels (C 64, etc.) après l'achat de la carte joystick relativement peu coûteuse doit changer d'avis. En effet, le PC fonctionne essentiellement avec des joysticks analogiques alors que les ordinateurs personnels utilisent presque exclusivement des joysticks digitaux.

La différence résulte principalement dans le fait qu'un joystick analogique est capable de traiter une importante masse d'informations. Contrairement à son collègue digital, il peut non seulement afficher si le joystick est en train de se déplacer vers la gauche ou la droite, mais peut faire la différence entre des degrés d'intensité variés.

Les joysticks PC sont équipés de deux touches accessibles séparément et utilisables à loisir par le logiciel. Ainsi, il est rare de rencontrer des touches spéciales destinées à des fonctions telles que "Tir continu".

Contrôle à l'aide du BIOS

Généralement, un driver logiciel n'est pas nécessaire pour contrôler la présence des joysticks car l'interface matérielle, tout comme l'interface logicielle, sont déjà définies dans un PC Ur avec une configuration matériel. Du côté matériel, il s'agit des ports 200h à 20Fh qui sont réservés pour une carte joystick. Le BIOS accède à ces ports standardisés pour contrôler la présence des joysticks.

Du côté logiciel, deux fonctions appelées à l'aide de l'interruption 15h assurent cette tâche. Il s'agit en l'occurrence des sous-fonctions 00h et 01h de la fonction 84h de cette interruption. Elles exécutent un contrôle joystick selon le principe du polling, demandent la position actuelle du joystick et l'état des touches joystick lors de l'appel, c'est-à-dire directement à partir des ports cités.

Il n'est pas possible d'effectuer un contrôle des joysticks avec une méthode différente de celle du polling. Les cartes joystick n'étant pas affectées à une interruption matérielle spéciale, celle-ci ne peut par exemple pas être appelée lors d'un déplacement des joysticks ou l'appui sur l'une des deux touches. Par conséquent, un programme est obligé d'appeler les paramètres des deux fonctions BIOS s'il veut exécuter correctement les actions de l'utilisateur.

Déterminer la position des joysticks

La position des joysticks peut être obtenue à l'aide de la sous-fonction 01h. Lors de son appel, le numéro de fonction 84h et le numéro de la sous-fonction 01h doivent se trouver respectivement dans les registres AH et DX. Après l'appel de l'interruption 15h, si le Flag Carry contient la valeur 0, cela signifie que la fonction est reconnue par le BIOS concerné. Les informations souhaitées peuvent donc être obtenues à partir des registres AX à DX.

Chacun des deux joysticks connectables sont représentés par deux registres reproduisant la position des joysticks par rapport aux axes X et Y. Pour le premier joystick, il s'agit des registres AX (axe X) et BX (axe Y), pour le second, des registres CX (axe X) et DX (axe Y).

Le contenu de ces registres permet également de savoir si un joystick est effectivement relié ou non, car dans ce cas, une valeur différente de 0 est retournée dans les deux registres correspondants. Si on rencontre par exemple la valeur 0 dans les registres CX et DX après appel de la fonction, on a alors la certitude que le second joystick n'est pas connecté. La même considération s'applique naturellement au premier joystick dont les registres sont AX et BX.

Des valeurs différentes de 0 représentent la position du joystick par rapport aux axes correspondants. Ces valeurs ne sont d'ailleurs pas normées de quelque manière que ce

soit. De plus, les valeurs retournées dépendent des potentiomètres situés à l'intérieur du joystick pour convertir sa position en une dimension électrique. Tant les potentiomètres intégrés dans les joysticks des divers constructeurs que les innombrables joysticks d'un seul constructeur retournent des valeurs complètement différentes les unes des autres.

La plupart des programmes devant effectuer un contrôle joystick permettent à l'utilisateur de déplacer le joystick dans la zone de commutation située en haut à droite et en bas à gauche. Cela permet de connaître l'intervalle dans lequel se déplacent les valeurs du potientiomètre qui ont été retournées. Avec mes joysticks par exemple, l'intervalle des valeurs se situe entre 10 à 120 par rapport à l'axe X et 9 à 102 par rapport à l'axe Y. Les valeurs retournées par d'autres joysticks seront naturellement tout à fait différentes. Il faut faire attention au fait que les valeurs augmentent de gauche à droite par rapport à l'axe X alors que sur l'axe Y, les valeurs croissent du haut vers le bas et non dans le sens inverse comme on pourrait le supposer.

Vous constatez qu'il n'est pas évident de déterminer l'emplacement du joystick sur un système de coordonnées linéaire. Etant donné que les potentiomètres ne fonctionnent pas toujours en mode linéaire dans tous les joysticks mais plutôt en mode exponentiel, il en résulte que la position fondamentale du joystick ne correspond nullement à une valeur moyenne par rapport à l'intervalle des valeurs. Les programmes fournis à la fin du chapitre illustrent d'ailleurs cette thématique. Vous pouvez les utiliser pour tester vos joysticks selon ce point de vue.

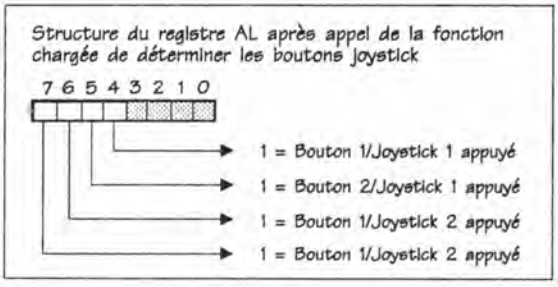
Pour éviter tout problème résultant de l'unité de mesure exponentielle des potentiomètres, de nombreux programmes renoncent à représenter les coordonnées joystick retournées dans un système de coordonnées linéaire. Généralement, cela s'avère tout à fait superflu comme par exemple dans le cas d'un fonctionnement du type PacMan où il s'agit surtout de déplacer l'objet écran (auto, PacMan, vaisseau spatial, etc.) vers le haut, le bas, la gauche ou la droite. A cet effet, il suffit de mesurer la position en cours du joystick au début du programme en considérant qu'il s'agit là de la position fondamentale. A l'intérieur du programme, les valeurs joystick obtenues doivent tout simplement être comparées avec la position fondamentale pour constater si le joystick a été déplacé vers la gauche, la droite ou ailleurs.

Déterminer les boutons joystick

Outre l'emplacement du joystick, les deux boutons joystick jouent également un rôle important dans le fonctionnement du programme. Elles peuvent être déterminées à l'aide de la sous-fonction 00h qui retourne la valeur 84h dans le registre AH et la valeur 00h dans le registre DX. On retrouve ici le Flag Carry qui donne des informations sur le bon déroulement de la fonction. S'il est effacé après l'appel de l'interruption 15h, les divers bits du registre AL représentent simultanément l'état des quatre boutons figurant sur les deux joysticks. Si l'utilisateur vient d'appuyer sur un bouton, le bit correspondant contient alors la valeur 1, sinon on rencontre la valeur 0.

Dans la mesure où les boutons sont rangés les unes sur les autres, le bouton du haut est considéré comme le premier bouton et celui du bas comme le second. Lorsque les boutons sont placés côte à côte, le bouton de gauche représente le premier bouton, celui de droite le second.

Voici la configuration du registre AL :



Lorsque vous utilisez cette fonction, rappelez-vous qu'elle ne possède pas de fonction buffer et retourne uniquement l'état actuel des boutons joystick. Si l'utilisateur appuie rapidement sur un bouton alors qu'un contrôle joystick n'a pas encore eu lieu, cette action reste caduque et l'utilisateur se demandera sûrement pourquoi son programme ne réagit pas. Il est donc important de contrôler en permanence l'état des boutons joystick à l'intérieur de vos programmes à l'aide de la sous-fonction 00h.

Programmes d'exemple

Les trois programmes en BASIC, Pascal et C illustrant ce chapitre contrôlent l'emplacement du joystick et l'état des boutons joystick à l'aide des fonctions décrites plus haut. Ces contrôles s'effectuent à l'aide de deux sous-programmes portant les noms GetJoyPos et GetJoyButton. Ces fonctions sont utilisées à l'intérieur du programme principal pour obtenir d'abord l'emplacement maximal du joystick après avoir demandé à l'utilisateur de placer le joystick dans la zone supérieure droite et d'appuyer ensuite sur l'une des deux touches.

Ici, le programme décide également de l'ordre dans lequel il va solliciter les deux joysticks. Grâce au bouton joystick appuyé par l'utilisateur, il est également possible de connaître le joystick installé ainsi que celui sélectionné par l'utilisateur.

L'utilisateur est ensuite invité à déplacer le joystick dans la partie inférieure gauche, ce qui permet de connaître également l'extrémité inférieure de l'intervalle des valeurs. Le programme configure par la suite cet intervalle sur les 80 colonnes et 25 lignes de l'écran texte et calcule la position actuelle du joystick par rapport à une position correspondante sur l'écran où apparaît un grand X. En outre, les valeurs potentiomètre retournées apparaissent dans le coin supérieur gauche de l'écran pour que l'utilisateur puisse traiter une image selon le fonctionnement de son joystick.

Le programme se termine dès que l'utilisateur appuie simultanément sur les deux boutons de son joystick.

Listing : JOYSTB.BAS

```
'*****************************************************************
'*                          J O Y S T B                          *
'*---------------------------------------------------------------*
'*  Fonction       : Démontre l'utilisation des Joysticks        *
'*                   à l'aide du BIOS                            *
'*  Auteur         : MICHAEL TISCHER                             *
'*  Développé le   : 25.02.1991                                  *
'*  Dernière modif : 26.02.1991                                  *
'*****************************************************************
DECLARE SUB GetJoyButton (j1b1%, j1b2%, j2b1%, j2b2%)
DECLARE SUB GetJoyPos (js1 AS ANY, js2 AS ANY)

REM $INCLUDE: 'qb.bi'

'--Déclaration de type --

TYPE JSPOS 'Décrit la position d'un Joystick
  x AS INTEGER
  y AS INTEGER
END TYPE

DIM jsp(1 TO 2) AS JSPOS                'Position actuelle du Joystick
DIM maxx AS INTEGER, maxy AS INTEGER    'Position maximale du Joystick
DIM minx AS INTEGER, miny AS INTEGER    'Position minimale du Joystick
DIM xold AS INTEGER, yold AS INTEGER    'Position ancienne à l'écran
DIM actstick AS INTEGER                 'Activer Joystick (1 ou 2)
DIM xfacteur AS SINGLE, yfacteur AS SINGLE  'facteurs de conversion X/Y
DIM j1but(1 TO 2) AS INTEGER            'Bouton 1 du Joystick 1 et 2
DIM j2but(1 TO 2) AS INTEGER            'Bouton 2 du Joystick 1 et 2

CLS
PRINT "Veuillez placer le joystick dans le coin supérieur droit "
PRINT "et appuyez sur l'un des deux boutons."

DO                                      'Attendre l'appui sur le bouton du Joystick
  CALL GetJoyButton(j1but(1), j2but(1), j1but(2), j2but(2))
LOOP WHILE (j1but(1) OR j2but(1) OR j1but(2) OR j2but(2)) = 0

IF j1but(1) OR j2but(1) <> 0 THEN               'Sélectionner Joystick
  actstick = 1
ELSE
  actstick = 2
END IF

CALL GetJoyPos(jsp(1), jsp(2))                  'Lire Position
maxx = jsp(actstick).x 'Position maximale
miny = jsp(actstick).y

DO                                      'Attendre à nouveau l'appui sur un bouton
  CALL GetJoyButton(j1but(1), j2but(1), j1but(2), j2but(2))
LOOP UNTIL (j1but(actstick) = 0) AND (j2but(actstick) = 0)

'--Lire maintenant la position minimale ---

PRINT : PRINT
PRINT "Placez le Joystick dans le coin inférieur gauche"
PRINT "et appuyez sur un bouton."

DO
  CALL GetJoyButton(j1but(1), j2but(1), j1but(2), j2but(2))
LOOP WHILE (j1but(actstick) = 0) AND (j2but(actstick) = 0)

CALL GetJoyPos(jsp(1), jsp(2))              'Lire la position actuelle
minx = jsp(actstick).x                      'et ranger la position minimale
maxy = jsp(actstick).y

facteurx = 80 / (maxx - minx + 1)   'Calculer les facteurs de conversion
facteury = 23 / (maxy - miny + 1)       'pour l'axe des X et pour l'axe Y

'-- Déterminer le Joystick et afficher sa position --
'-- jusqu'à ce que les deux boutons soient appuyés --

CLS
LOCATE 2, 44
PRINT "JOYSTB  - (c) 1991 MICHAEL TISCHER";
LOCATE 25, 1
PRINT "Appuyez sur les deux boutons  ";
PRINT "pour quitter le programme"
xold = 1                                    'Prédéfinir l'ancienne Position
yold = 1

DO
  '-- Lire la position --
  CALL GetJoyPos(jsp(1), jsp(2))
  LOCATE 1, 1
  PRINT "("; jsp(actstick).x; "/"; jsp(actstick).y; ")  ";

  '-- Calculer la nouvelle position X du Joystick ----
  x% = facteurx * (jsp(actstick).x - minx% + 1)
  IF x% < 1 THEN x% = 1
  IF x% > 80 THEN x% = 80

  '-- Calculer la nouvelle position Y du Joystick ----
  y% = facteury * (jsp(actstick).y - miny% + 1)
  IF y% < 1 THEN y% = 1
  IF y% > 23 THEN y% = 23

  '-- Afficher la nouvelle position si celle-ci a été modifiée ---
  IF (x% <> xold) OR (y% <> yold) THEN
    LOCATE yold + 1, xold
    PRINT " ";
    LOCATE y% + 1, x%
    PRINT "X";
    xold = x%
    yold = y%
  END IF

  CALL GetJoyButton(j1but(1), j2but(1), j1but(2), j2but(2))
LOOP UNTIL (j1but(actstick) = 1) AND (j2but(actstick) = 1)

CLS

'*****************************************************************
'* GetJoyButton: Renvoie l'emplacement des boutons de joystick   *
'*---------------------------------------------------------------*
'* Entrée : J1B1% = 1, si le bouton 1 du joystick 1 est appuyé,  *
'*                  sinon 0                                      *
'*          J1B2% = comme J1B1%, mais pour le 2nd bouton du joystick 1*
'*                                                               *
'*          J2B1% = comme J1B1%, mais pour le bouton 1 du joystick 2  *
'*          J2B2% = comme J1B2%, mais pour le bouton 2 du joystick 2  *
'*****************************************************************
'
SUB GetJoyButton (j1b1%, j1b2%, j2b1%, j2b2%)
  DIM regs AS RegType 'Registres processeur pour appel d'interruption

  regs.ax = &HB400                              'Fonction BIOS B4h
  regs.dx = &HD                                 'Sous-fonction 00h
  CALL INTERRUPT(&H15, regs, regs)
  j1b1% = (regs.ax AND 16) \ 16 XOR 1   'Bit 4 de AX = J1B1
  j1b2% = (regs.ax AND 32) \ 32 XOR 1   'Bit 5 de AX = J1B2
  j2b1% = (regs.ax AND 64) \ 64 XOR 1   'Bit 6 de AX = J2B1
  j2b2% = (regs.ax AND 128) \ 128 XOR 1 'Bit 7 de AX = J2B2
END SUB

'*****************************************************************
'* GetJoyPos : Retourne la position des deux Joysticks           *
'*---------------------------------------------------------------*
'* Paramètre en entrée: JS1 = Structure pour le 1er Joystick     *
'*                      JS2 = Structure pour le second Joystick  *
'*****************************************************************
'
SUB GetJoyPos (js1 AS JSPOS, js2 AS JSPOS)
  DIM regs AS RegType               'Registres pour appel d'interruption

  regs.ax = &HB400                                  'Fonction B4h
  regs.dx = &H1                                     'Sous-fonction 01h
  CALL INTERRUPT(&H15, regs, regs)
  js1.x = regs.ax                                   'Position du Joystick
  js1.y = regs.bx
  js2.x = regs.cx
  js2.y = regs.dx
END SUB
```

Listing : JOYSTP.PAS

```
{*****************************************************************}
{                          J O Y S T P                            }
{-----------------------------------------------------------------}
{   Fonction        : Démontre l'utilisation des Joysticks        }
{                     à l'aide du BIOS                            }
{-----------------------------------------------------------------}
{   Auteur          : MICHAEL TISCHER                             }
{   Développé le    : 25.02.1991                                  }
{   Dernière modif  : 25.02.1991                                  }
{*****************************************************************}

program JOYSTP;

uses DOS, CRT;

{== Déclaration de type ==========================================}

type JSPOS = record                    { Décrit la position d'un Joystick }
               x,
               y : integer;
             end;

{== Constantes ===================================================}

const CRLF = #13#10;

{== Variables globales ===========================================}

var jsp        : array [1..2] of JSPOS; { Position actuelle du Joystick}
    maxx, maxy,                         { Position maximale du Joystick}
    minx, miny,                         { Position minimale du Joystick}
    x, y,                               { Position actuelle à l'écran}
    xold, yold : integer;               { Position ancienne à l'écran }
    actstick   : byte;                        { Joystick sélectionné}
    j1but,                              { Bouton 1 du Joystick 1 et 2 }
    j2but      : array[1..2] of byte;   { Bouton 2 du Joystick 1 et 2 }
    xfacteur,                           { facteurs de conversion X et Y }
    yfacteur   : real;

{*****************************************************************
*  GetJoyButton:  Renvoie l'emplacement des boutons de Joystick  *
******************************************************************
*  Paramètres entrée: J1B1 = Variable pour Bouton 1 / Joystick 1 *
*                     J1B2 = Variable pour Bouton 2 / Joystick 1 *
*                     J2B1 = Variable pour Bouton 1 / Joystick 2 *
*                     J2B2 = Variable pour Bouton 2 / Joystick 2 *
*  Info             : Les diverses variables reçoivent la valeur 1*
*                     (TRUE) lorsque le bouton correspondant du   *
*                     joystick est appuyé, sinon 0 (FALSE).       *
*****************************************************************}

procedure GetJoyButton( var j1b1, j1b2, j2b1, j2b2 : byte );

var Regs : Registers;          { Registres processeur pour appel d'int. }

begin
  Regs.ah := $84;                                    { Fonction 84h }
  Regs.dx := 0;                                      { Sous-fonction 00h }
  intr( $15, Regs );                                 { Appel interrupt 15h }
  j1b1 := (( Regs.al and  16 ) shr 4) xor 1;         { Bit 4 de AL = J1B1 }
  j1b2 := (( Regs.al and  32 ) shr 5) xor 1;         { Bit 5 de AL = J1B2 }
  j2b1 := (( Regs.al and  64 ) shr 6) xor 1;         { Bit 6 de AL = J2B1 }
  j2b2 := (( regs.al and 128 ) shr 7) xor 1;         { Bit 7 de AL = J2B2 }
end;

{*****************************************************************
*  GetJoyPos : Retourne la position des deux Joysticks           *
******************************************************************
*  Paramètre en entrée: JS1 = Structure Joystick pour le 1er Joystick *
*                       JS2 = Structure Joystick pour le 2eme Joystick *
*****************************************************************}

procedure GetJoyPos( var Js1, Js2 : JSPOS );

var Regs : Registers;          { Registres pour appel d'interruption }

begin
  Regs.ah := $84;                                    { Fonction 84h }
  Regs.dx := 1;                                      { Sous-fonction 01h }
  intr( $15, Regs );                                 { Appel interrupt 15h }
  Js1.x := Regs.ax;                                  { Position X Joystick 1 }
  Js1.y := Regs.bx;                                  { Position Y Joystick 1 }
  Js2.x := Regs.cx;                                  { Position X Joystick 2 }
  Js2.y := Regs.dx;                                  { Position Y Joystick 2 }
end;

{*****************************************************************}

{*                        PROGRAMME PRINCIPAL                    *}
{*****************************************************************}

begin
                    {-- Lire d'abord la position maximale du Joystick--}

  ClrScr;
  writeln( 'Veuillez placer le joystick dans le coin supérieur droit ' +
           + CRLF + 'et appuyez sur l''un des deux boutons' );

  repeat                       { Attendre l'appui sur le bouton du Joystick }
    GetJoyButton( j1but[1], j2but[1], j1but[2], j2but[2] );
  until ( ( j1but[1] or j2but[1] or j1but[2] or j2but[2] ) <> 0 );

  if ( j1but[1] or j2but[1] ) <> 0 then          { Sélectionner Joystick }
    actstick := 1
  else
    actstick := 2;

  GetJoyPos( jsp[1], jsp[2] );                          { Lire Position     }
  maxx := jsp[actstick].x;                              { et ranger         }
  miny := jsp[actstick].y;

  repeat                             { Attendre que le bouton soit relâché }
    GetJoyButton( j1but[1], j2but[1], j1but[2], j2but[2] );
  until ( ( j1but[actstick] or j2but[actstick] ) = 0 );

  {-- Lire maintenant la position minimale ----------------------------}

  writeln( CRLF + CRLF + 'Placez le Joystick dans le coin'+
           ' inférieur gauche' + CRLF +
           'et appuyez sur un bouton' );

  repeat                         { Attendre à nouveau l''appui sur un bouton }
    GetJoyButton( j1but[1], j2but[1], j1but[2], j2but[2] );
  until ( ( j1but[actstick] or j2but[actstick] ) <> 0 );

  GetJoyPos( jsp[1], jsp[2] );                          { Lire la position  }
  minx := jsp[actstick].x;                              { et ranger         }
  maxy := jsp[actstick].y;

  xfacteur := 80.0 / ( maxx - minx + 1 );    { Calculer les facteurs de }
  yfacteur := 23.0 / ( maxy - miny + 1 );{ conversion pour les axes X/Y }

              {-- Déterminer le Joystick et afficher sa position ------}
              {-- jusqu'à ce que les deux boutons soient appuyés ------}

  ClrScr;
  GotoXY( 45, 1 );
  write( 'JOYSTP - (c) 1991 MICHAEL TISCHER' );
  GotoXY( 1, 25 );
  write( 'Appuyez sur les deux boutons pour quitter le programme');

  xold := 0;                                 { Prédéfinir l'ancienne Position }
  yold := 0;

  repeat
    GetJoyPos( jsp[1], jsp[2] );                        { Lire la position }

    {-- Calculer la nouvelle position X du Joystick ------------------}

    x := round(xfacteur * ( jsp[actstick].x - minx + 1 ));
    if ( x < 0 ) then
      x := 0;
    if ( x > 79 ) then
      x := 79;

    {-- Calculer la nouvelle position Y du Joystick ------------------}

    y := round(yfacteur * ( jsp[actstick].y - miny + 1 ));
    if ( y < 0 ) then
      y := 0;
    if ( y > 22 ) then
      y := 22;

    {-- Afficher la nouvelle position si celle-ci a été modifiée ------}

    if ( x <> xold ) or ( y <> yold ) then
      begin
        GotoXY( xold+1, yold+2 );
        write( ' ' );
        GotoXY( x+1, y+2 );
        write( 'X' );
        xold := x;
        yold := y;
      end;
  GotoXY( 1, 1 );
```

```
!   write( '(', jsp[actstick].x:3, '/', jsp[actstick].y:3, ')' );
!   GetJoyButton( j1but[1], j2but[1], j1but[2], j2but[2] );
! until ( j1but[actstick] = 1 ) and ( j2but[actstick] = 1 );
! ClrScr;

! GotoXY( 1, 1 );
! writeln( 'Fin de programme' );
!end.
```

Listing : JOYSTC.C

```
!/*****************************************************************/
!/*                          J O Y S T C                          */
!/*---------------------------------------------------------------*/
!/*  Fonction       : Démontre l'utilisation des Joysticks        */
!/*                   à l'aide du BIOS                            */
!/*---------------------------------------------------------------*/
!/*  Auteur         : MICHAEL TISCHER                             */
!/*  Développé le   : 25.02.1991                                  */
!/*  Dernière modif : 25.02.1991                                  */
!/*    (MICROSOFT C)                                              */
!/*    Compilation  : CL /AS JOYSTC.C                             */
!/*    Appel        : JOYSTC                                      */
!/*---------------------------------------------------------------*/
!/*    (BORLAND TURBO C)                                          */
!/*    Compilation  : Avec les commandes COMPILE / MAKE           */
!/*****************************************************************/

!/*-- Fichiers Include ---------------------------------------------*/

!#include <dos.h>
!#include <stdio.h>
!#include <stdarg.h>

!/*-- Déclaration de types -----------------------------------------*/

!typedef unsigned char BYTE;
!typedef struct {                   /* Décrit la position d'un Joystick */
!                int x;
!                int y;
!              } JSPOS;

!/*****************************************************************
!*  ClrScr: Effacement de l'écran                                *
!*---------------------------------------------------------------*
!*  Paramètre d'entrée: COULEUR  = Attribut du caractère         *
!*  Valeur de retour  : aucun                                    *
!*****************************************************************/

!void ClrScr( BYTE COULEUR )
!{
! union REGS regs;                  /* regs. pour appel des interrupts */

!   /*--Effacer l'écran à l'aide de la fonction Scroll-BIOS ----------*/

! regs.h.ah = 6;                  /* Numéro de fonction pour Scroll-Down */
! regs.h.al = 0;                   /* Scroll de 0 ligne ( = effacement ) */
! regs.h.bh = COULEUR;                        /* COULEUR du caractère */
! regs.x.cx = 0;                  /* coin supérieur gauche de la fenêtre */
! regs.x.dx = ( 24 << 8 ) + 79;                /* coin inférieur droit */
! int86(0x10, &regs, &regs);          /* Appel de l'interrupt BIOS Video */

!/*-- Placer le curseur dans le coin supérieur gauche à l'aide du BIOS-*/

! regs.h.ah = 2;                  /* Numéro de fonction pour Set Cursor */
! regs.h.bh = 0;                             /* Accéder à la page 0 */
! regs.x.dx = 0;                        /* Coin sup. gauche de l'écran */
! int86(0x10, &regs, &regs);                 /* Appel int. vidéo BIOS */
!}

!/*****************************************************************
!*  printfat : Affiche une chaîne formatée en un endroit quelconque *
!*             de l'écran                                        *
!*---------------------------------------------------------------*
!*  Paramètre d'entrée: COLONNE = Position de sortie             *
!*                      LIGNE                                    *
!*                      STRING = Pointeur sur la chaîne          *
!*                      ...    = Argument identique à  PRINTF()  *
!*  Valeur de retour  : aucune                                   *
!*  Info              : Cette fonction ne peut être appelée que si *
!*                      l'existence d'une carte EGA- ou VGA -    *
!*                      a été confirmée                          *
!*****************************************************************/

!void printfat( BYTE COLONNE, BYTE LIGNE, char * string, ... )
!{
! va_list parameter;          /* Liste de paramètres pour macros VA_... */
! union REGS regs;               /* registres pour appel d'interruption */

!/*-- Placer le curseur sur la position de sortie à l'aide du BIOS ----*/

! regs.h.ah = 2;                  /* Numéro de fonction pour Set Cursor */
! regs.h.bh = 0;                             /* Accéder à la page 0 */
! regs.h.dh = LIGNE;                              /* ranger LIGNE */
! regs.h.dl = COLONNE;                          /* ranger COLONNE */
! int86(0x10, &regs, &regs);          /* Appel d'int. vidéo du BIOS */

! /*-- Sortie de la chaîne ----------------------------------------*/

! va_start( parameter, string );
! vprintf( string, parameter );
!}

!/*****************************************************************
!*  Fonction        : G E T J O Y B U T T O N                    *
!*---------------------------------------------------------------*
!*  Fonction        : Retourne l'emplacement des boutons de Joystick *
!*  Paramètre d'entrée: J1B1 = Ptr sur var. pour Bouton 1 / Joystick 1 *
!*                      J1B2 = Ptr sur var. pour Bouton 2 / Joystick 1 *
!*                      J2B1 = Ptr sur var. pour Bouton 1 / Joystick 2 *
!*                      J2B2 = Ptr sur var. pour Bouton 2 / Joystick 2 *
!*  Valeur de retour    : aucune                                 *
!*  Info            : Les diverses variables reçoivent la val. 1 (TRUE)* 
!*                    lorsque le bouton correspondant du joystick *
!*                    est appuyé, sinon 0 (FALSE).               *
!*****************************************************************/

!void GetJoyBouton( BYTE *j1b1, BYTE *j1b2, BYTE *j2b1, BYTE *j2b2 )
!{
! union REGS regs;                              /* pour les registres */

! regs.h.ah = 0x84;                                 /* Fonction 84h */
! regs.x.dx = 0;                              /* Sous-fonction 00h */
! int86( 0x15, &regs, &regs );                    /* Appel Int 15h */
! *j1b1 = (( regs.h.al &  16 ) >> 4) ^ 1;     /* Bit 4 de AL = J1B1 */
! *j1b2 = (( regs.h.al &  32 ) >> 5) ^ 1;     /* Bit 5 de AL = J1B2 */
! *j2b1 = (( regs.h.al &  64 ) >> 6) ^ 1;     /* Bit 6 de AL = J2B1 */
! *j2b2 = (( regs.h.al & 128 ) >> 7) ^ 1;     /* Bit 7 de AL = J2B2 */
!}

!/*****************************************************************
!*  Fonction        : G E T J O Y P O S                          *
!*---------------------------------------------------------------*
!*  Fonction        : Retourne la position des deux Joysticks    *
!*  Paramètre d'entrée: JS1PTR =  Structure pour le 1er Joystick *
!*                      JS2PTR = Structure pour le second Joystick *
!*  Valeur de retour  : aucune                                   *
!*****************************************************************/

!void GetJoyPos( JSPOS *Js1Ptr, JSPOS *Js2Ptr )
!{
! union REGS regs;                              /* pour les registres */

! regs.h.ah = 0x84;                                 /* Fonction 84h */
! regs.x.dx = 1;                              /* Sous-fonction 01h */
! int86( 0x15, &regs, &regs );                    /* Appel Int 15h */
! Js1Ptr->x = regs.x.ax;                    /* Position X Joystick 1 */
! Js1Ptr->y = regs.x.bx;                    /* Position Y Joystick 1 */
! Js2Ptr->x = regs.x.cx;                    /* Position X Joystick 2 */
! Js2Ptr->y = regs.x.dx;                    /* Position Y Joystick 2 */
!}

!/*****************************************************************/
!/**                        PROGRAMME PRINCIPAL                  **/
!/*****************************************************************/

!void main()
!{
! JSPOS jsp[2];                       /*  Position Joystick actuelle*/
! int   maxx, maxy,                   /* Position Joystick maximale */
!       minx, miny,                   /* Position Joystick minimale */
!       x, y,                         /* Position écran actuelle*/
!       xold, yold;                   /* Dernière position d'écran*/
! BYTE  actstick,                     /* Joystick sélectionné*/
!       j1but[2],                     /* Bouton 1 du Joystick 1 et 2 */
!       j2but[2];                     /* Bouton 2 du Joystick 1 et 2 */
! float xfacteur, yfacteur;           /* facteurs de conversion X et Y */
```

```
/*-- Lire d'abord la position maximale du Joystick ------------------*/

ClrScr( 0x07 );
printf( "Veuillez placer le joystick dans le coin supérieur droit\n"
        "et appuyez sur l'un des deux boutons");

do                          /* Attendre l'appui sur le bouton du Joystick */
 GetJoyBouton( &j1but[0], &j2but[0], &j1but[1], &j2but[1] );
while ( ( j1but[0] | j2but[0] | j1but[1] | j2but[1] ) == 0 );

actstick = ( j1but[0] | j2but[0] ) ? 0 : 1; /* Sélectionner Joystick */

GetJoyPos( &jsp[0], &jsp[1] );                  /* Lire la position */
maxx = jsp[actstick].x;                         /* et ranger        */
miny = jsp[actstick].y;

do                              /* Attendre que le bouton soit relâché */
 GetJoyBouton( &j1but[0], &j2but[0], &j1but[1], &j2but[1] );
while ( ( j1but[actstick] | j2but[actstick] ) != 0 );

 /*-- Lire maintenant la position minimale ---------------------------*/

printf( "\n\nPlacez le Joystick dans le coin inférieur gauche\n"\
        "et appuyez sur l'un des deux boutons\n" );

do                          /*  Attendre à nouveau l'appui sur un bouton */
 GetJoyBouton( &j1but[0], &j2but[0], &j1but[1], &j2but[1] );
while ( ( j1but[actstick] | j2but[actstick] ) == 0 );

GetJoyPos( &jsp[0], &jsp[1] );                  /* Lire la position */
minx = jsp[actstick].x;                         /* et ranger        */
maxy = jsp[actstick].y;

xfacteur = 80.0 / ( maxx - minx + 1 );  /* Calculer les facteurs de  */
yfacteur = 23.0 / ( maxy - miny + 1 );/* conversion pour axes X et Y */

            /*-- Déterminer le Joystick et afficher sa position ----*/
           /*-- jusqu'à ce que les deux boutons soient appuyés  ----*/

ClrScr( 0x07 );
printfat( 43, 0, "JOYSTC - (c) 1991 MICHAEL TISCHER" );
printfat( 0, 24 , "Appuyez sur les deux boutons " \
                  "pour quitter le programme" );

xold = yold = 0;                        /* Prédéfinir l'ancienne Position */
do
 {
  GetJoyPos( &jsp[0], &jsp[1] );                /* Lire la position */

  /*-- Calculer la nouvelle position X du Joystick ----------------*/

  x = (int) ( xfacteur * (float) ( jsp[actstick].x - minx + 1 ) );
  if ( x < 0 )
    x = 0;
  if ( x > 79 )
    x = 79;

  /*-- Calculer la nouvelle position Y du Joystick ----------------*/

  y = (int) ( yfacteur * (float) ( jsp[actstick].y - miny + 1 ) );
  if ( y < 0 )
    y = 0;
  if ( y > 22 )
    y = 22;

  /*-- Afficher la nouvelle position si celle-ci a été modifiée -----*/

  if ( x != xold  ||  y != yold )
   {
    printfat( (BYTE) xold, (BYTE) (yold+1), " " );
    printfat( (BYTE) x, (BYTE) (y+1), "X" );
    xold = x;
    yold = y;
   }

  printfat( 0, 0, "(%3d,%3d)", jsp[actstick].x, jsp[actstick].y );
  GetJoyBouton( &j1but[0], &j2but[0], &j1but[1], &j2but[1] );
 }
while (!( j1but[actstick] == 1  && j2but[actstick] == 1 ));
ClrScr( 0x07 );
printf( "Fin de programme\n" );
}
```

11. Date, Heure et horloge temps réel

La date et l'heure sont des informations utilisées par DOS pour classer les fichiers. Mais les applications DOS doivent aussi accéder à ces informations pour faire par exemple apparaître la date en cours sur une lettre commerciale ou conserver l'heure de réception d'un appel téléphonique. Le BIOS dispose ainsi de nombreuses fonctions destinées à cet usage. De même, les AT, les 386 et 486 contiennent une horloge temps réel sur piles qui continue de fonctionner même si le PC est éteint.

Ce chapitre décrit :

- ✓ le mode de définition de la date et l'heure à l'aide de la ROM BIOS
- ✓ les fonctions BIOS étendues pour accéder à l'horloge temps réel
- ✓ la programmation directe de l'horloge temps réel et le remplissage de son registre.

11.1. Détermination de la date et l'heure avec le BIOS

L'interruption 1Ah du BIOS permet de solliciter les diverses fonctions de la ROM BIOS concernant l'heure. Par rapport aux PC et XT qui ne disposaient que de deux fonctions, il en existe 8 à partir de l'AT. Les fonctions étendues concernent surtout l'horloge temps réel sur piles (RTC, Real Time Clock) introduite avec l'AT. Elle est capable de calculer l'heure même si l'ordinateur est éteint. Cette méthode a changé considérablement le mode de calcul de l'heure connu jusqu'à présent dans les PC et XT puisqu'ils calculaient l'heure avec une interruption logicielle. Avant de décrire les diverses fonctions BIOS conçues pour obtenir la date et l'heure, voici d'abord un petit exposé sur le calcul de l'heure avant l'apparition de l'horloge temps réel.

Calcul de l'heure à l'aide de l'interruption Timer 08h

Le Chip Timer du PC (un Chip Intel 8254 ou compatible) reçoit 1.193.180 signaux par seconde en provenance du coeur du système par chaque oscillation du quartz. Avec une fréquence de 65536 signaux, soit près de 18,2 fois (exactement 18,20648193) par seconde, il crée un appel de l'interruption 08h qui est transmis à l'unité centrale à travers le contrôleur d'interruption. La fréquence d'oscillation du quartz est indépendante de la fréquence d'horloge, cela vaut naturellement pour l'appel de l'interruption 08h également. Elle convient parfaitement au calcul de l'heure puisqu'on sait qu'une seconde s'est écoulée après environ 18,2 appels.

Le BIOS dirige l'interruption 08h vers un gestionnaire d'interruptions approprié dans la ROM BIOS qui incrémente un compteur horaire interne à chaque appel. On peut obtenir l'heure en cours à n'importe quel moment grâce à ce compteur. Il doit être un

multiple de 18,2 pour que l'heure en secondes puisse être facilement convertie en heures, minutes et secondes.

Le gestionnaire d'interruptions attribue en outre à l'interruption 08h la tâche de désactiver le moteur des lecteurs de disquettes au bout d'un certain temps d'inactivité. Cela s'effectue en collaboration avec les fonctions de l'interruption BIOS 13h qui permet d'accéder à un lecteur de disquettes. Etant donné que l'activité d'un moteur de disquettes dure relativement longtemps, il reste en service après l'appel d'une des fonctions de disquettes BIOS pour qu'il ne soit pas nécessaire d'effectuer une réinitialisation lors d'une opération ultérieure.

Naturellement, il est inutile de surveiller le moteur minute par minute pendant que le prochain accès disquettes est en attente. C'est pourquoi, l'interruption Timer émet un temps d'arrêt au bout d'une fréquence déterminée à l'intérieur du gestionnaire d'interruptions.

Une fois que le gestionnaire d'interruptions a accompli son travail en faveur de l'interruption Timer, il appelle l'interruption 1Ch. Normalement, une instruction IRET se cache derrière cette interruption. Celle-ci retourne immédiatement l'exécution du programme au gestionnaire de l'interruption Timer. Ainsi, un programme peut définir un gestionnaire destiné spécialement à cette interruption pour participer au cycle du signal horaire. Ce gestionnaire est également appelé 18,2 fois par seconde. Cela est particulièrement intéressant lorsqu'il s'agit par exemple d'afficher en permanence l'heure actuelle sur l'écran, comme le montre le programme TSR décrit au chapitre 31.

L'interruption Timer 08h dont l'unique tâche consiste à calculer l'heure dans les PC et XT, les ordinateurs AT, 386 et 486 disposent d'une horloge temps réel sur piles qui détermine l'heure sans l'aide d'une interruption Timer. Mais la ROM BIOS de l'AT et des modèles ultérieurs continue néanmoins à gérer l'interruption Timer car son rôle essentiel est d'assurer la compatibilité logicielle avec le PC et le XT.

Demander l'heure

Les deux fonctions d'heure élémentaires concernent la lecture et le réglage de l'heure. La première porte le numéro de fonction 00h et est appelée dans le registre AH à l'aide de ce numéro. Elle retourne l'heure dans les registres CX et DX. Ces deux registres forment ensemble un compteur 32 bits (CX contient les 16 bits de poids fort, DX les bits de poids faible) qui ne reproduit pas l'heure sous forme d'heures ou de minutes. En fait, il s'agit d'une valeur qui augmente à chaque appel de l'interruption Timer via le BIOS. Ce compteur peut être converti en heures, minutes et secondes en multipliant le contenu du registre CX par 65536 et en y ajoutant le contenu du registre DX. En divisant ensuite cette valeur par 18,2, on obtient l'heure en secondes. Pour convertir à nouveau cette valeur en minutes et secondes, il suffit de la diviser par 60.

Dans les PC/XT, le résultat de ce calcul doit être interprété différemment par rapport aux AT et les modèles ultérieurs. Dans un PC/XT, le BIOS fixe le compteur à 0 lors du lancement du système si bien que le résultat du calcul ne concerne pas l'heure actuelle mais la période qui s'est écoulée depuis l'allumage de l'ordinateur. Pour obtenir la véritable durée, on doit d'abord convertir l'heure actuelle par la valeur correspondante dans le compteur et la transmettre ensuite au BIOS comme l'heure en cours.

Dans le cas de l'AT et des modèles ultérieurs, il n'est pas indispensable d'ajuster le compteur puisque le BIOS lit l'heure actuelle dans l'horloge lors du lancement du système puis la convertit en une valeur correspondante dans le compteur. Grâce à la lecture du compteur avec la fonction 00h, on obtient toujours l'heure actuelle dans un AT.

Après appel de la fonction horaire 00h, on obtient une autre valeur dans le registre AL. Au cas où il s'agit de la valeur 0, la valeur transmise indique que 24 heures ne se sont pas encore écoulées depuis la dernière lecture de l'heure. Si le registre reçoit une valeur différente de 0, cela signifie que 24 heures se sont écoulées sans que l'on puisse connaître le nombre de cycles de 24 heures courus par le compteur.

Si la conversion de la valeur horaire en heures, minutes et secondes semble trop compliquée pour vous, n'hésitez pas à utiliser la fonction 2Ch de l'interruption DOS 21h. Celle-ci se contente de lire l'heure actuelle au moyen de la fonction 00h de l'interruption 1Ah et la convertit en un format quelque peu maniable. Pour plus d'informations à ce sujet, reportez-vous au chapitre 21.

Régler l'heure

La fonction 01h représente la seconde fonction horaire élémentaire. Elle sert à régler le compteur horaire interne du BIOS sur une valeur précise. Cette fonction est appelée avec le numéro de fonction contenu dans le registre AH, les 16 bits de poids fort du registre CX et les 16 bits de poids faible du registre DX. Si la conversion de l'heure actuelle en une valeur du compteur ne vous inspire pas, vous pouvez utiliser une fonction DOS dans ce cas également. Il s'agit en l'occurrence de la fonction 2Dh de l'interruption DOS 21h. Reportez-vous au chapitre 21 pour de plus amples informations.

Fonctions pour l'accès à l'horloge temps réel

Les 6 fonctions suivantes sont disponibles uniquement sur l'AT et les modèles ultérieurs. Il existe certes des horloges en temps réel sur les PC et XT, mais elles ne sont généralement pas soutenues par la ROM BIOS de l'ordinateur. Il suffit que le fabricant ne fournisse pas un petit programme TSR en vue d'installer les fonctions BIOS nécessaires de l'AT pour que l'appel de ces fonctions soit ignoré sur les PC et XT. Le flag Carry est d'ailleurs réglé plusieurs fois pour signaler l'échec de l'appel de ces fonctions. Il faut donc que votre programme n'utilise ces fonctions que s'il est parfaite-

ment sûr de disposer d'un AT et d'un ordinateur 386 ou 486. A cet effet, vous ne devez pas hésiter à contrôler l'identification du modèle, comme il est décrit au chapitre 3.

Les 6 fonctions utilisent le format BCD pour indiquer l'heure et la date. Dans ce format, deux chiffres d'un nombre sont codés par octet dont le nombre de poids fort est codé dans le Nibble de poids fort et le nombre de poids faible dans le Nibble de poids faible. Les 6 fonctions utilisent également le flag Carry pour retourner les valeurs. S'il est réglé, l'horloge ne fonctionne pas convenablement (lorsque la pile est par exemple à plat). Dans ce cas, la fonction appelée ne peut pas s'exécuter correctement.

Lire et sauvegarder l'heure provenant de l'horloge

La fonction 02h permet de lire l'heure stockée dans l'horloge temps réel. Après l'appel avec la valeur 2 du registre AH, vous obtenez l'heure en cours dans le registre CH, la minute dans le registre CL et les secondes en DH.

La fonction 03h est appelée selon la même méthode à partir du contenu du registre. Elle sert à régler l'heure de l'horloge. Ici aussi, le numéro de fonction est placé dans le registre AH, l'heure en CH, la minute en CL et les secondes en DH. Dans le registre DL, vous pouvez spécifier également si vous souhaitez l'heure d'été "daylight savings time option". Un 1 sélectionne l'heure d'été, 0 détermine l'heure normale. Dans le premier cas, l'horloge convertit automatiquement l'heure en heure d'été lorsqu'arrive la date prévue.

Gérer la date dans l'horloge temps réel

Alors que les fonctions 02h et 03h servent à régler et lire l'heure de l'horloge, les fonctions 04h et 05h concernent la date sauvegardée dans l'horloge. En guise d'arguments, les deux fonctions utilisent le siècle, l'année, le mois et le jour. Le jour de la semaine géré par l'horloge n'entre pas en ligne de compte. Si vous souhaitez en prendre connaissance, vous devez accéder directement à l'horloge. Lisez les sections suivantes pour en savoir davantage.

La fonction 04h est appelée avec le numéro de fonction contenu dans le registre AH. Elle sert à lire la date en cours. Après l'appel, le registre CH reçoit les deux premiers chiffres de l'année, soit le siècle. Ici, seules les valeurs 19 et 20 sont autorisées. Le registre CL obtient les deux derniers chiffres de l'année, par exemple 92. Le mois est retourné dans le registre DH, le jour du mois en DL.

Lors de son appel, la fonction 05h attend ces mêmes valeurs de registre pour changer la date sauvegardée dans l'horloge. Faites attention à la diversité des valeurs du registre AH qui ne doit pas contenir 04h, mais la valeur 05h.

Installer l'heure de l'alarme

La fonction 06h permet de programmer l'heure de l'alarme. Comme il s'agit d'indiquer ici l'heure, la minute et la seconde uniquement, cette heure d'alarme concerne toujours le jour en cours. Si l'heure de l'alarme est atteinte, l'horloge appelle une routine BIOS qui déclenche à son tour l'interruption 4Ah. Cette interruption permet d'installer une routine spécifique créant par exemple des bips sonores pour simuler un réveil.

Lors de l'initialisation du système, l'interruption 4Ah se place par exemple sur une routine qui reçoit uniquement l'instruction IRET. Cette dernière autorise l'unité centrale à terminer immédiatement l'interruption si bien que l'heure du déclenchement de l'alarme est atteinte à l'insu de l'utilisateur.

Lors de l'appel de la fonction 06h, la valeur 06h doit se trouver dans le registre AH. L'heure d'alarme doit figurer dans le registre CH, la minute en CL et la seconde en DH. Notez que vous ne pouvez installer qu'une seule heure d'alarme. Lorsque vous appelez cette fonction pendant qu'une autre heure d'alarme se trouve encore en place ou n'est pas encore atteinte, vous devez régler le flag Carry après l'appel de la fonction. Ainsi, l'ancienne alarme n'est pas remplacée par la nouvelle. Si vous tenez absolument à fixer une nouvelle heure d'alarme ou supprimer l'ancienne, vous devez d'abord appeler la fonction 07h.

Lors de l'appel, aucun autre argument, excepté le numéro de fonction, ne doit se trouver dans le registre AH. Cet appel supprime la dernière heure d'alarme, ce qui permet de programmer une nouvelle heure si vous le souhaitez.

11.2. Déterminer et programmer l'horloge temps réel

L'AT et les modèles ultérieurs comportent de façon standard sur leur carte mère une horloge temps réel alimentée sur piles. Le terme utilisé en anglais est RTC (Real Time Clock). Cette horloge fait partie d'un processeur MC 146818 de la société Motorola, qui comporte en plus de cette horloge une RAM de 64 octets alimentée sur piles. Cette mémoire RAM, qu'il est donc possible de lire ou d'écrire, reçoit les données utiles pour l'horloge mais aussi un certain nombre de données de configuration du système général. Elle peut être appelée à travers les adresses de port 70h à 7Fh mais nous verrons par la suite que ce sont essentiellement les ports 70h et 71h qui peuvent nous intéresser.

Les cellules de mémoire de l'horloge temps réel

Intéressons-nous tout d'abord à l'horloge et aux cellules de mémoire qu'elle occupe. Comme l'indique la table suivante, il s'agit des cellules de mémoire 00h à 0Dh ainsi que de la cellule 32h.

Cellules de mémoire de l'horloge dans la RAM alimentée sur piles	
Adresse	**Contenu**
0	Seconde actuelle
1	Seconde de l'alarme
2	Minute actuelle
3	Minute de l'alarme
4	Heure actuelle
5	Heure de l'alarme
6	Jour de la semaine
7	Jour du mois
8	Mois
9	Année
0Ah	Registre d'état A de l'horloge
0Bh	Registre d'état B de l'horloge
0Ch	Registre d'état C de l'horloge
0Dh	Registre d'état D de l'horloge
32h	Siècle (19 ou 20)

Comme vous le voyez, chacun des trois champs horaires (Secondes, minutes, heures) est suivi d'un champ d'alarme correspondant. Les champs d'alarme permettent de fixer une heure d'alarme (mais seulement pour le jour même). Lorsque cette heure sera atteinte, une interruption déterminée sera déclenchée. Mais nous y reviendrons bientôt plus en détail...

Le champ "jour de la semaine" indique le numéro du jour actuel de la semaine, la valeur 1 représentant le Dimanche. La valeur 2 correspond donc à Lundi, 3 à Mardi, etc...

L'année est indiquée par rapport au siècle. Si ce champ contient la valeur 88, nous aurons par exemple l'année 1988.

Accès aux différents registres de l'horloge temps réel

Ce registre est un élément tout à fait normal de la RAM de 64 octets dont est doté ce circuit électronique. Il est donc possible d'y accéder exactement de la même façon que pour toutes les autres cellules de mémoire de ce circuit :

On charge tout d'abord le numéro de la cellule de mémoire à adresser dans le registre AL (dans le cas du registre d'état A ce sera donc 10). On envoie ensuite cette valeur sur le port 70h à l'aide de l'instruction OUT. Le circuit est ainsi prévenu que l'on veut accéder à l'une de ces cellules de mémoire. L'étape suivante consiste alors soit à exécuter une instruction OUT sur le port 71h, pour écrire dans cette cellule de mémoire, soit à appeler une instruction IN pour lire sur le port 71h le contenu de la cellule de mémoire.

Lecture d'une cellule de mémoire de l'horloge temps réel

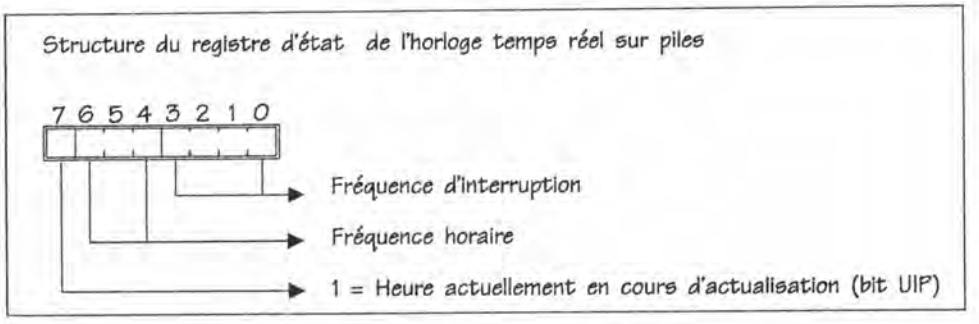
```
mov al,Cellule
out 70h,al
in  al,71h
```

Ecriture dans une cellule de mémoire de l'horloge temps réel

```
mov  al,Cellule
out  70h,al
mov  al,Nouveau_contenu
out  71h,al
```

Le registre d'état A

Les 4 registres d'état de l'horloge présentent un intérêt tout particulier pour nous car ils permettent de programmer l'horloge.



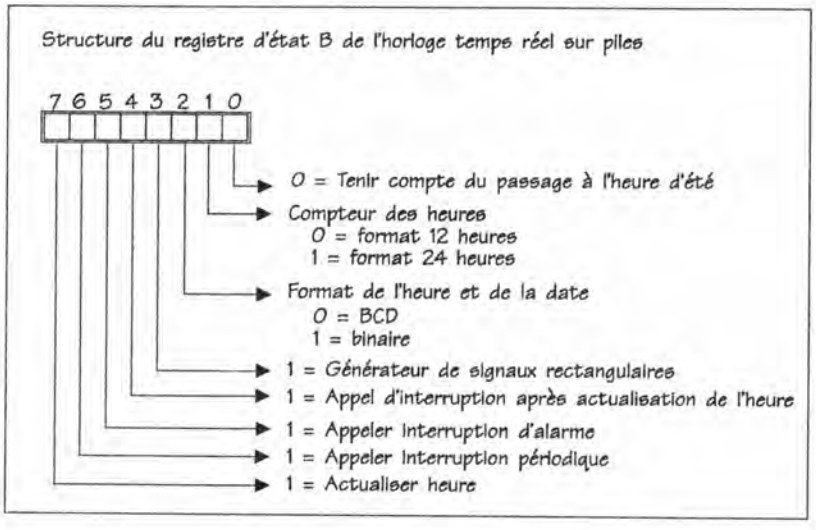
Les deux champs inférieurs de ce registre sont fixés par le BIOS en ROM au cours de l'initialisation du système et ne sont plus modifiés par la suite. Le champ de fréquence d'interruption reçoit la valeur 0110b. Cette valeur entraîne une fréquence d'interruption de 1024 interruptions à la seconde (une interruption toutes les 976,562 microsecondes).

Le contenu du champ de fréquence d'horloge est de 010b, ce qui entraîne une fréquence d'horloge de 32 768 Kilohertz.

Pour le programmeur, c'est le bit 7 du registre d'état qui est intéressant par rapport à ces deux champs. Il indique en effet si une seconde vient de s'achever et si les champs horaires (secondes et éventuellement minutes, heures, etc...) sont en train d'être incrémentés (d'où son nom : UIP = Update In Progress). Si c'est le cas, ce bit contient un 1. Ce bit est intéressant car il est déconseillé de lire les différents champs horaires lorsqu'ils sont en cours d'actualisation. Il peut sinon arriver qu'une minute vienne juste de s'écouler et que le compteur des secondes ait déjà été fixé sur 0 mais que le compteur des minutes soit lu avant d'avoir pu être incrémenté. La conséquence serait (pour une horloge affichée sur l'écran, par exemple) que l'heure sauterait de 13:59:59 à 13:59:00 avant d'afficher, à juste titre, 14:00:01 à la seconde suivante. Mais comment accéder au registre d'état A ?

Le registre d'état B

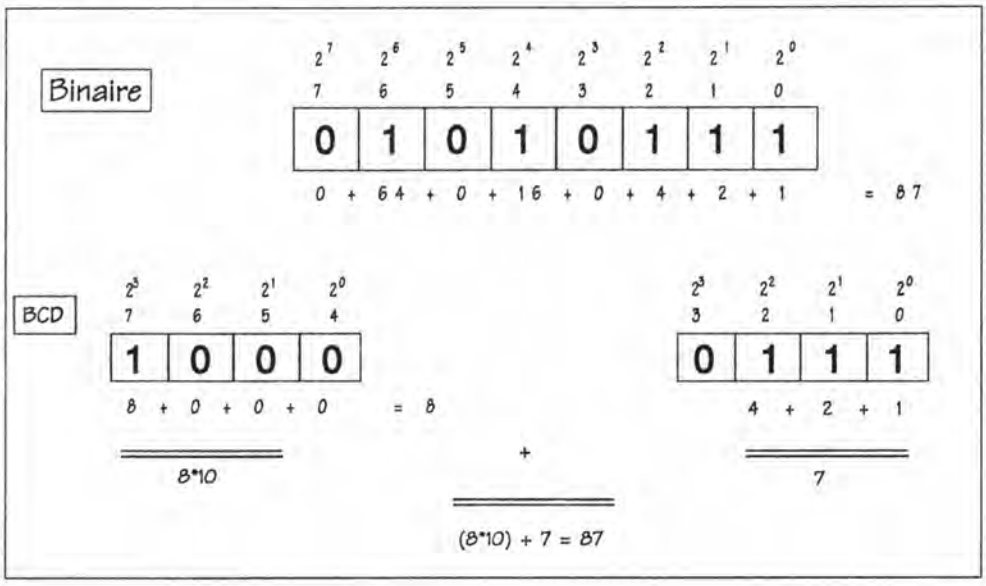
Le registre d'état B permet de programmer certains paramètres de l'horloge :



Comme la France applique l'heure d'été chaque année, le bit 0 du registre d'état B nous intéresse. Si nous fixons en effet ce bit sur 1, nous indiquons que c'est l'heure d'été qui s'applique actuellement. La valeur par défaut de ce bit est 0, elle indique au contraire que l'heure d'été ne s'applique pas.

Le bit 1 fixe si l'horloge doit être exploitée en mode de 12 heures ou en mode de 24 heures. En mode de 12 heures, l'horloge revient à une heure à la fin de chaque cycle de 12 heures (après midi et minuit) alors qu'elle ne revient à 1 heure qu'au bout de 24 heures en mode de 24 heures. C'est le mode de 24 heures, qui est le mode le plus couramment utilisé en Europe, qui est fixé lors du lancement du système.

Le format de stockage des champs d'heure et de date est défini par le bit 2. Si ce bit contient un 1, les différentes données seront stockées, comme d'habitude, en binaire. L'année (19)87 sera donc codée sous la forme 01010111b. La valeur 0 dans le bit 2 sélectionne par contre le format BCD. Sous ce format, deux chiffres décimaux seront stockés dans chaque octet, le chiffre le plus élevé dans les 4 bits de plus fort poids et le chiffre le moins élevé dans les 4 bits de plus faible poids.



Le nombre 87 en formats binaire et BCD

Ce bit contient normalement un 0, de sorte que les nombres sont stockés en format BCD.

Il importe, à ce propos, de ne jamais oublier que le BIOS exige, par exemple pour les fonctions de date appelées à travers l'interruption 1Ah, un format BCD de stockage de ces nombres. Les programmes d'application qui appellent ces fonctions du BIOS risquent donc d'être induits en erreur s'ils reçoivent ces informations en format binaire au lieu du format BCD standard. Il en va de même pour les modes 12 ou 24 heures, même si les conséquences d'un passage au mode de 12 heures sont beaucoup moins graves que celles d'une modification du format de stockage des informations de date et heure.

Le bit 4 définit si une interruption doit être appelée après actualisation de l'heure et éventuellement de la date. Si oui, ce bit devra contenir un 1. Le système interdit au départ cette interruption en fixant ce bit sur 0 lors de l'initialisation.

Si une alarme doit être déclenchée à une heure déterminée, le bit 5 doit être fixé sur 1. L'horloge tire l'heure d'alarme des cellules de mémoire 1, 3, et 5 (secondes, minutes et

heures) de la mémoire RAM de l'horloge. Lorsque l'heure de l'alarme est atteinte, une interruption est déclenchée. Cette interruption est également interdite au départ par le système qui fixe le bit 5 sur 0 lors de l'initialisation.

L'interruption est appelée périodiquement si le bit 6 contient un 1. La fréquence de cette interruption est codée par les bits 0 à 3 du registre d'état A. C'est une fréquence de 1024 kilohertz qui y est fixée lors du lancement du système, de sorte que l'interruption sera appelée toutes les 967 562 microsecondes. Le bit 6 est toutefois également fixé sur 0 lors du lancement du système, de sorte qu'un programme d'application devra d'abord le fixer sur 1 pour que l'interruption soit appelée périodiquement.

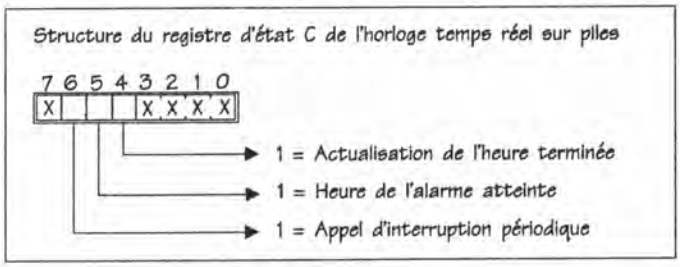
Le bit 7 contrôle enfin l'actualisation périodique (toutes les secondes) de l'heure et de la date. Ici cependant, 0 ne signifie pas actualisation désactivée et 1 actualisation activée mais exactement le contraire. C'est pourquoi ce bit est également fixé sur 0 lors du lancement du système, de façon à ce que l'heure coure en permanence. Lorsque vous voulez inscrire une heure entièrement nouvelle avec une nouvelle date dans les différentes cellules de mémoire prévues à cet effet, il convient donc de fixer auparavant ce bit sur 1 pour que l'horloge ne recommence pas immédiatement à modifier votre nouvelle heure. Une fois toutes les données inscrites, vous pourrez alors annuler ce bit à nouveau et l'heure continuera de courir.

L'interruption RTC 70h

Dans cette description des différents bits, nous avons employé à plusieurs reprises l'expression "appeler l'interruption" sans préciser quelle interruption il s'agit au juste d'appeler. Bien que l'horloge puisse avoir des motifs différents pour appeler une interruption (heure de l'alarme atteinte, interruption périodique, etc...), c'est cependant toujours la même interruption qu'elle appellera, l'interruption 70h. Cette interruption abrite une routine du BIOS qui est chargée, entre autres, de la gestion des deux fonctions "horaires" de l'interruption 15h. Mais si cette routine est appelée à diverses occasions, comment fait-elle pour identifier le motif de chaque appel ?

Le registre d'état C

Elle utilise à cet effet le registre d'état C de l'horloge. Seuls les bits 4, 5 et 6 de ce registre sont significatifs. Leur valeur coïncide avec celle des bits correspondants du registre d'état B. Cela signifie donc que si l'interruption d'alarme est déclenchée, par exemple, ce qui suppose que le bit 5 soit fixé dans le registre d'état B, le bit 5 sera également fixé dans le registre d'état C pour indiquer que l'heure d'alarme est atteinte.



La première tâche de la routine qui intercepte l'interruption 70h consiste donc à lire le registre d'état C, de façon à identifier le motif de l'appel d'interruption et à réagir en conséquence.

Le registre d'état D

Le dernier registre d'état, le registre D, est le plus "pauvre" puisque seul le bit 7 de ce registre est significatif. Il indique l'état des piles qui alimentent la sauvegarde des données même lorsque le PC est éteint. Si ce bit vaut 0, vous avez tout intérêt à vous procurer des piles car les piles actuelles sont à plat.

Le registre d'état D est le dernier registre de la zone de données consacrée à l'horloge temps réel. Ensuite viennent les informations sur la configuration de la machine.

11.3. Informations sur la configuration

Outre les diverses informations concernant la date et l'heure, les 64 cellules de mémoire de l'horloge temps réel sur piles contiennent d'autres informations qui sont énumérées dans le tableau suivant.

Les différents constructeurs BIOS n'adoptent une formule standard que pour les cellules de mémoire définies comme des cellules réservées. Les autres peuvent être utilisées par les fabricants de BIOS ou d'électronique comme ils le souhaitent. Par conséquent, elles ne doivent pas être écrasées par un programme.

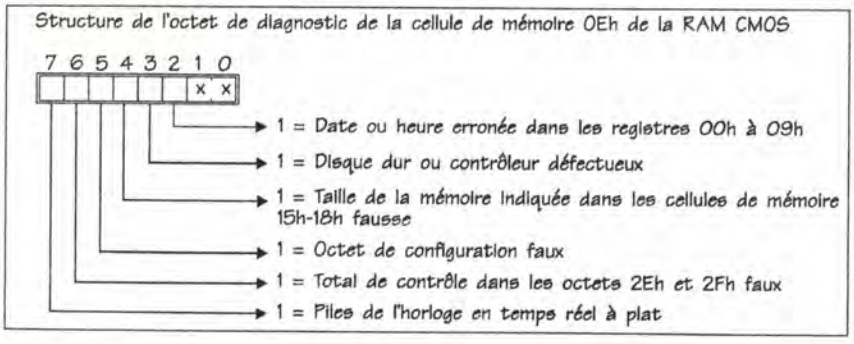
Informations de configuration dans les cellules de mémoire de l'horloge temps réel sur piles

Adresse	Contenu
0Eh	Octet de diagnostic
0Fh	Etat lors de l'arrêt du système
10h	Description des disquettes

Informations de configuration dans les cellules de mémoire de l'horloge temps réel sur piles	
Adresse	**Contenu**
11h	Type du premier disque dur
12h	Type du second disque dur
13h	Réservé
14h	Octet de configuration
15h	Octet faible de la taille en Ko de la mémoire sur la carte mère
16h	Octet fort de la taille en Ko de la mémoire sur la carte mère
17h	Octet faible de la taille en Ko de la mémoire sur une carte supplémentaire
18h	Octet fort de la taille en Ko de la mémoire sur une carte supplémentaire
19h-2Dh	Réservé
2Eh	Octet fort de la somme de contrôle des cellules de mémoire 10h-2Dh
2Fh	Octet faible de la somme de contrôle des cellules de mémoire 10h-2Dh
30h	Octet faible de la taille en Ko de la mémoire d'extension
31h	Octet fort de la taille en Ko de la mémoire d'extension
32h	Les deux premiers chiffres du siècle en format BCD
33h-3Fh	Réservé

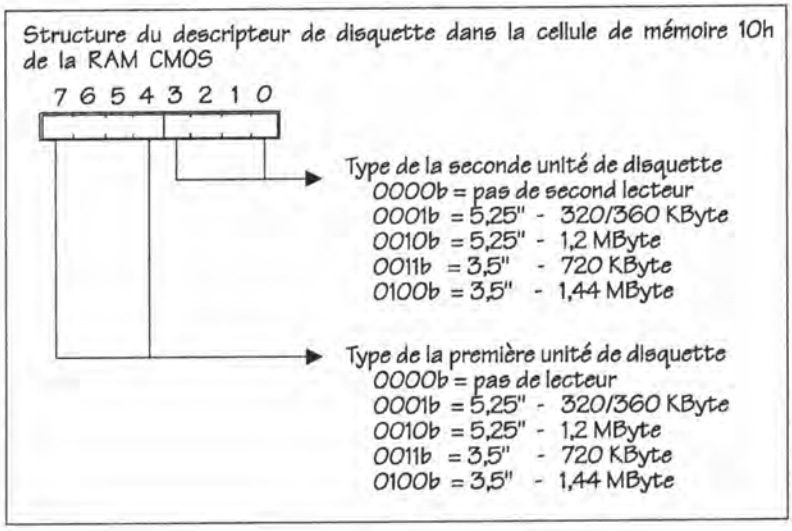
Octet de diagnostic (0Eh)

Toutes sortes de sources d'erreurs, produites lors de l'initialisation du système (POST), sont placées dans l'octet de diagnostic.



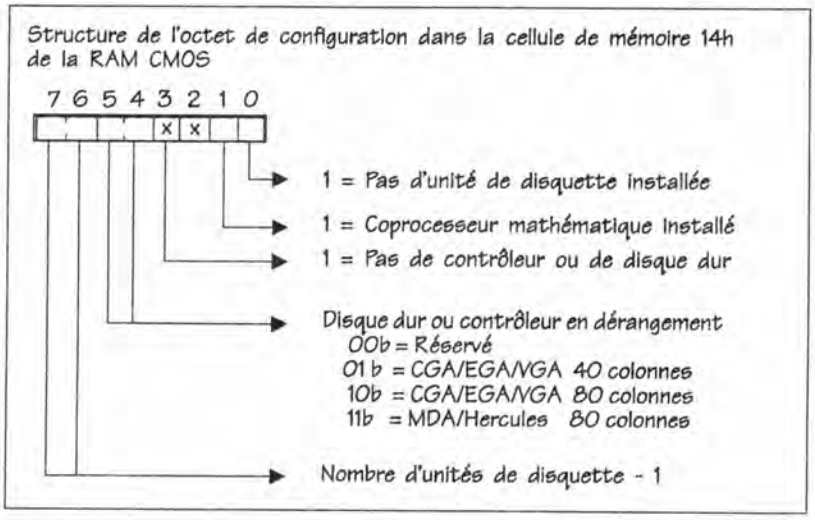
Description des lecteurs de disquettes (10h)

La cellule de mémoire 10h de la RAM sur piles contient des informations sur les premier et second lecteurs de disquettes dont dépendent leur format (5,25" ou 3,5") et leur capacité.



Octet de configuration (14h)

La cellule de mémoire 14h de la RAM sur piles contient des informations sur la configuration décrivant le nombre de lecteurs de disquettes, le mode vidéo lors du lancement du système et l'existence d'un co-processeur mathématique.



11.4. Programmes d'exemple

Pour illustrer l'accès à l'horloge sur piles à partir des trois langages évolués, voici maintenant trois programmes d'exemple dans les langages BASIC, Pascal et C. Le coeur de ces programmes est chaque fois constitué par trois routines. La première lit une valeur dans l'une des cellules de mémoire de l'horloge, la seconde y inscrit une valeur et la troisième lit également une valeur mais seulement après avoir testé si l'horloge est exploitée en mode binaire ou BCD ; elle convertit le cas échéant cette valeur de BCD en binaire. Cette routine est importante pour l'accès à toutes les cellules de mémoire contenant les informations de date ou d'heure car ces cellules peuvent être codées aussi bien en format BCD qu'en format binaire.

Le programme principal appelle les deux routines qui lisent le contenu de cellules de mémoire pour, entre autres, lire la date et l'heure actuelles sur l'horloge temps réel et sortir ces informations sur l'écran. On teste toutefois auparavant si les piles de l'horloge ne sont pas à plat, auquel cas ces données seraient sans valeur.

La routine pour écrire dans des cellules de mémoire n'est pas appelée par le programme principal. Il ne devrait cependant pas vous être difficile de modifier chaque programme de façon à ce que la date ou l'heure soient écrites sur l'horloge, à l'aide de la routine d'écriture, au lieu d'être lues. C'est une suggestion que vous pourriez retenir si vous souhaitez vous livrer à diverses expériences avec ces routines.

Voici maintenant les listings des trois programmes. Ils sont documentés de façon suffisamment complète pour que des explications supplémentaires apparaissent superflues.

Listing : RTCB.BAS

```
'*****************************************************************
'*                          R T C B                              *
'*---------------------------------------------------------------*
'* Fonction       : fournit deux sous-programmes qui permettent  *
'*                  de lire des données ou d'en écrire sur la    *
'*                  RTC                                          *
'* Auteur         : MICHAEL TISCHER                              *
'* développé le   : 24.7.87                                      *
'* dernière modif.: 17.2.92                                      *
'*****************************************************************
'
DECLARE FUNCTION RTCRead% (Adresse%)
DECLARE FUNCTION RTCDT% (Adresse%)

CLS                                                 'Vider l'écran
PRINT "RTC (c) 1987, 92 by Michael Tischer": PRINT
PRINT "Informations tirées de l'horloge temps réel sur piles"
PRINT "-----------------------------------------"
PRINT

IF (RTCRead(&HE) AND 128) = 128 THEN          'Bit 7=1 --> piles vide
  PRINT "       ATTENTION ! Les piles de l'horloge sont vides"
ELSE
  PRINT "- L'horloge est exploitée en mode";
  PRINT (RTCRead(&HB) AND 2) * 6 + 12; "heures"

  PRINT "- Il est : ";
  PRINT USING "##:"; RTCDT(&H4);
  PRINT USING "##:"; RTCDT(&H2);
  PRINT USING "##"; RTCDT(&H0)

  PRINT "- Nous sommes le : ";
  PRINT USING "##"; RTCDT(&H7);
  PRINT USING ".##"; RTCDT(&HB) / 100;
  PRINT USING ".####"; (RTCDT(&H32) * 100 + RTCDT(&H9)) / 10000
  PRINT

END IF

'*****************************************************************
'* Lire contenu de l'une des cellules de mémoire de la RTC       *
'*---------------------------------------------------------------*
'* Entrée : Adresse% = numéro de la cellule de mémoire (0 à 63) *
'* Sortie : contenu de cette cellule de mémoire                  *
'*****************************************************************
'
FUNCTION RTCDT% (Adresse%)

  Ret% = RTCRead(Adresse%)          'Lecture registre d'adresse RTC
  IF (RTCRead(&HB) AND 4) <> 4 THEN
    RTCDT% = (Ret% AND 15) + INT(Ret% / 16) * 10
  ELSE
    RTCDT% = Ret%
  END IF

END FUNCTION

'*****************************************************************
'* Ecrire une cellule de mémoire dans RTC                        *
'*---------------------------------------------------------------*
'* Entrée : Adresse% = numéro de la cellule de mémoire (0 à 63) *
'* Sortie : aucune                                               *
'*****************************************************************
'
FUNCTION RTCRead% (Adresse%)

  IF (Adresse% < 0) OR (Adresse% > 63) THEN
    RTCRead% = -1
  ELSE
    OUT &H70, Adresse%   'Envoyer n° cellule sur reg d'adresse RTC
    RTCRead% = INP(&H71)          'Lire le contenu du reg données RTC
  END IF

END FUNCTION

'*****************************************************************
'* Lire contenu d'une des cellules de date ou d'heure dans la    *
'* RTC et convertir en décimal                                   *
'*---------------------------------------------------------------*
'* Entrée : Adresse% = numéro de la cellule de mémoire (0 à 63) *
'*          Valeur% = nouvelle valeur de la cellule              *
'* Sortie : aucune                                               *
'*****************************************************************
'
SUB RTCWrite (Adresse%, Valeur%)

  OUT &H70, Adresse%
  OUT &H71, Valeur%

END SUB
```

Listing : RTCP.PAS

```
{*****************************************************************}
{*                          R T C P                              *}
{*---------------------------------------------------------------*}
{*   Fonction       : fournit deux fonctions qui permettent d'écrire *}
{*                    ou de lire des données sur l'horloge en temps  *}
{*                    réel                                           *}
{*---------------------------------------------------------------*}
{*   Auteur         : MICHAEL TISCHER                              *}
{*   développé le   : 10/07/1987                                   *}
{*   dernière modif.: 17/02/1992                                   *}
{*****************************************************************}

program RTCP;

Uses Crt;                                          { Intégrer unité CRT }

type Jours = array[1..7] of string[10];       { Tableau des noms jours }

const RTCAdrPort  = $70;                  { Registre d'adresse de la RTC }
      RTCDtaPort  = $71;                  { Registre de données de la RTC }

      SECONDES    = $00;  { Adresses de quelques cellules mémoires RTC }
      MINUTES     = $02;
      HEURES      = $04;
      JOURSEMAINE = $06;
      JOUR        = $07;
      MOIS        = $08;
      ANNEE       = $09;
      ETATA       = $0A;
      ETATB       = $0B;
      ETATC       = $0C;
      ETATD       = $0D;
      DIAGNOSTIC  = $0E;
      SIECLE      = $32;

{*****************************************************************}
{* RTCRead : Lire le contenu d'un des cellules de mémoire de la RTC  *}
{* Entrée  : Adresse de la cellule de mémoire dans la RTC            *}
{* Sortie  : Contenu de cette cellule de mémoire                     *}
{* Info    : si l'adresse sort du domaine autorisé (0 à 63), la      *}
{*           valeur -1 sera renvoyée                                 *}
{*****************************************************************}

function RTCRead(Adresse : integer) : integer;

begin
  if (Adresse < 0) or (Adresse > 63)              { Adresse correcte ? }
  then RTCRead := -1                                        { NON ! }
  else
    begin
      Port[RTCAdrPort] := Adresse;          { Transmettre adresse à la RTC }
      RTCRead := Port[RTCDtaPort]                    { Lire son contenu }
    end;
end;

{*****************************************************************}
{* RTCDT: lit une des cellules de mémoire de la date ou de l'heure de *}
{*        la RTC et convertit le résultat en une valeur binaire si la *}
{*        RTC travaille en format BCD                                 *}
{* Entrée  : Adresse de la cellule de mémoire dans la RTC            *}
{* Sortie  : Contenu de cette cellule de mémoire comme valeur binaire *}
{* Info    : Si l'adresse est en dehors du domaine autorisé (0 - 63), *}
```

```
{*          la valeur -1 sera renvoyée                          *}
{*****************************************************************}

function RTCDT(Adresse : integer) : integer;

var Valeur : integer;                    { Pour stocker une valeur lue }

begin
 Valeur := RTCRead(Adresse);             { Lire contenu de cellule mémoire }

 if( RTCRead(ETATB) and 4 = 4)                    { Mode BCD ou binaire ? }
  then
   RTCDT := Valeur                               { C'est le mode binaire }
  else                                             { C'est le mode BCD }
   RTCDT := (Valeur shr 4) * 10 + Valeur and 15; { Conv BCD en binaire }
end;

{*****************************************************************}
{* RTCWrite: Ecrire une valeur dans l'une des cellules de la RTC  *}
{* Entrée  : Voir plus bas                                        *}
{* Sortie  : aucune                                               *}
{* Info    : Adresse doit être comprise entre 0 et 63             *}
{*****************************************************************}

procedure RTCWrite(Adresse : integer;  { Adresse de cellule de mémoire }
                   Contenu : byte);          { Son nouveau contenu }

begin
 Port[RTCAdrPort] := Adresse;            { Transmettre adresse à la RTC }
 Port[RTCDtaPort] := Contenu                    { Ecrire nouvelle valeur }

end;

{*****************************************************************}
{*                      PROGRAMME PRINCIPAL                      *}
{*****************************************************************}

begin

 ClrScr;                                                { Vider l'écran }
 writeln('RTC (c) 1987, 92 by Michael Tischer'#13#10);
 writeln('Informations tirées de l''horloge temps réel');
 writeln('-------------------------------------------'#13#10);

 if RTCRead(Diagnostic) and 128 = 0 then          { Piles en bon état ? }
  begin                                                        { Oui }
   writeln('- L''horloge est exploitée en mode ',
           (RTCRead(ETATB) and 2)*6+12,
           ' heures');
   writeln('- Il est : ', RTCDT(HEURES), ':', RTCDT(MINUTES):2,
           ':', RTCDT(SECONDES):2);
   write('- Nous sommes le : ');
   writeln(RTCDT(JOUR), '.', RTCDT(MOIS),
           '.',RTCDT(SIECLE), RTCDT(ANNEE));
  end
 else                                              { Piles de la RTC vides }
  write('      ATTENTION ! Les piles de l''horloge sont vides')
end.
```

Listing : RTCC.C

```
/*****************************************************************/
/*                              R T C                            */
/*---------------------------------------------------------------*/
/*   Fonction       : fournit deux fonctions permettant d'écrire des */
/*                    données ou d'en lire sur l'horloge en temps  */
/*                    réel                                         */
/*---------------------------------------------------------------*/
/*   Auteur         : MICHAEL TISCHER                              */
/*   développé le   : 15/08/1987                                   */
/*   dernière modif.: 17/02/1992                                   */
/*---------------------------------------------------------------*/
/*   (MICROSOFT C)                                               */
/*   Création       : CL /AS RTC.C                                 */
/*   Appel          : RTC                                          */
/*---------------------------------------------------------------*/
/*   (BORLAND TURBO C)                                           */
/*   Création       : Avec instruction RUN dans ligne d'instruction */
/*                    (sans fichier Project)                       */
/*****************************************************************/

/*-- Intégrer fichiers Include ------------------------------------*/

#include <dos.h>                          /* Intégrer fichier header */
#include <stdio.h>
#include <conio.h>

/*-- Typedefs -----------------------------------------------------*/

typedef unsigned char byte;      /* Voilà comment se bricoler un BYTE */

/*-- Constantes ---------------------------------------------------*/

#define RTCAdrPort  0x70              /* Registre d'adresse de la RTC */
#define RTCDtaPort  0x71             /* Registre de données de la RTC */

#define SECONDES    0        /* Adresses de cellules de mémoire RTC */
#define MINUTES     2
#define HEURES      4
#define JOURSEMAINE 6
#define JOUR        7
#define MOIS        8
#define ANNEE       9
#define ETATA      10
#define ETATB      11
#define ETATC      12
#define ETATD      13
#define DIAGNOSTIC 14
#define SIECLE     50

/*****************************************************************/
/* RTCREAD: lire le contenu d'une des cellules de mémoire de la RTC  */
/* Entrée  : adresse de la cellule de mémoire dans la RTC        */
/* Sortie  : le contenu de cette cellule de mémoire              */
/*****************************************************************/

byte RTCRead(byte Adresse)
{
  (void)outp(RTCAdrPort, Adresse );   /* Communiquer adresse à la RTC */
  return( inp( RTCDtaPort ) );                /* Lire et transmettre */
}

/*****************************************************************/
/* RTCDT: lit une des cellules de mémoire de la date ou de l'heure   */
/*        et convertit le résultat en une valeur binaire si      */
/*        l'horloge travaille en format BCD                      */
/* Entrée  : adresse de la cellule de mémoire dans la RTC        */
/* Sortie  : contenu de cette cellule de mémoire comme valeur binaire */
/* Infos   : si l'adresse sort du domaine autorisé (0 à 63), la  */
/*           valeur -1 sera renvoyée                             */
/*****************************************************************/

byte RTCDt(byte Adresse)
{
  if( !(RTCRead( ETATB ) & 4))                /* Mode BCD ou binaire ? */
    return((RTCRead(Adresse) >> 4) * 10 + (RTCRead(Adresse) & 15));
  else
    return( RTCRead(Adresse) );                /* C'est le mode binaire */
}

/*****************************************************************/
/* RTCWRITE: écrire une valeur dans une des cellules de la RTC       */
/* Entrée  : Voir plus bas                                       */
/* Sortie  : aucune                                              */
/* Infos   : l'adresse doit être comprise entre 0 et 63          */
/*****************************************************************/

void RTCWrite( byte Adresse, byte Contenu )
{
  (void)outp(RTCAdrPort, Adresse);    /* Communiquer adresse à la RTC */
  (void)outp(RTCDtaPort, Contenu);           /* Ecrire nouvelle valeur */
}

/*****************************************************************/
/**                      PROGRAMME PRINCIPAL                    **/
/*****************************************************************/

void main()
{
  clrscr();

  printf("\nRTC (c) 1987, 92 by Michael Tischer\n\n");
  printf("Informations tirées de l'horloge en temps réel sur piles\n");
```

```
  printf("-------------------------------------------------\n");

  if( !(RTCRead(DIAGNOSTIC) & 128))          /* Piles en bon état ? */
  {                                                  /* Oui */
    printf("- L'horloge est exploitée en mode %d heures\n",
            (RTCRead(ETATB) & 2)*6+12);
    printf("- Il est : %2d:%02d:%02d\n",
             RTCDt(HEURES), RTCDt(MINUTES), RTCDt(SECONDES));
    printf("- Nous sommes le : ");
    printf("%d.%02d.%d%d\n", RTCDt(JOUR), RTCDt(MOIS),
                             RTCDt(SIECLE), RTCDt(ANNEE));
  }
  else
    printf("      ATTENTION ! Les piles de l'horloge sont vides\n");
}
```

12. Les extensions de mémoire

Lorsque l'ancêtre de tous les PC vit le jour, en l'an de grâce 1980 dans les laboratoires de développement d'IBM, ses possibilités étaient encore très en avance sur son temps. Cela valait notamment pour la taille de sa mémoire centrale, tellement élevée, avec ses 640 Ko au maximum, que personne ne savait très bien à quoi elle pourrait servir. Les premiers PC furent livrés avec une mémoire centrale RAM de 64 Ko, plus tard de 128 Ko et enfin de 256 Ko. Si nous avons beaucoup de mal à réprimer une moue condescendante à l'évocation de ces chiffres, c'est entre autres parce qu'une RAM de 640 Ko est entre-temps devenue l'équipement standard des XT, et surtout des AT.

Comme nous entrons maintenant dans l'ère des processeurs i486, des interfaces utilisateur graphiques et des OS multitâche, les 640 Ko sont devenus insuffisants pour exploiter pleinement les potentialités des PC. Mais nous ne pouvons plus régler ce genre de problème en ajoutant juste des cartes mémoire additionnelles. D'autant qu'accéder à plus d'1 Mo avec des processeurs 8086 semble difficile !

L'autre facteur important est la préservation de la compatibilité. Pour que tous les programmes puissent fonctionner du simple XT jusqu'au 486, il faut que les processeurs soient compatibles, mais il faut aussi remplir d'autres conditions primordiales liées notamment à la structure de la mémoire. Rien ne serait compliqué si on se contentait de répondre aux exigences des applications modernes, mais le hic est qu'il s'agit de se conformer au standard défini depuis 1980.

La structure de la mémoire du PC

Bloc	Adresse	Contenu
15	F000:0000 - F000:FFFF	ROM BIOS
14	E000:0000 - E000:FFFF	Libre pour cartouches ROM
13	D000:0000 - D000:FFFF	Libre pour cartouches ROM
12	C000:0000 - C000:FFFF	BIOS ROM supplémentaire
11	B000:0000 - B000:FFFF	RAM Vidéo
10	A000:0000 - A000:FFFF	RAM Vidéo supplémentaire (EGA/VGA)
9	9000:0000 - 9000:FFFF	RAM de 576 Ko à 640 Ko
8	8000:0000 - 8000:FFFF	RAM de 512 Ko à 576 Ko
7	7000:0000 - 7000:FFFF	RAM de 448 Ko à 512 Ko
6	6000:0000 - 6000:FFFF	RAM de 384 Ko à 448 Ko
5	5000:0000 - 5000:FFFF	RAM de 320 Ko à 384 Ko
4	4000:0000 - 4000:FFFF	RAM de 256 Ko à 320 Ko
3	3000:0000 - 3000:FFFF	RAM de 192 Ko à 256 Ko

La structure de la mémoire du PC		
Bloc	**Adresse**	**Contenu**
2	2000:0000 - 2000:FFFF	RAM de 128 Ko à 192 Ko
1	1000:0000 - 1000:FFFF	RAM de 64 Ko à 128 Ko
0	0000:0000 - 0000:FFFF	RAM de 0 Ko à 64 Ko

D'après la figure précédente, l'espace semble plutôt réduit puisque seuls les premiers 640 Ko peuvent être utilisés par la RAM. Tout ce qui suit est réservé pour la RAM vidéo, les extensions électroniques et le BIOS.

Bien qu'à l'heure actuelle, la mémoire puisse facilement être augmentée au-delà de la limite des 1 Mo (les 386 sont souvent livrés avec au moins 2 Mo de mémoire), cette mémoire ne peut toutefois pas être réclamée sous DOS. DOS fonctionne en effet dans le mode réel du processeur qui n'autorise pas l'accès à la mémoire située au-delà des 1 Mo. Le nouveau système d'exploitation OS/2 promet de se libérer de cette contrainte, mais cette réalisation ne peut réellement voir le jour qu'au cours des cinq années à venir.

Aujourd'hui déjà, des solutions sont proposées pour sortir de cette crise, comme nous allons le voir dans le cadre de ce chapitre. A l'heure actuelle, les extensions de mémoire existent sous forme de mémoire étendue (extended memory) et de mémoire paginée (expanded memory). Elles prêtent souvent à confusion en raison de leur similitude phonétique en langue anglaise, mais leurs technologies sont fondamentalement différentes.

Grâce à la multiplicité des Mo la mémoire s'agrandit mais encore faut-il que les logiciels soient en mesure d'en tirer pleinement profit. Les applications professionnelles telles que Lotus 1-2-3 ou Windows 3.x connaissent ce processus depuis longtemps. Ce chapitre explique également comment utiliser cette mémoire dans des programmes personnels.

Mémoire paginée

La mémoire paginée est une mémoire additionnelle que vous trouverez surtout dans les ordinateurs PC/XT. A cause du processeur 8088, ces machines sont limitées à une RAM utilisateur de 640 Ko. Avec une mémoire paginée, la RAM au-dessus de la barrière des 640 Ko peut être utilisée. Rappelez-vous que la mémoire paginée étend la RAM au-dessus des 640 Ko.

Mémoire étendue

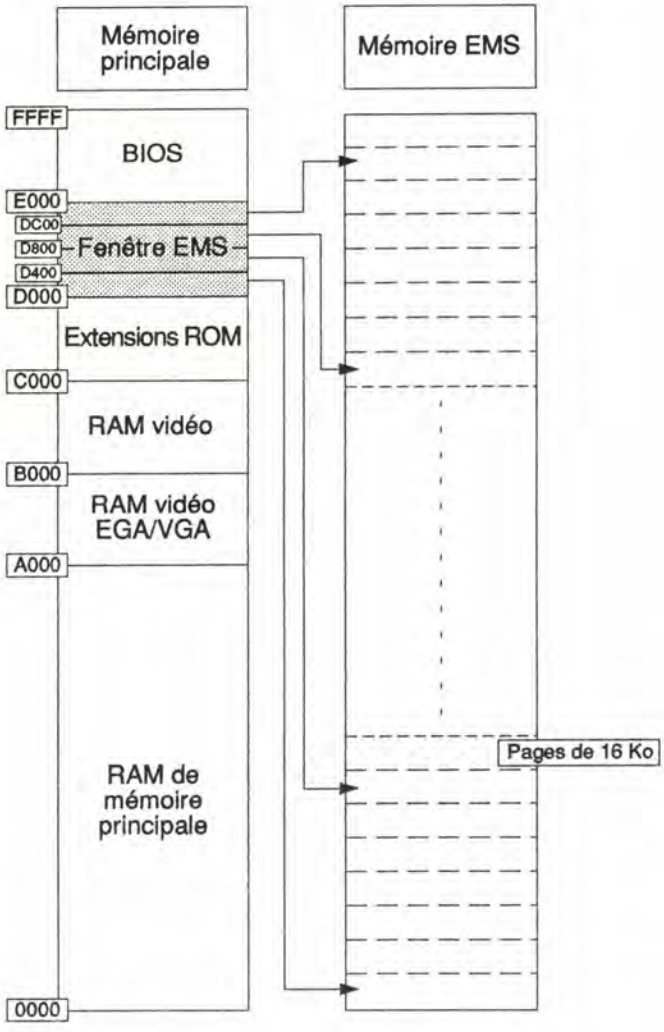
La mémoire étendue est une mémoire additionnelle que vous trouverez dans les machines à base de 80286 ou plus. Elle permet une extension au-dessus du premier méga.

12.1. Mémoire paginée (standard EMS)

Une machine PC ou PC/XT est limitée à 640 Ko de mémoire conventionnelle, les ordinateurs 80286 sont limités à 16 Mo de RAM. Toutefois, ces 16 Mo ne sont disponibles que si le PC tourne en mode protégé, ce qui rend la mémoire inaccessible aux programmes DOS.

Pour remédier à cette situation, plusieurs sociétés en pointe dans le domaine du PC, qui avaient un intérêt commun à ce que même les possesseurs de PC, XT et AT puissent accéder à des capacités RAM plus élevées, et donc aussi à des applications plus complexes et plus puissantes, ont réuni leur efforts. Il s'agissait des sociétés Lotus (les développeurs de Lotus 1-2-3), Microsoft (développeur de MS-DOS et Windows) et Intel (fabricant des processeurs du PC). Elles ont ainsi développé un standard qui a été baptisé standard LIM, d'après les initiales des noms de ces trois sociétés.

Ce standard prévoit que la mémoire centrale du PC peut être étendue à l'aide d'une carte d'extension jusqu'à 32 Mo. Sur ces 32 Mo, seuls 64 Ko au maximum sont visibles à un moment donné, à travers une sorte de fenêtre appelée Page Frame, en dessous de la limite des 1 Mo. Une mémoire installée à l'aide d'une carte d'extension de ce type est appelée "expanded memory" (mémoire paginée) mais ne doit pas être confondue avec "extended memory" (qui veut dire mémoire étendue) au-delà des 1 Mo sur l'AT. Le système ainsi obtenu est appelé "expanded memory system" ou EMS en abrégé.



Accès à la mémoire EMS suivant le standard LIM, à l'aide d'une "fenêtre"

Ouverture d'une fenêtre mémoire

Lors de la réalisation de ce principe, le travail des développeurs a été facilité par le fait que toute la mémoire entre la fin de la mémoire RAM et la limite de 1 Mo n'est pas attribuée au BIOS, à la RAM vidéo et aux autres extensions du système. On peut donc toujours trouver 64 Ko qui n'aient pas encore été attribués et qui peuvent donc être

employés comme une fenêtre ouverte sur la mémoire EMS. Cette fenêtre se situe généralement à l'adresse de segment D000h, mais le matériel EMS est très souple à ce point de vue.

Cette fenêtre étant donc toujours située en dessous de la limite des 1 Mo de mémoire, il est possible, au niveau le plus bas, d'accéder à cette mémoire à l'aide d'instructions assembleur normales, comme par exemple pour la RAM vidéo, qui est également située au-delà de la fin de la mémoire RAM. Sont possibles aussi bien des accès en lecture que des accès en écriture. Nous y reviendrons plus tard en prenant un exemple concret.

Division en Page Frame (cadre de page)

La méthode utilisée est encore affinée par le fait que la mémoire EMS ainsi que le Page Frame ne sont pas divisés en segments de 64 Ko mais en pages de 16 Ko. Le programmeur a ainsi la possibilité, à travers les 4 pages mises en place à l'intérieur du Page Frame, d'accéder à 4 zones différentes de la mémoire EMS, qui peuvent éventuellement être fort éloignées les unes des autres.

Pour qu'on ne soit cependant pas obligé d'appeler sans cesse les mêmes 4 pages, les registres électroniques de la carte EMS peuvent servir à incruster n'importe quelles pages de la mémoire EMS dans le Page Frame. Il s'agit véritablement d'une incrustation au sens propre du terme car les pages ne sont pas copiées de la mémoire EMS dans le Page Frame mais ce sont les canaux d'adresse de la carte EMS qui sont programmés de telle façon que les pages voulues portent effectivement l'adresse de la page correspondante du Page Frame. La mémoire sélectionnée se retrouve donc physiquement à l'intérieur du Page Frame. Cette méthode qu'on appelle Bank Switching rappellera certainement de vieux souvenirs à ceux d'entre vous qui ont bien connu l'époque des ordinateurs familiaux.

En dehors du matériel électronique, EMS comporte aussi une interface logicielle, qui vous décharge par exemple du travail de programmation des registres électroniques EMS ainsi que d'autres tâches de contrôle. Elle s'appelle EMM (expanded memory manager) et constitue une interface standard qui permet d'accéder sans problème aux cartes EMS de différents fabricants et travailler en collaboration avec les innombrables émulateurs EMS qui pullulent déjà sur le marché des ordinateurs 386. Les programmes les plus connus adoptant cette méthode sont 386Max de Qualtias, QEMM386 de Quaterdeck et évidemment Windows 3.x qui propose ses applications dans le mode étendu 386 ainsi que dans la mémoire paginée derrière laquelle se cache en réalité la mémoire étendue.

Tous les produits cités reposent sur un mode d'exploitation spécial du i386, à savoir le mode virtuel 86. Il permet d'intégrer la mémoire située au-delà de la limite des 1 Mo dans la mémoire conventionnelle au-dessous de cette barrière pour qu'elle se présente comme un Page Frame tout à fait normal. Outre une réalisation électronique conventionnelle, il comporte de nombreux avantages :

- ✓ si vous décidez de renoncer à l'émulateur EMS, vous pouvez réutiliser normalement la mémoire étendue,
- ✓ l'accès à la mémoire EMS et la commutation entre les différentes pages s'effectue rapidement parce qu'un matériel devant avoir recours aux ports I/O n'est pas impliqué,
- ✓ les coûts ne sont pas élevés parce qu'en général une carte EMS est plus chère qu'un émulateur logiciel, et une insuffisance, tout aussi équivalente en mémoire étendue,
- ✓ vous économisez de la place.

Comme nous l'avons déjà mentionné, l'émulation EMS ne concerne que les i386, i486 et les modèles suivants, les ordinateurs AT n'ont pas droit à cette ressource. Dans ce cas également, il existe de nombreuses techniques d'émulation de la mémoire EMS qui dépendent généralement de la vitesse d'accès à la mémoire.

Les fabricants d'ordinateurs AT offrent cependant une technique permettant d'utiliser la mémoire étendue comme mémoire EMS. Ils adoptent en général une électronique découverte par la société Chips & Technologies appelée Neat Chips (NEAT = New Enhanced AT). Sur le plan électronique, ils prévoient de configurer la mémoire étendue sur la mémoire paginée sans qu'il soit nécessaire d'installer une carte d'extension spéciale à cet effet. Celle-ci se trouve d'ores et déjà sur la carte mère et peut être configurée selon les besoins au moyen d'un programme Setup étendu.

L'interface logicielle entre l'EMM et un programme est identique aux autres interfaces logicielles que l'on rencontre dans le monde PC pour la simple raison que les diverses fonctions EMM doivent être appelées à travers une interruption logicielle. Pour de nombreuses interfaces logicielles, la fabrication de l'interface EMS est également typique puisqu'elle a été étendue et modifiée à plusieurs reprises.

12.1.1. L'histoire du standard LIM

Pendant que la plupart des programmes logiciels voient le jour avec le numéro de version 1.0, une version 1.0 du standard EMS n'a jamais existé. Une telle version n'a du moins pas quitté le laboratoire de développement car pendant que Lotus et Intel envisageaient de donner leur accord pour un standard EMS au début de 1985, on voyait déjà apparaître la version 3.0. Jusqu'à ce moment donné, Microsoft était exclu de la partie mais entrait rapidement dans le circuit parce qu'on peut parfaitement se servir de cette technique pour étendre la mémoire dans ses propres produits.

EMS Version 3.2

La version 3.0 a rapidement donné naissance à la version 3.2 et le standard EMS au standard LIM EMS. Cela s'est passé en automne 1985 et quelques semaines plus tard, un consortium regroupant les sociétés AST, Quadram et Ashton Tate présentait leur standard EMS personnel, à savoir EEMS (Enhanced Expanded Memory Specification). Cette mémoire était évidemment basée sur la version EMS 3.2 sans pour autant présenter des avantages intéressants. Par conséquent, elle n'a pas trouvé sa place sur le marché. Aujourd'hui, elle est tombée dans l'oubli, mais autrefois, elle pouvait utiliser des zones supérieures à 64 Ko en tant que Page Frame - une condition préalable importante pour les programmes multitâches de DOS comme DESQview. Les mêmes capacités sont toutefois reconnues par EMS 4.0 si bien qu'aujourd'hui personne ne parle plus d'EEMS.

EMS 3.2 a connu très vite un grand succès et est soutenue par la plupart des fabricants renommés. Lotus a pris les devants en promouvant les cartes EMS. Grâce à l'EMS, il devient désormais possible de calculer des feuilles de calcul volumineuses avec le bestseller d'antan Lotus 1-2-3. Mais les programmes TSR, les disques RAM et de nombreux utilitaires commencèrent très vite à utiliser la mémoire EMS à des fins personnelles. En tout cas, Microsoft n'a pas tardé à soutenir la mémoire EMS depuis la version 4.0 à travers MS-DOS et ce, lamentablement et de manière plutôt erronée dans les débuts.

EMS Version 4.0

Alors que la version EMS 3.2 pouvait répondre à tous les besoins du programmeur avec ses 14 fonctions variées, la version 4.0 est apparue en automne 1987 avec 58 fonctions. Elles sont nécessaires pour soutenir l'électronique étendue parce que la version 4.0 reconnaît jusqu'à 32 Mo au lieu de 8 Mo. En outre, la fenêtre EMS peut être définie à n'importe quel endroit de la mémoire. Etant donné que sa taille est variable, l'environnement de gestion de la mémoire EMS a considérablement augmenté.

Ainsi, la version 4.0 n'exige pas de l'électronique EMS de remplir des conditions spectaculaires qui ne pouvaient pas d'ailleurs pas être remplies par les anciennes cartes EMS. En quelque sorte, c'est la raison pour laquelle la version 4.0 n'a pas pu prendre le pas sur la version 3.2 qui représente depuis toujours le standard pour la programmation EMS.

Cependant, il est regrettable qu'une performance fasse défaut. En effet, il n'est pas possible de protéger les pages EMS contre la suppression lors d'un démarrage à chaud. En fait, les disques RAM ont perdu l'habitude de conserver leur contenu à la suite d'un plantage ou d'une réinitialisation. Pourtant, cela faciliterait nettement le travail.

12.1.2. EMS version 3.2

Ce chapitre traite de l'accès concret à la mémoire EMS à l'aide des diverses fonctions définies à travers le gestionnaire de mémoire paginée (EMM). Comme vous l'avez appris dans les sections précédentes, leur nombre s'est accru considérablement entre la version 3.2 et la version 4.0. Toujours est-il que les fonctions convenant parfaitement dans l'exploitation de la mémoire EMS ne sont pas légion.

Toutes les fonctions existent déjà dans la version EMS 3.2 et c'est d'ailleurs la raison pour laquelle cette version occupe la principale place dans ce chapitre. Les fonctions de la version 4.0 sont toutefois décrites en détail en annexe, mais elles ne jouent aucun rôle dans ce contexte. Rares sont les fabricants de cartes mémoire qui reconnaissent la version 4.0. Ses fonctions ne sont même pas sollicitées dans les programmes normaux.

EMM

Suivant le même principe que pour l'interruption DOS 21h, qui fait office d'interface avec les fonctions du système d'exploitation, les fonctions de l'EMM peuvent être appelées à travers l'interruption 67h. Avant qu'un programme suppose, de façon inconsidérée, qu'une mémoire EMS et une EMM sont installées, il serait toutefois préférable qu'il vérifie qu'il en est bien ainsi. S'il ne le fait pas et si aucune mémoire EMS n'est installée, les conséquences d'un appel de l'interruption 67h sont imprévisibles. Il se peut que rien ne se passe, mais il se peut aussi que le système soit planté.

Pour éviter cela, tout programme souhaitant avoir recours à la mémoire EMS devrait donc vérifier qu'elle est bien disponible. On peut tirer parti pour cela du fait que l'EMM est intégrée dans le système comme un driver de périphérique tout à fait normal lors du lancement du système (c'est-à-dire avec une instruction appropriée dans le fichier de configuration CONFIG.SYS). Elle possède donc un en-tête de driver précédant le driver dans la mémoire, et dont la structure est imposée par le DOS. Comme il s'agit d'un driver de caractère, le nom du driver figure à partir de l'adresse 10 de l'en-tête de driver. Le standard LIM prévoit que ce nom doit être EMMXXXX0. Les programmes d'exemple à la fin de ce chapitre recherchent ce nom après avoir calculé l'adresse de segment de l'interruption 67h. S'il s'agit de l'EMM, l'adresse de segment doit correspondre au segment dans lequel le driver de périphérique EMMXXXX0 (c'est-à-dire l'EMM) a été chargé. Comme l'en-tête de périphérique figure à l'adresse d'offset 0 par rapport au début de ce segment, il ne reste plus qu'à comparer le contenu des cellules de mémoire 10 avec le nom EMMXXXX0 pour savoir si une mémoire EMS, avec l'EMM correspondante, est installée.

Une fois que le test a permis de s'assurer qu'une mémoire EMS était bien disponible, l'accès à cette mémoire se déroule essentiellement en trois étapes.

① De même que la mémoire centrale normale doit être allouée à travers une fonction DOS, un programme doit aussi se faire allouer par l'EMM un certain nombre de

pages EMS. Le nombre de pages allouées ne dépend bien sûr pas seulement des besoins en mémoire du programme mais aussi de la quantité de mémoire EMS installée et n'ayant pas encore été attribuée à d'autres programmes.

② Si le programme a pu se faire allouer le nombre de pages qu'il souhaitait, les pages appelées doivent tout d'abord être amenées dans l'une des quatre pages du Page Frame avant que le programme puisse y transférer des données ou lire leur contenu. Il en résulte une superposition de l'une des pages logiques allouées sur l'une des quatre pages physiques du Page Frame, qu'on désigne par le terme de Mapping.

③ Lorsque le programme ou le travail avec la mémoire EMS est terminé, il convient de libérer à nouveau les pages allouées. Si ce n'est pas fait, les pages allouées resteront propriété du programme (même après qu'il se soit terminé) et ne pourront être attribuées à d'autres programmes.

Ces trois étapes caractérisent l'appel de nombreuses fonctions de l'EMM que nous allons vous présenter dans les pages suivantes. Comme pour l'appel de l'interruption DOS 21h, le numéro de fonction doit chaque fois être chargé dans le registre AH. Contrairement aux fonctions DOS, le numéro de fonction ne correspond pas ici à la valeur du registre AH mais doit encore être additionné à la valeur 3Fh. Pour appeler la fonction 2h, il faudra donc charger la valeur 3Fh + 2h ou 41h dans le registre AH. Après appel de la fonction, ce registre contient le statut d'erreur de la fonction appelée, la valeur 0 indiquant que la fonction s'est conclue par un succès alors que les valeurs supérieures ou égales à 80h signalent une erreur.

A propos des erreurs

Les différents codes d'erreur sont indiqués dans la description des fonctions que vous trouverez en annexe de cet ouvrage. Signalons simplement ici une erreur particulière : si la valeur 84h figure dans le registre AH après appel de l'interruption EMM 67h, cela indique qu'un numéro de fonction incorrect avait été transmis dans le registre AH lors de l'appel de l'interruption, c'est-à-dire que la fonction appelée n'existe pas.

Deux autres erreurs résultant d'un défaut de fonctionnement de l'EMM ou de l'électronique EMS peuvent apparaître à la suite de pratiquement n'importe quel appel de fonction : il s'agit des codes d'erreur 80h et 81h.

Fonctions EMS

Voici les fonctions dont a besoin un programme transitoire pour accéder à la mémoire EMS :

Fonction	Utilisation
40H (01h)	Lire l'état EMM
41H (02h)	Lire l'adresse de segment du Page Frame
42H (03h)	Lire le nombre de pages
43H (04h)	Allouer des pages EMS
44H (05h)	Fixer Mapping
45H (06h)	Libérer des pages EMS

Pour garantir un fonctionnement impeccable de l'électronique EMS et de l'EMM, il est conseillé de procéder à un test de l'état EMM, à l'aide de la fonction 40h, qui n'attend pas d'autre paramètre que le numéro de fonction dans le registre AH. Si elle renvoie la valeur 0 dans le registre AH, c'est que tout va bien et le travail avec la mémoire EMS peut commencer.

Limites de l'allocation EMS

Le nombre de pages EMS pouvant être allouées est naturellement limité vers le haut par le nombre de pages encore libres. C'est pourquoi il convient, avant de procéder à l'allocation, de vérifier que les besoins en mémoire du programme n'excèdent pas la capacité encore disponible. On peut avoir recours à cet effet à la fonction 42h qui renvoie le nombre de pages non encore allouées. Cette fonction n'attend pas non plus d'autre paramètre que le numéro de fonction. Elle renvoie dans le registre BX le nombre de pages non encore allouées. Elle fournit également dans le registre DX le nombre total de pages EMS installées, une information qui peut être intéressante pour un rapport d'état et que nous avons d'ailleurs utilisée dans nos programmes d'exemple.

Si la mémoire EMS disponible permet de couvrir les besoins du programme ou si ces besoins ont été adaptés à la quantité de mémoire libre, la mémoire peut enfin être allouée. La fonction 43h doit recevoir pour cela le numéro de fonction ainsi que le nombre de pages à allouer dans le registre BX. Si le nombre de pages réclamé a pu être alloué sans problème (0 dans le registre AH après appel de la fonction), le programme d'appel trouve dans le registre BX un Handle qui lui servira, lors des appels ultérieurs, de clé d'accès aux pages allouées et qui permettra à l'EMM d'identifier celui qui appelle. Ce Handle doit être stocké dans une variable par le programme d'appel car sa "perte" lui interdirait non seulement d'accéder aux pages allouées mais même de libérer ces pages au profit d'autres programmes. Cette fonction peut d'ailleurs être appelée à plusieurs reprises par un programme, lorsqu'il s'agit d'allouer plusieurs blocs logiques.

Avec le handle ainsi obtenu, l'accès aux pages peut commencer. Pour toutes les fonctions, le handle doit toujours être transmis dans le registre DX. Cela vaut aussi pour la fonction 44h, qui sert à placer une page logique dans l'une des quatre pages physiques du Page Frame. Le numéro de la page logique doit pour cela être transmis dans le

registre BX, celui de la page physique dans AL. Notez bien que ces deux valeurs sont comptées à partir de 0. Si vous avez par exemple alloué 15 pages, le numéro de page logique devra donc être compris entre 0 et 14 inclus.

Une fois la page appelée placée dans le Page Frame, elle peut être appelée exactement comme la mémoire normale. Si l'adresse d'offset du début de la page peut être calculée d'après le numéro de la page physique, l'adresse de segment correspondante doit tout d'abord être obtenue à l'aide d'une fonction EMM. Cette adresse ne peut cependant être modifiée au cours du travail avec la mémoire EMS, de sorte qu'il suffit de la rechercher une seule fois au cours de l'initialisation du programme, puis de la stocker dans une variable. C'est la fonction 41H qui renvoie dans le registre BX l'adresse de segment du Page Frame.

Tout travail avec la mémoire EMS doit se terminer par la restitution des pages allouées à l'EMM. La fonction 45H doit simplement recevoir pour cela le handle qui avait été obtenu auparavant.

Outre ces six fonctions auxquelles un programme normal peut ou doit avoir recours pour accéder à la mémoire EMS, il existe six autres fonctions qui peuvent être utiles aux programmes dans certaines situations déterminées. Il s'agit des fonctions suivantes :

Fonction	Utilisation
46H (07h)	Lire le numéro de version EMM.
47H (08h)	Sauver Mapping actuel.
48H (09h)	Rétablir Mapping sauvegardé.
49H (0Ah)	Lire le nombre de handles EMM.
4AH (0Bh)	Lire le nombre de pages allouées à un Handle.
4BH (0Ch)	Lire tous les handles et le nombre de pages allouées.

Numéros de version

Le test du numéro de version EMM n'est pas dépourvu d'intérêt dans la mesure où le standard LIM a naturellement continué d'évoluer depuis son apparition. Cela signifie que certaines fonctions ont disparu, et ne sont donc plus soutenues dans les versions suivantes, alors que d'autres fonctions ont été ajoutées. Les fonctions présentées ici se réfèrent à la version 3.2, qui a entre-temps été suivie d'une version 4.0. La version 3.2 n'est cependant pas dépassée pour autant. Elle représente en fait un bon compromis car elle est très répandue et de plus totalement compatible avec la version 4.0. Si vous voulez malgré tout renoncer à soutenir certaines versions EMS antérieures ou ultérieures dans votre programme, il faudrait que vous recherchiez le numéro de version de l'EMM avant le début du programme. Après appel de la fonction 46H, ce numéro de version est renvoyé dans le registre AL. Il y est codé sous forme d'un nombre BCD, les 4 bits

de plus fort poids contenant le numéro de version principale et les 4 bits de plus faible poids le numéro de sous-version.

Les fonctions 47H et 48H sont importantes pour les programmes TSR souhaitant utiliser la mémoire EMS. Lorsqu'un programme TSR interrompt l'exécution d'un programme transitoire pour passer au premier plan pendant un certain temps, il doit tenir compte du fait que le programme interrompu accède peut-être lui-même à la mémoire EMS et qu'un Mapping déterminé a donc pu être mis en place. Il ne faut naturellement pas que ce Mapping ait été modifié lorsque reprendra l'exécution du programme interrompu. Ce Mapping doit donc être sauvegardé lors de l'activation du programme TSR pour pouvoir être rétabli lorsque ce programme se terminera. C'est à cela que servent les fonctions 47H et 48H. La fonction 47H sauvegarde le Mapping actuel à l'intérieur de l'EMM, alors que la fonction 48H rétablit l'état sauvegardé. Les deux fonctions doivent recevoir le handle du programme d'appel. Il ne s'agit donc pas, en l'occurrence, du handle du programme interrompu, sous lequel le Mapping avait été fixé, mais bien du handle du programme TSR.

Les trois dernières fonctions de cette liste n'ont d'intérêt que pour les contrôleurs de mémoire chargés, au niveau le plus bas, de répartir la mémoire au modules de rang hiérarchique plus élevé. Il n'y a donc pas lieu de décrire ces fonctions en détail. Vous trouverez de plus amples informations sur ces fonctions dans l'annexe consacrée à la description des fonctions de l'EMM.

Programmes de démonstration

Pour vous aider à appliquer dans la pratique les informations fournies dans ce chapitre, vous trouverez dans les pages suivantes deux programmes en C et en Pascal pour le travail avec la mémoire EMS. Il n'est pas besoin ici de programme assembleur car les appels des fonctions EMM consistent au fond simplement à charger des variables et constantes dans des registres et à appeler l'interruption EMM 67h. A l'aide de la description des fonctions figurant en annexe, un programmeur en assembleur ne devrait avoir aucun mal à réaliser lui-même les appels de fonction nécessaires. Nous ne vous proposons pas non plus de programme BASIC car la mémoire EMS ne peut être utilement employée qu'avec des applications complexes et nécessitant beaucoup de mémoire. Or ce genre d'applications ne peuvent de toute façon être réalisées en BASIC (ou plus précisément en GW-BASIC).

Les deux programmes présentés sont identiques pour l'essentiel, de sorte que nous pouvons nous limiter ici à une description de la structure fondamentale de ces programmes. Ils offrent une série de fonctions et de procédures permettant d'appeler les différentes fonctions EMM. Ces deux programmes comportent en outre une fonction EMS_INST (ou EmsInst) qui détermine tout d'abord, d'après la méthode décrite plus haut, si une EMM est installée. Il est particulièrement important qu'il s'agisse d'un pointeur FAR car le Page Frame se situe en dehors du segment de données du programme et ne peut donc être adressé à travers le contenu du registre DS. Cela ne

pose pas de problème en Pascal puisque le code généré pour les références aux données travaille systématiquement avec des pointeurs FAR. En C, par contre, il faut veiller à ce que le programme soit compilé dans l'un des modèles de mémoire travaillant avec des pointeurs FAR pour les références des données. C'est le cas des modèles de mémoire Compact, Large et Huge.

Le programme principal teste tout d'abord si une EMM est installée et recherche alors, à l'aide de différentes fonctions, les informations d'état sur la mémoire EMS, pour les afficher sur l'écran. Une page est ensuite allouée et plaquée sur la première page (la page 0) du Page Frame. Le contenu actuel de la RAM vidéo y est alors copié avant d'être annulé. On se rend compte ici qu'il est très pratique de travailler avec des pointeurs FAR car cela permet d'employer les fonctions de bibliothèque (en C) ou les instructions tout à fait normales pour accéder à la mémoire EMS. Il s'agit en l'occurrence des instructions MOVE (Pascal) et MEMCPY (C).

Après l'opération de copie, un message est affiché pour l'utilisateur et on attend qu'une touche soit actionnée. La seconde opération de copie est alors déclenchée, qui consiste à recopier de la page 0 du Page Frame vers la RAM vidéo le contenu de la RAM vidéo qui avait été sauvegardé auparavant. Le programme est alors terminé.

L'intérêt principal de ce programme est de bien montrer que le contenu d'une page du Page Frame peut être traité exactement comme des données normales que vous auriez placées dans des variables globales, dans une variable sur le Heap ou dans une zone de mémoire allouée à travers le DOS.

Une fois qu'un pointeur désignant la page voulue dans le Page Frame a été mis en place, vous pouvez effectuer des opérations de variables tout à fait normales, et donc aussi travailler avec des objets complexes tels que des structures et vecteurs ou tableaux. Il importe seulement de veiller à ce que vos objets tiennent dans une page ou alors de ne pas oublier de changer de page ou d'amener la page requise dans le Page Frame si, pour des objets de taille importante, l'adresse d'offset par rapport au début de la page dépasse 16 Ko.

Listing : EMMC.C

```
/*****************************************************************
*                              EMMC                             *
******------------------------------------------------------******
*   Fonction       : Fournit quelques fonctions pour l'accès à la *
*                    mémoire EMS (Expanded Memory).               *
******------------------------------------------------------******
*   Auteur         : MICHAEL TISCHER                            *
*   Développé le   : 30/08/1988                                 *
*   Dernière modif.: 30/03/1992                                 *
******------------------------------------------------------******
*   (MICROSOFT C)                                               *
*   Création       : CL /AC EMMC.C                              *
*                    : LINK EMMC;                               *
*   Appel          : EMMC                                       *
******------------------------------------------------------******
*   (BORLAND TURBO C)                                           *
*   Création       : Avec l'instruction RUN dans la ligne de menu *
*                    (sans fichier Project)                     *
*   Infos          : Notez bien que le modèle de mémoire Compact *
*                    doit être sélectionné avec l'instruction   *
*                              Option-Compiler-Model            *
*****************************************************************/

/*--- Intégrer les fichiers Include ---------------------------*/

#include <stdio.h>
#include <conio.h>
#include <dos.h>
#include <stdlib.h>
#include <string.h>

/*--- Typedefs ------------------------------------------------*/

typedef unsigned char BYTE;          /* Nous nous bricolons un octet */
typedef unsigned int  WORD;
typedef BYTE          BOOL;          /* Comme BOOLEAN en Pascal */

/*--- Macros --------------------------------------------------*/

/*-- MK_FP compose un pointeur FAR sur un objet à partir d'une -------*/
```

```
/*-- adresse de segment et d'une adresse d'offset ------*/

#ifdef MK_FP                          /* MK_FP à déjà été défini ? */
  #undef MK_FP
#endif
#define MK_FP(seg, ofs) ((void far *)(((unsigned long)(seg)<<16)|(ofs)))

/*-- PAGE_ADR fournit un pointeur sur la page physique X ----------*/
/*-- à l'intérieur du Page Frame de la mémoire EMS ----------*/

#define PAGE_ADR(x) ((void *) (MK_FP(ems_frame_seg() + ((x)<<10), 0)))

/*-- Constantes ---------------------------------------------------*/

#define TRUE    ( 0 == 0 )    /* Constantes pour travailler avec BOOL */
#define FALSE   ( 1 == 0 )

#define EMS_INT 0x67          /* N° d'interruption pour l'accès à l'EMM */
#define EMS_ERR -1                    /* Est renvoyé en cas d'erreur */

/*-- Variables globales -------------------------------------------*/

BYTE emm_ec;                  /* Ici sont placés les codes d'erreur EMM */

/*****************************************************************
*  Fonction         : E M S _ I N S T                             *
*----------------------------------------------------------------*
*  Fonction         : Détermine si une mémoire EMS et un driver EMS *
*                     correspondant (EMM) sont installés.         *
*  Param. en entrée : Aucun                                       *
*  Valeur Return    : TRUE si mémoire EMS installée, sinon        *
*                     FALSE.                                      *
*****************************************************************/

BOOL ems_inst( void )
{
  static char   emm_name[] = { 'E', 'M', 'M', 'X', 'X', 'X', 'X', '0' };
  union REGS    regs; /* Registres processeur pour appel interruption */
  struct SREGS  sregs;  /* Registre segment pour l'appel interruption */

   /* Mettre en place pointeur sur nom dans en-tête driver de périph.*/

  regs.x.ax = 0x3567;   /* N° fct.: Rechercher vecteur interrupt 0x67 */
  intdosx( &regs, &regs, &sregs );  /* Appeler interruption DOS 0x21 */

  return (!memcmp( MK_FP(sregs.es, 10), emm_name, sizeof emm_name ));
                                              /*TRUE si nom trouvé */
}

/*****************************************************************
*  Fonction         : E M S _ N U M _ P A G E                     *
*----------------------------------------------------------------*
*  Fonction         : Détermine le nombre total de pages EMS.     *
*  Param. en entrée : Aucun                                       *
*  Valeur Return    : EMS_ERR en cas d'erreur, sinon le nombre de *
*                     pages EMS.                                  *
*****************************************************************/

int ems_num_page( void )
{
  union REGS   regs;              /* Reg. proc. pour appel interrupt. */

  regs.h.ah = 0x42;             /* N° fct.: Déterminer nombre de pages */
  int86( EMS_INT, &regs, &regs );                 /* Appeler EMM */
  if( (int)(emm_ec = regs.h.ah) )    /* Une erreur est-elle apparue ? */
    return( EMS_ERR );                          /* OUI, afficher erreur */
  else                                          /* Pas d'erreur */
    return( regs.x.dx );          /* Renvoyer le nombre total de pages */
}

/*****************************************************************
*  Fonction         : E M S _ F R E E _ P A G E                   *
*----------------------------------------------------------------*
*  Fonction         : Détermine le nombre de pages EMS encore libres. *
*  Param. en entrée : Aucun                                       *
*  Valeur Return    : EMS_ERR en cas d'erreur, sinon le nombre de *
*                     pages EMS libres.                           *
*****************************************************************/

int ems_free_page( void )
{
  union REGS  regs;   /* Registres processeur pour appel interruption */

  regs.h.ah = 0x42;             /* N° fct.: Déterminer nombre de pages */
  int86( EMS_INT, &regs, &regs );                 /* Appeler EMM */
  if((int)(emm_ec = regs.h.ah))       /* Une erreur est-elle apparue ? */
    return( EMS_ERR );                          /* OUI, afficher erreur */
  else                                          /* Pas d'erreur */
    return( regs.x.bx );          /* Renvoyer nombre de pages libres */
}

/*****************************************************************
*  Fonction         : E M S _ F R A M E _ S E G                   *
*----------------------------------------------------------------*
*  Fonction         : Détermine l'adresse de segment du Page Frame EMS *
*  Param. en entrée : Aucun                                       *
*  Valeur Return    : EMS_ERR en cas d'erreur, sinon adresse segment *
*                     du Page Frame.                              *
*****************************************************************/

WORD ems_frame_seg( void )
{
  union REGS  regs;

  regs.h.ah = 0x41;             /* N° fct.: adresse segment Page Frame */
  int86( EMS_INT, &regs, &regs );                 /* Appeler EMM */
  if( (int)(emm_ec = regs.h.ah) )    /* Une erreur est-elle apparue ? */
    return( EMS_ERR );                          /* OUI, afficher erreur */
  else                                          /* Pas d'erreur */
    return( regs.x.bx );          /* Renvoyer adresse de segment */
}

/*****************************************************************
*  Fonction         : E M S _ A L L O C                           *
*----------------------------------------------------------------*
*  Fonction         : Alloue le nombre de pages spécifié et renvoie un *
*                     Handle pour l'accès à ces pages.            *
*  Param. en entrée : PAGES : Nombre de pages à allouer           *
*                     (de 16 Ko chacune)                          *
*  Valeur Return    : EMS_ERR en cas d'erreur, sinon le Handle EMS. *
*****************************************************************/

int ems_alloc(int pages)
{
  union REGS  regs;

  regs.h.ah = 0x43;                         /* N° fct.: Allouer pages */
  regs.x.bx = pages;              /* Fixer nombre de pages à allouer */
  int86( EMS_INT, &regs, &regs );                 /* Appeler EMM */
  if( (int)(emm_ec = regs.h.ah) )    /* Une erreur est-elle apparue ? */
    return(EMS_ERR);                            /* OUI, afficher erreur */
  else                                          /* Pas d'erreur */
    return( regs.x.dx );                        /* Renvoyer Handle EMS */
}

/*****************************************************************
*  Fonction         : E M S _ M A P                               *
*----------------------------------------------------------------*
*  Fonction         : Calque une des pages logiques allouées sous le *
*                     handle transmis sur la page physique du     *
*                     Page Frame.                                 *
*  Param. en entrée : HANDLE: Le Handle renvoyé par EMS_ALLOC.    *
*                     LOGP  : La page logique (0 à n-1)           *
*                     PHYSP : La page physique (0 à 3)            *
*  Valeur Return    : FALSE en cas d'erreur, sinon TRUE.          *
*****************************************************************/

BOOL ems_map(int handle, int logp, BYTE physp)
{
  union REGS  regs;

  regs.h.ah = 0x44;                       /* N° fct.: Fixer Mapping */
  regs.h.al = physp;                        /* Fixer page physique */
  regs.x.bx = logp;                         /* Fixer page logique */
  regs.x.dx = handle;                       /* Fixer Handle EMS */
  int86( EMS_INT, &regs, &regs );                 /* Appeler EMM */
  return( !(emm_ec = regs.h.ah) );
}

/*****************************************************************
*  Fonction         : E M S _ F R E E                             *
*----------------------------------------------------------------*
*  Fonction         : Libère à nouveau la mémoire allouée sous un *
*                     Handle.                                     *
*  Param. en entrée : HANDLE: le Handle renvoyé par EMS_ALLOC.    *
*  Valeur Return    : FALSE en cas d'erreur, sinon TRUE.          *
*****************************************************************/

BOOL ems_free(int handle)
{
  union REGS  regs;

  regs.h.ah = 0x45;                       /* N° fct.: Libérer pages */
  regs.x.dx = handle;                       /* Fixer Handle EMS */
  int86( EMS_INT, &regs, &regs );                 /* Appeler EMM */
  return( !(emm_ec = regs.h.ah) ); /* Si AH contient 0, tout va bien. */
}

/*****************************************************************
*  Fonction         : E M S _ V E R S I O N                       *
*----------------------------------------------------------------*
*  Fonction         : Détermine le numéro de version EMM.         *
*  Param. en entrée : Aucun                                       *
*  Valeur Return    : EMS_ERR en cas d'erreur, sinon le numéro de *
*                     version EMM.                                *
*  Infos            : Pour le n° de version, 10 signifie 1.0, 11  *
*                     signifie 1.1, 34 signifie 3.4 etc.          *
*****************************************************************/
```

```
|
|BYTE ems_version( void )
|{
|  union REGS  regs;
|
|  regs.h.ah = 0x46;                  /* N° fct.: Déterminer version EMM */
|  int86( EMS_INT, &regs, &regs );                    /* Appeler EMM */
|  if( (int)(emm_ec = regs.h.ah) )    /* Une erreur est-elle apparue ? */
|    return( EMS_ERR );                              /* OUI, afficher erreur */
|  else          /* Pas d'erreur, n° version à partir du nombre BCD */
|    return( (regs.h.al & 15) + (regs.h.al >> 4) * 10 );
|}
|
|/*****************************************************************
|*  Fonction         : E M S _ S A V E _ M A P                   *
|**---------------------------------------------------------------**
|*  Fonction         : Sauvegarde le calquage (Mapping) entre pages *
|*                     logiques et physiques.                     *
|*  Param. en entrée : HANDLE: le handle renvoyé par EMS_ALLOC.   *
|*  Valeur Return    : FALSE en cas d'erreur, sinon TRUE.         *
|*****************************************************************/
|
|BOOL ems_save_map(int handle)
|{
|  union REGS  regs;
|
|  regs.h.ah = 0x47;                      /* N° fct.: Sauvegarder Mapping */
|  regs.x.dx = handle;                                /* Fixer Handle EMS */
|  int86( EMS_INT, &regs, &regs );                    /* Appeler EMM */
|  return (!(emm_ec = regs.h.ah));  /* Si AH contient 0, tout va bien. */
|}
|
|/*****************************************************************
|*  Fonction         : E M S _ R E S T O R E _ M A P             *
|**---------------------------------------------------------------**
|*  Fonction         : Rétablit un calquage entre pages logiques et *
|*                     physiques sauvegardé préalablement avec    *
|*                     EMS_SAVE_MAP.                              *
|*  Param. en entrée : HANDLE: le handle renvoyé par EMS_ALLOC.   *
|*  Valeur Return    : FALSE en cas d'erreur, sinon TRUE.         *
|*****************************************************************/
|BOOL ems_restore_map(int handle)
|{
|  union REGS  regs;
|
|  regs.h.ah = 0x48;                      /* N° fct.: Rétablir Mapping */
|  regs.x.dx = handle;                                /* Fixer Handle EMS */
|  int86( EMS_INT, &regs, &regs );                    /* Appeler EMM */
|  return( !(emm_ec = regs.h.ah) ); /* Si AH contient 0, tout va bien. */
|}
|
|/*****************************************************************
|*  Fonction         : P R I N T _ E R R                         *
|**---------------------------------------------------------------**
|*  Fonction         : Sort un message d'erreur EMS sur l'écran et *
|*                     termine le programme.                      *
|*  Param. en entrée : Aucun                                      *
|*  Valeur Return    : Aucune                                     *
|*  Infos            : Cette fonction ne doit être appelée que si une *
|*                     erreur s'est produite lors d'un appel antérieur *
|*                     d'une fonction de l'EMM.                   *
|*****************************************************************/
|
|void print_err( void )
|{
|  static char    nid[] = "non identifiable";
|  static char    *err_vec[] =
|  {
|    "Erreur dans le driver EMS (EMM détruit)",              /* 0x80 */
|    "Erreur dans l'électronique EMS",                       /* 0x81 */
|     nid,                                                   /* 0x82 */
|    "Handle EMM incorrect",                                 /* 0x83 */
|    "Fonction EMS appelée n'existe pas",                    /* 0x84 */
|    "Plus de handles EMS disponibles",                      /* 0x85 */
|    "Erreur de sauvegarde ou de reconstitution du Mapping", /* 0x86 */
|    "Plus de pages réclamées qu'il n'en existe physiquement", /* 0x87 */
|    "Plus de pages réclamées qu'il n'en reste de libres",   /* 0x88 */
|    "Zéro page réclamée",                                   /* 0x89 */
|    "Page logique ne correspondant pas au Handle",          /* 0x8A */
|    "Numéro de page physique incorrect",                    /* 0x8B */
|    "Zone de mémoire Mapping pleine",                       /* 0x8C */
|    "Sauvegarde du Mapping déjà effectuée",                 /* 0x8D */
|    "Reconstitution du Mapping sans sauvegarde antérieure"
|  };
|
|  printf("\nATTENTION ! Erreur lors de l'accès à la mémoire EMS\n");
|  printf("          ... %s\n", (emm_ec<0x80 || emm_ec>0x8E)
|                              ? nid
|                              : err_vec[emm_ec-0x80]);
|  exit( 1 );                  /* Terminer programme avec code d'erreur */
|}
|
|/*****************************************************************
|*  Fonction         :  V R _ A D R                              *
|**---------------------------------------------------------------**
|*  Fonction         : Fournit un pointeur sur la RAM vidéo       *
|*  Entrée           : Aucune                                     *
|*  Sortie           : Pointeur sur la RAM vidéo                  *
|*****************************************************************/
|
|void *vr_adr( void )
|{
|  union REGS  regs;
|
|  regs.h.ah = 0x0f;                  /* N° fct.: Déterminer mode vidéo */
|  int86(0x10, &regs, &regs);    /* Appeler interruption vidéo du BIOS */
|  return( MK_FP(((regs.h.al==7) ? 0xb000 : 0xb800), 0) );
|}
|
|/*****************************************************************/
|/**                      PROGRAMME PRINCIPAL                    **/
|/*****************************************************************/
|
|void main()
|{
|  int  nmbpage,                                  /* Nombre de pages EMS */
|       handle,                     /* Handle pour l'accès à la mémoire EMS */
|       i;                                        /* Compteur de boucle */
|  WORD pageseg;                    /* Adresse de segment du Page Frame */
|  BYTE emmver;                             /* Numéro de version de l'EMM */
|
|  printf("EMMC  -  (c) 1988, 92 by MICHAEL TISCHER\n\n");
|  if( ems_inst() )                          /* Mémoire EMS installée ? */
|  {                                                          /* Oui */
|    /*-- Sortir des informations sur la mémoire EMS ----------------*/
|    if( (int)(emmver = ems_version()) == EMS_ERR )     /* n° de version */
|      print_err();      /* Erreur : Sortir message d'erreur et terminer */
|    else                                                /* Pas d'erreur */
|      printf("Numéro de version EMM         : %d.%d\n",
|             emmver/10, emmver%10);
|
|    if( (nmbpage = ems_num_page()) == EMS_ERR)        /* nmb. de pages */
|      print_err();             /* Erreur : message d'erreur et terminer */
|    else
|      printf("Nombre de pages EMS           : %d (%d Ko)\n",
|             nmbpage, nmbpage << 4);
|
|    if( (nmbpage = ems_free_page()) == EMS_ERR )
|      print_err();             /* Erreur : message d'erreur et terminer */
|    else
|      printf("... dont libres               : %d (%d Ko)\n",
|             nmbpage, nmbpage << 4);
|
|    if( (int) (pageseg = ems_frame_seg()) == EMS_ERR )
|      print_err();             /* Erreur : message d'erreur et terminer */
|    else
|      printf("Adresse de segment du Page Frame : %X\n", pageseg);
|
|    printf("\nUne page de la mémoire EMS va maintenant être\n");
|    printf("allouée et le contenu de l'écran va être copié de la\n");
|    printf("RAM vidéo dans cette page.\n");
|    printf("                        ... Veuillez frapper une touche\n");
|    getch();                                  /* Attendre une touche */
|
|    /*-- Allouer une page et la calquer sur la première page ----------*/
|    /*-- logique dans le Page Frame                          ----------*/
|    if( (handle = ems_alloc(1)) == EMS_ERR )
|      print_err();             /* Erreur : message d'erreur et terminer */
|    if( !ems_map(handle, 0, 0) )                        /* Fixer Mapping */
|      print_err();             /* Erreur : message d'erreur et terminer */
|
|    /*-- Copier 4000 octets de la RAM vidéo dans la mémoire EMS -------*/
|
|    memcpy( PAGE_ADR(0), vr_adr(), 4000 );
|
|    for( i = 0; i < 24; ++i )                          /* Vider l'écran */
|      printf("\n");
|
|    printf("L'ancien contenu de l'écran a maintenant été effacé et\n");
|    printf("est donc perdu. Cependant, comme il avait\n");
|    printf("été sauvegardé dans la mémoire EMS, il peut être recopié\n");
|    printf("dans la RAM vidéo à partir de là.\n");
|    printf("                        ... Veuillez frapper une touche\n");
|    getch();                                  /* Attendre une touche */
|
|    /*-- Recopier le contenu de la RAM vidéo d'après la mémoire EMS ---*/
|    /*-- et libérer à nouveau la mémoire EMS allouée               ---*/
|
|    memcpy( vr_adr(), PAGE_ADR(0), 4000 );          /* Recopier V-RAM */
|    if( !ems_free(handle) )                        /* Libérer mémoire */
|      print_err();             /* Erreur : message d'erreur et terminer */
|    printf("FIN");
|  }
|  else                          /* Le driver EMS n'a pu être identifié */
|    printf("ATTENTION ! Pas de mémoire EMS installée\n");
|}
|
```

Listing : EMMP.PAS

```
{*********************************************************************}
{*                          E M M P                                  *}
{*-------------------------------------------------------------------*}
{*  Fonction       : Fournit différentes fonctions permettant        *}
{*                   d'accéder à la mémoire EMS ou à l'EMM.          *}
{*-------------------------------------------------------------------*}
{*  Auteur         : MICHAEL TISCHER                                 *}
{*  Développé le   : 30/08/1988                                      *}
{*  Dernière modif.: 30/03/1992                                      *}
{*********************************************************************}

program EMMP;

Uses Dos, CRT;                          { Intégrer les unités DOS et CRT }

type  ByteBuf = array[0..1000] of byte;      { zone mémoire comme Byte }
      CharBuf = array[0..1000] of char;      { zone mémoire comme Char }
      BytePtr = ^ByteBuf;                 { Pointeur sur une zone d'octet }
      CharPtr = ^CharBuf;                   { Pointeur sur une zone Char }

const EMS_INT   = $67;          { N° d'interruption pour l'accès à l'EMM }
      EMS_ERR   = -1;                     { Est renvoyé en cas d'erreur }
      W_EMS_ERR = $FFFF;                { Code d'erreur en format WORD }
      EmmName   : array[0..7] of char = 'EMMXXXX0';    { Nom de l'EMM }

var   EmmEC,                          { Ici sont placés les codes d'erreur EMM }
      i       : byte;                           { Compteur de boucle }
      Handle,                        { Handle pour l'accès à la mémoire EMS }
      EmmVer  : integer;                  { Numéro de version de l'EMM }
      NmbPage,                                 { Nombre de pages EMS }
      PageSeg : word;                { Adresse de segment du Page Frame }
      Touche  : char;


{*********************************************************************}
{* EmsInst: Détermine si mémoire EMS et un EMM correspondant         *}
{*          installés.                                               *}
{* Entrée : Aucune                                                   *}
{* Sortie : TRUE si mémoire EMS présente, sinon FALSE.               *}
{*********************************************************************}

function EmsInst : boolean;

type  EmmName  = array[1..8] of char;          { Nom du driver EMS }
      EmmNaPtr = ^EmmName;                   { Pointeur sur le nom }

const Name : EmmName = 'EMMXXXX0';           { Désignation du driver }

var Regs : Registers;  { Registres processeur pour appel interruption }

begin
        {*-- Mettre en place ptr sur nom dans en-tête driver de périph.}
  Regs.ax := $35 shl 8 + EMS_INT; { N° fct.: Rech. vec. interrupt. $67 }
  MsDos( Regs );                        { Appeler interruption DOS $21 }

  EmsInst := (EmmNaPtr(Ptr(Regs.ES,10)^ = Name);
                                                  { TRUE si nom trouvé }
end;

{*********************************************************************}
{* EmsNumPage: Détermine le nombre total de pages EMS.               *}
{* Entrée : Aucune                                                   *}
{* Sortie : EMS_ERR en cas d'erreur, sinon nombre de pages EMS.      *}
{*********************************************************************}

function EmsNumPage : integer;

var Regs : Registers;  { Registres processeur pour appel interruption }

begin
  Regs.ah := $42;              { N° fct.: Déterminer nombre de pages }
  Intr(EMS_INT, Regs);                                { Appeler EMM }
  if( Regs.ah <>0 ) then              { Une erreur est-elle apparue ? }
    begin                                                  { Oui }
      EmmEC := Regs.ah;                        { Ranger le code d'erreur }
      EmsNumPage := EMS_ERR;                     { Afficher l'erreur }
    end
  else                                                 { Pas d'erreur }
    EmsNumPage := Regs.dx;          { Renvoyer le nombre total de pages }
end;

{*********************************************************************}
{* EmsFreePage: Détermine le nombre de pages EMS encore libres.      *}
{* Entrée : Aucune                                                   *}
{* Sortie : EMS_ERR en cas d'erreur, sinon nombre de pages EMS       *}
{*          non encore réservées.                                    *}
{*********************************************************************}

function EmsFreePage : integer;

var Regs : Registers;  { Registres processeur pour appel interruption }

begin
  Regs.ah := $42;              { N° fct.: Déterminer nombre de pages }
  Intr(EMS_INT, Regs);                                { Appeler EMM }
  if (Regs.ah <>0 ) then              { Une erreur est-elle apparue ? }
    begin                                                  { Oui }
      EmmEC := Regs.ah;                        { Ranger le code d'erreur }
      EmsFreePage := EMS_ERR;                    { Afficher l'erreur }
    end
  else                                                 { Pas d'erreur }
    EmsFreePage := Regs.bx;         { Renvoyer le nombre de pages libres }
end;

{*********************************************************************}
{* EmsFrameSeg: Détermine l'adresse de segment du Page Frame.        *}
{* Entrée : Aucune                                                   *}
{* Sortie : EMS_ERR en cas d'erreur, sinon adr. seg. obtenue         *}
{*********************************************************************}

function EmsFrameSeg : word;

var Regs : Registers;  { Registres processeur pour appel interruption }

begin
  Regs.ah := $41;           { N° fct.: Rech adr segment de Page Frame }
  Intr(EMS_INT, Regs);                                { Appeler EMM }
  if (Regs.ah <>0 ) then              { Une erreur est-elle apparue ? }
    begin                                                  { Oui }
      EmmEC := Regs.ah;                        { Ranger le code d'erreur }
      EmsFrameSeg := W_EMS_ERR;                  { Afficher l'erreur }
    end
  else                                                 { Pas d'erreur }
    EmsFrameSeg := Regs.bx;   { Renvoyer adresse segment de Page Frame }
end;

{*********************************************************************}
{* EmsAlloc: Alloue le nombre de pages spécifié et renvoie un handle *}
{*           pour l'accès à ces pages.                               *}
{* Entrée : PAGES: nombre de pages à allouer.                        *}
{* Sortie : EMS_ERR en cas d'erreur, sinon le handle.                *}
{*********************************************************************}

function EmsAlloc( Pages : integer ) : integer;

var Regs : Registers;  { Registres processeur pour appel interruption }

begin
  Regs.ah := $43;                          { N° fct.: Allouer pages }
  Regs.bx := Pages;                  { Fixer nombre de pages à allouer }
  Intr(EMS_INT, Regs);                                { Appeler EMM }
  if (Regs.ah <>0 ) then              { Une erreur est-elle apparue ? }
    begin                                                  { Oui }
      EmmEC := Regs.ah;                        { Ranger le code d'erreur }
      EmsAlloc := EMS_ERR;                       { Afficher l'erreur }
    end
  else                                                 { Pas d'erreur }
    EmsAlloc := Regs.dx;                          { Renvoyer handle }
end;

{*********************************************************************}
{* EmsMap  : Calque une des pages logiques allouées sous le handle   *}
{*           transmis sur la page physique du Page Frame.            *}
{* Entrée : HANDLE: Le Handle fourni par EmsAlloc.                   *}
{*          LOGP  : La page logique à calquer                        *}
{*          PHYSP : La page physique dans le Page Frame.             *}
{* Sortie : FALSE en cas d'erreur, sinon TRUE.                       *}
{*********************************************************************}

function EmsMap(Handle, LogP : integer; PhysP : byte) : boolean;

var Regs : Registers;  { Registres processeur pour appel interruption }
begin
  Regs.ah := $44;                          { N° fct.: Fixer Mapping }
  Regs.al := PhysP;                          { Fixer page physique }
  Regs.bx := LogP;                           { Fixer page logique }
  Regs.dx := Handle;                         { Fixer Handle EMS }
  Intr(EMS_INT, Regs);                                { Appeler EMM }
  EmmEC := Regs.ah;                       { Ranger code d'erreur }
  EmsMap := (Regs.ah = 0)           { Renvoyer TRUE si pas d'erreur }
end;

{*********************************************************************}
{* EmsFree : Libère à nouveau la mémoire allouée sous un Handle      *}
{* Entrée : HANDLE: Le Handle fourni par EmsAlloc.                   *}
```

```
{* Sortie : FALSE en cas d'erreur, sinon TRUE.                    *}
{*****************************************************************}

function EmsFree(Handle : integer) : boolean;

var Regs : Registers;   { Registres processeur pour appel interruption }

begin
  Regs.ah := $45;                              { N° fct.: Libérer pages }
  Regs.dx := handle;                             { Fixer Handle EMS }
  Intr(EMS_INT, Regs);                             { Appeler EMM }
  EmmEC := Regs.ah;                          { Ranger code d'erreur }
  EmsFree := (Regs.ah = 0)             { Renvoyer TRUE si pas d'erreur }
end;

{*****************************************************************}
{* EmsVersion: Détermine le numéro de version de l'EMM           *}
{* Entrée : Aucune                                               *}
{* Sortie : EMS_ERR en cas d'erreur, sinon le numéro de version. *}
{*          11 signifie 1.1, 40 signifie 4.0 etc.                *}
{*****************************************************************}

function EmsVersion : integer;

var Regs : Registers;   { Registres processeur pour appel interruption }

begin
  Regs.ah := $46;                    { N° fct.: Déterminer version EMM }
  Intr(EMS_INT, Regs);                             { Appeler EMM }
  if (Regs.ah <>0 ) then             { Une erreur est-elle apparue ? }
    begin                                                { Oui }
      EmmEC := Regs.ah;                        { Ranger le code d'erreur }
      EmsVersion := EMS_ERR;                     { Afficher l'erreur }
    end
  else  { Pas d'erreur, calculer numéro de version d'après nombre BCD }
    EmsVersion := (Regs.al and 15) + (Regs.al shr 4) * 10;
end;

{*****************************************************************}
{* EmsSaveMap: Sauvegarde calquage entre pages logiques et physiques *}
{*             pour le Handle transmis.                          *}
{* Entrée : HANDLE: Le Handle fourni par EmsAlloc.               *}
{* Sortie : FALSE en cas d'erreur, sinon TRUE.                   *}
{*****************************************************************}

function EmsSaveMap( Handle : integer ) : boolean;

var Regs : Registers;   { Registres processeur pour appel interruption }

begin
  Regs.ah := $47;                    { N° fct.: Sauvegarder Mapping }
  Regs.dx := handle;                             { Fixer Handle EMS }
  Intr(EMS_INT, Regs);                             { Appeler EMM }
  EmmEC := Regs.ah;                          { Ranger code d'erreur }
  EmsSaveMap := (Regs.ah = 0)          { Renvoyer TRUE si pas d'erreur }
end;

{*****************************************************************}
{* EmsRestoreMap: Rétablit le calquage entre pages logiques et   *}
{*                physiques tel qu'il avait été sauvegardé auparavant *}
{*                avec EmsSaveMap.                               *}
{* Entrée : HANDLE: Le Handle fourni par EmsAlloc.               *}
{* Sortie : FALSE en cas d'erreur, sinon TRUE.                   *}
{*****************************************************************}

function EmsRestoreMap( Handle : integer ) : boolean;

var Regs : Registers;   { Registres processeur pour appel interruption }

begin
  Regs.ah := $48;                    { N° fct.: Rétablir Mapping }
  Regs.dx := handle;                             { Fixer Handle EMS }
  Intr(EMS_INT, Regs);                             { Appeler EMM }
  EmmEC := Regs.ah;                          { Ranger code d'erreur }
  EmsRestoreMap := (Regs.ah = 0)       { Renvoyer TRUE si pas d'erreur }
end;

{*****************************************************************}
{* PrintErr: Sort un message d'erreur EMS et termine le programme. *}
{* Entrée : Aucune                                               *}
{* Sortie : Aucune                                               *}
{* Infos  : Cette fonction ne doit être appelée que si une erreur a *}
{*          été signalée lors d'un appel préalable d'une fonction de *}
{*          ce module.                                           *}
{*****************************************************************}

procedure PrintErr;

begin
  writeln('ATTENTION ! Erreur d''accès à la mémoire EMS');
  write('     ... ');
  if ((EmmEC<$80) or (EmmEc>$8E) or (EmmEc=$82)) then
    writeln('non identifiable')
  else
    case EmmEC of
      $80 : writeln('Erreur dans le driver EMS (EMM détruit)');
      $81 : writeln('Erreur dans l''électronique EMS');
      $83 : writeln('Handle EMM incorrect');
      $84 : writeln('Fonction EMS appelée n''existe pas');
      $85 : writeln('Plus de handles EMS disponibles');
      $86 : writeln('Erreur de sauvegarde ou de reconstitution ',
                    'du Mapping');
      $87 : writeln('Plus de pages réclamées qu''il n''en existe ',
                    'physiquement');
      $88 : writeln('Plus de pages réclamées qu''il n''en reste',
                    ' de libres');
      $89 : writeln('Zéro page réclamée');
      $8A : writeln('Page logique ne correspondant pas au Handle');
      $8B : writeln('Numéro de page physique incorrect');
      $8C : writeln('Zone de mémoire de Mapping pleine');
      $8D : writeln('Sauvegarde du Mapping déjà effectuée');
      $8E : writeln('Reconstitution du Mapping sans ',
                    'sauvegarde antérieure');
    end;
  Halt;                                          { Terminer programme }
end;

{*****************************************************************}
{* VrAdr: Fournit un pointeur sur la RAM vidéo.                  *}
{* Entrée : Aucune                                               *}
{* Sortie : Pointeur sur la RAM vidéo.                           *}
{*****************************************************************}

function VrAdr : BytePtr;

var Regs : Registers;   { Registres processeur pour appel interruption }

begin
  Regs.ah := $0f;                      { N° fct.: Déterminer mode vidéo }
  Intr($10, Regs);                  { Appeler interruption vidéo du BIOS }
  if (Regs.al = 7) then                    { Carte d'écran monochrome ? }
    VrAdr := ptr($8000, 0)                 { Oui, RAM vidéo en B000:0000 }
  else                                      { Carte couleur, EGA ou VGA }
    VrAdr := ptr($8800, 0);                  { RAM vidéo en B800:0000 }
end;

{*****************************************************************}
{* PageAdr: Fournit l'adresse d'une page physique dans le Page Frame *}
{* Entrée : PAGE: Numéro de page physique (0 à 3)                *}
{* Sortie : Pointeur sur la page physique.                       *}
{*****************************************************************}

function PageAdr( Page : integer ) : BytePtr;

begin
  PageAdr := ptr( EmsFrameSeg + (Page shl 10), 0 );
end;

{*****************************************************************}
{**                      PROGRAMME PRINCIPAL                      **}
{*****************************************************************}

begin
  ClrScr;                                          { Vider l'écran }
  writeln('EMMP - (c) 1988, 92 by MICHAEL TISCHER',#13#10);
  if EmsInst then                          { Mémoire EMS installée ? }
    begin                                                { Oui }
      (*-- Sortir informations sur la mémoire EMS ----------------*)

      EmmVer := EmsVersion;          { Déterminer numéro de version EMM }
      if EmmVer = EMS_ERR then         { Une erreur est-elle apparue ? }
        PrintErr; { Oui, sortir message d'erreur et terminer programme }
      writeln('Numéro de version EMM        : ',EmmVer div 10, '.',
              EmmVer mod 10);

      NmbPage := EmsNumPage;         { Déterminer nombre total de pages }
      if NmbPage = EMS_ERR then        { Une erreur est-elle apparue ? }
        PrintErr; { Oui, sortir message d'erreur et terminer programme }
      writeln('Nombre de pages EMS          : ', NmbPage, ' (',
              NmbPage shl 4, ' Ko)');

      NmbPage := EmsFreePage;        { Déterminer nombre de pages libres }
      if NmbPage = EMS_ERR then        { Une erreur est-elle apparue ? }
        PrintErr; { Oui, sortir message d'erreur et terminer programme }
      writeln('... dont libres              : ', NmbPage, ' (',
              NmbPage shl 4, ' Ko)');

      PageSeg := EmsFrameSeg;        { Adresse de segment du Page Frame }
      if PageSeg = W_EMS_ERR then      { Une erreur est-elle apparue ? }
        PrintErr; { Oui, sortir message d'erreur et terminer programme }
      writeln('Adresse de segment du Page Frame: ', EmsFrameSeg );

      writeln;
      writeln('On alloue maintenant une page de la mémoire EMS et');
      writeln('le contenu de l''écran est copié de la RAM vidéo');
      writeln('dans cette page.');
```

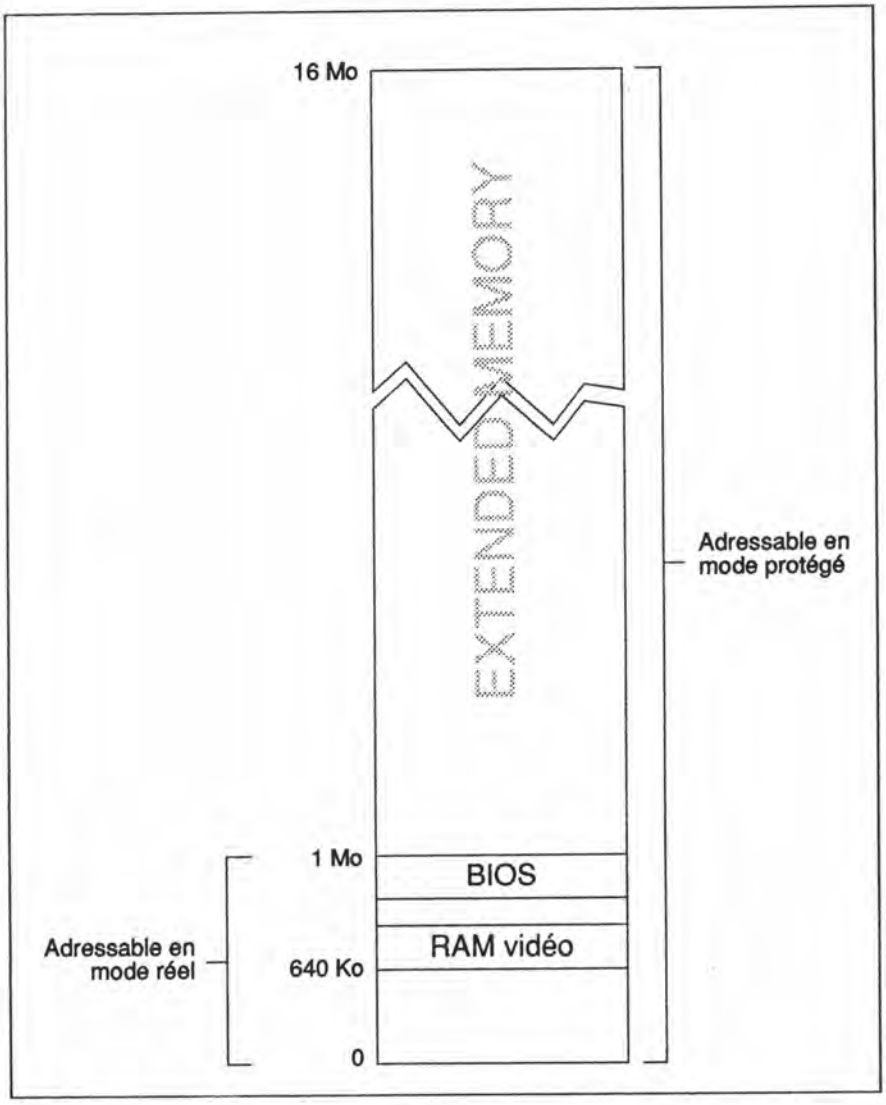
```
    writeln('                    ... veuillez actionner une Touche');
    Touche := ReadKey;                          { Attendre une touche }

    {*-- Allouer une page et la calquer sur la première page  ------*}
    {*-- logique dans le Page Frame                           ------*}

    Handle := EmsAlloc( 1 );                    { Allouer une page }
    if Handle = EMS_ERR then        { Une erreur est-elle apparue ? }
      PrintErr; { Oui, sortir message d'erreur et terminer programme }
    if not(EmsMap(Handle, 0, 0)) then           { Fixer Mapping }
      PrintErr;             { Erreur : message d'erreur et terminer }

    {*-- Copier 4000 octets de la RAM vidéo dans la mémoire EMS ----*}

    Move(VrAdr^, PageAdr(0)^, 4000);

    ClrScr;                                     { Vider l'écran }
    while KeyPressed do                         { Lire le buffer clavier }
      Touche := ReadKey;
    writeln('L''ancien contenu de l''écran a maintenant été vidé et');
    writeln('est donc définitivement perdu. Mais comme il avait été');
    writeln('sauvegardé dans la mémoire EMS, il peut être recopié');
    writeln('de là dans la RAM vidéo.');
    writeln('                    ... Veuillez actionner une Touche');
    Touche := ReadKey;                          { Attendre une touche }

      {*-- Recopier le contenu de la RAM vidéo d'après la mémoire --*}
      {*-- EMS et libérer à nouveau la mémoire EMS allouée        --*}

    Move(PageAdr(0)^, VrAdr^, 4000);            { Recopier RAM vidéo }
    if not(EmsFree(Handle)) then                { Libérer mémoire }
      PrintErr;             { Erreur : message d'erreur et terminer }
    GotoXY(1, 15);
    writeln('FIN')
  end                                   { Le driver EMS n'a pu être identifié }
else
  writeln('ATTENTION ! Pas de mémoire EMS installée.');
end.
```

12.2. La mémoire étendue

Alors qu'un programme ne doit réclamer la mémoire paginée que par à-coups, l'ensemble de la mémoire étendue peut être accédé simultanément, à condition de se trouver au préalable en mode protégé. En mode réel, la mémoire étendue ne donne aucun résultat parce qu'elle est inaccessible par le processeur au-delà de la limite des 1 Mo. Par conséquent, les ordinateurs équipés d'un processeur 8088 ou 8086, c'est-à-dire les XT de forme simple, ne peuvent pas utiliser la mémoire étendue. Leurs processeurs ne connaissent que le mode réel et, contrairement à leurs aînés, ils ne sont pas en mesure de commuter vers le mode protégé.

En fait, il n'est pas du tout compliqué d'accéder au mode protégé. Il suffit tout simplement de régler un bit précis dans le registre de flags du processeur. Une fois cette action réalisée, le programme dispose d'une mémoire étendue allant jusqu'à 16 Mo dont on ne peut profiter puisque l'ordinateur se plante dans le moment qui suit. Le responsable de ce désagrément est le mécanisme utilisé par le 80286 et les modèles suivants pour adresser la mémoire en mode protégé. Cette technique diffère complètement de celle adoptée en mode réel.

Le processeur ne travaille plus avec les adresses de segment directes, mais ce qu'on appelle les descripteurs de segment regroupés dans une liste globale et d'innombrables listes locales. Il ne faut espérer ici aucun soutien de la part de DOS dont le travail se limite exclusivement au mode réel sans pouvoir commuter de quelque manière que ce soit vers le mode protégé.



La mémoire étendue dans la cellule d'adresse d'un PC

Si on ne définit et n'initialise pas personnellement les listes du descripteur avant de commuter vers le mode protégé, on obtient vite de mauvaises surprises. Cette tâche nécessite en effet quelque expérience dans la programmation en Assembleur et des connaissances approfondies sur le fonctionnement du processeur en mode protégé.

Les diverses fonctions BIOS et les drivers XMS, destinés spécialement à assumer cette tâche, apportent fort heureusement une aide intéressante lors de l'accès à la mémoire étendue. Par rapport aux fonctions BIOS, ces drivers offrent un meilleur mécanisme d'accès à la mémoire étendue. Ils partent du principe que la mémoire étendue ne doit

pas être adressée uniquement par un seul programme, mais doit être partagée par plusieurs programmes. La bousculade pour les droits d'accès à la mémoire étendue reste néanmoins le problème majeur dont nous parlerons dans les sections suivantes.

Ces problèmes deviennent particulièrement épineux lorsque vous accédez directement à la mémoire étendue, alors que cela est également possible à partir du mode réel pour les 64 Ko inférieurs. Ce chapitre explique également cette méthode.

Mais nous parlerons surtout des diverses techniques d'accès dans les sections suivantes qui traitent notamment des fonctions BIOS, de l'accès direct à la HMA et des fonctions du standard XMS.

12.2.1. Accès à la mémoire étendue par le BIOS

On peut ou plutôt on ne doit accéder à la mémoire étendue que lorsque cette dernière existe bien évidemment. C'est pratiquement le cas de tous les ordinateurs équipés d'une mémoire supérieure à 1 Mo dont seulement 640 ou 512 Ko sont généralement répartis en-deçà de la limite des 1 Mo. Le reste se trouve toujours au-dessus de cette limite et constitue la mémoire étendue.

Pour que l'utilisateur n'ait pas à préciser manuellement la taille de la mémoire étendue disponible lors du lancement d'une application, un programme doit contrôler automatiquement l'existence et la taille de la mémoire étendue. A cet effet, on peut se servir d'une fonction BIOS accessible à travers l'interruption 15h. En fait, elle a été conçue en tant qu'interface par rapport au registre de cassettes et utilisée comme outil d'enregistrement des informations durant les premiers jours du PC. L'élimination de ces périphériques du monde PC a entraîné également la disparition des fonctions correspondantes qui étaient accessibles à travers cette interruption.

De nouvelles fonctions ont pris leur place. Elles ne servent pas uniquement à accéder à la mémoire étendue, mais elles remplissent aussi d'autres tâches comme celle de déterminer les joysticks. L'une de ces fonctions est la fonction 88h. Elle retourne la taille de la mémoire étendue lorsqu'elle est appelée à travers l'interruption 15h avec le numéro de fonction situé dans le registre AH. Le résultat apparaît dans le registre AX qui indique la taille de la mémoire étendue en Ko.

C'est très agréable d'obtenir une information sur une telle mémoire, mais à quoi bon si on ne peut y accéder (du moins à partir du mode réel) ?

Pour apaiser tout souci, l'interruption cassettes fournit la fonction 87h permettant de copier des blocs mémoire à l'intérieur de la zone de mémoire. Autrement dit, des blocs mémoire de 1 Mo peuvent être transférés au-delà de cette limite et inversement. En outre, il est possible de copier des blocs mémoire au-dessus et au-dessous de la limite des 1 Mo. Cette fonction ne doit cependant pas être utilisée pour remplir la dernière tâche car il est très difficile de l'appeler et elle comporte des inconvénients.

En fait, pour franchir réellement la frontière des 1 Mo, le processeur doit être activé en mode protégé. Dans ce mode, les performances du 80286 et des modèles ultérieurs deviennent certes apparentes, il n'en reste pas moins que, dans ce mode, la programmation des processeurs présente des complexités à divers points de vue. En mode réel au contraire, ils peuvent être exploités sous DOS. Ainsi, les informations à fournir à la fonction 87h par un programme sont réellement abondantes.

Nous n'allons pas vous accabler par la liste de ces informations. Mais pour simplifier la tâche, nous avons écrit un programme dans les divers langages. Il se trouve à la fin du chapitre. Vous pouvez vous en servir pour programmer le processeur avec la fonction 87h sans avoir des connaissances approfondies. Que faut-il donc transmettre à la fonction 87h lors de son appel ?

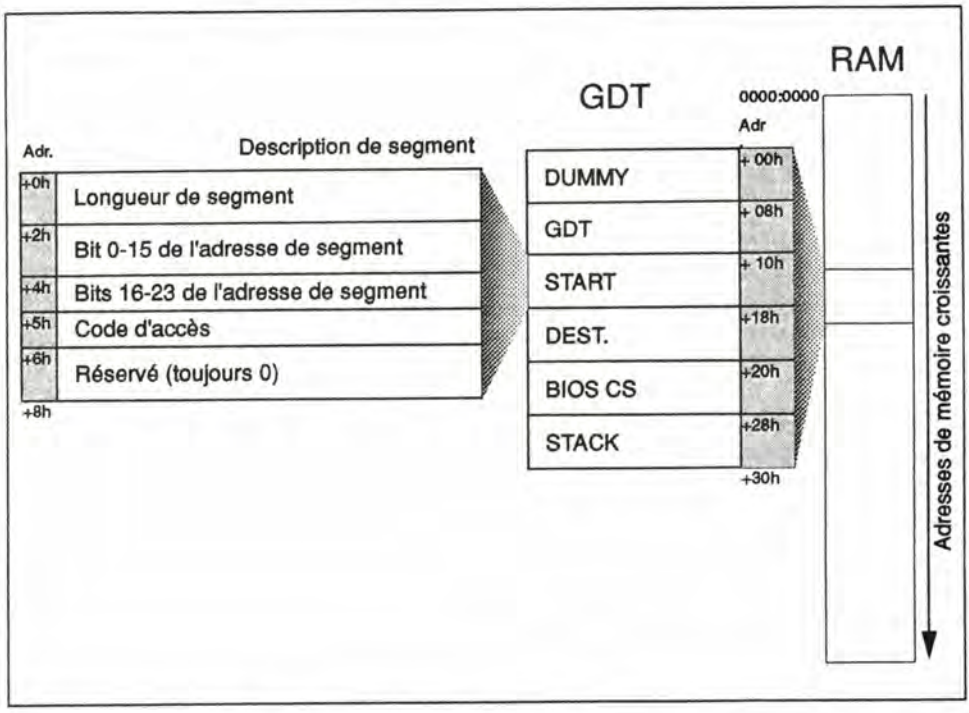
Tout d'abord, il convient naturellement de préciser le numéro de fonction 87h dans le registre AH. Puis, le nombre de mots à déplacer -il peut s'agir également d'octets- dans le registre AX. La valeur maximale autorisée ici est 8000h correspondant à un maximum de 64 Ko.

La table de descripteur globale (GDT)

Dans la paire de registres ES:SI, il faut transmettre l'adresse de la GDT que le programmeur doit définir dans son programme. La GDT (Global Descriptor Table) est une table décrivant les différents segments de mémoire du 80286 en mode protégé. Dans ce contexte, notez que les segments ne sont pas soumis aux mêmes limitations en mode protégé qu'en mode réel. En mode réel, ils doivent commencer obligatoirement par une cellule de mémoire divisible par 16. En mode protégé, ils peuvent commencer à une cellule quelconque. Il n'est pas indispensable que leur taille soit égale à 64 Ko, mais elle peut varier entre 1 octet et 64 Ko.

Une autre nouveauté du mode protégé est le code d'accès qui doit être défini pour chaque segment. Il indique si le segment décrit est un segment de données ou de codes (seuls ce type de segments peuvent être exécutés). Il spécifie également la priorité à respecter pour accéder au segment et si le segment peut réellement être décrit. Si on regroupe ces informations et si on obtient la valeur 0 à la fin du descripteur pour le mot réservé, cela signifie chaque descripteur de segment se compose de 8 octets.

Lors de l'appel, la fonction 87h attend que 6 descripteurs de segment soient déjà définis dans la GDT ou la place nécessaire leur soit réservée. La figure suivante montre de quels descripteurs il s'agit et décrit la structure d'un descripteur de segment.



La structure du descripteur de segment vue par la fonction 87H

Pour nous, seuls les deux descripteurs de segment désignés par descripteur source et descripteur cible présentent un réel intérêt car la fonction BIOS se charge de compléter les autres descripteurs. Le premier décrit le segment à partir duquel les données sont à extraire. Le descripteur cible décrit le segment dans lequel les données sont à copier.

Comme longueur de segment, vous pouvez indiquer 0FFFFh, soit 64 Ko dans les deux cas, même si vous souhaitez copier moins d'octets (ou mots). Si vous précisez toutefois une petite valeur, sachez qu'elle doit toujours être supérieure au nombre d'octets à copier (nombre de mots fois 2). Sinon, le processeur effectue un accès au-delà de la limite imposée lors de la copie et provoque une erreur.

Vous devez convertir l'adresse des deux zones de mémoire en une adresse (physique) 24 bits. Les 16 bits inférieurs de cette adresse sont à préciser dans le second champ du descripteur de segment et les 8 bits supérieurs dans le troisième champ. Comme code d'accès, vous pouvez utiliser toujours 92h. Ce code signale au processeur que le segment décrit est un segment de données doté de la plus haute parité, qu'il se trouve dans la mémoire et peut être décrit. Le dernier champ du descripteur n'existe que pour des raisons de compatibilité avec le processeur 80386. Par conséquent, il doit toujours contenir la valeur 0.

A l'intérieur de votre programme, l'adresse du buffer dans lequel ou à partir duquel vous souhaitez copier des données est prédéfinie. Quant à l'adresse située au-delà des 1 Mo

dans laquelle vous allez copier les données (dans le cadre de la mémoire RAM disponible), vous pouvez la sélectionner librement. Le tableau suivant indique les adresses des divers blocs de 1 Ko au-delà de la limite des 1 Mo sous forme d'adresses 24 bits.

```
0 Ko = 100000h  124 Ko = 11F000h
1 Ko = 100400h  125 Ko = 11F400h
2 Ko = 100800h  126 Ko = 11F800h
3 Ko = 100C00h  127 Ko = 11FC00h
4 Ko = 101000h  128 Ko = 120000h
5 Ko = 101400h  129 Ko = 120400h
6 Ko = 100800h  130 Ko = 120800h
7 Ko = 100C00h  131 Ko = 120C00h
8 Ko = 102000h  132 Ko = 121000h
9 Ko = 102400h  133 Ko = 121400h
  ......
  ......
  ......
60 Ko = 10F000h  252 Ko = 12F000h
61 Ko = 10F400h  253 Ko = 13F400h
62 Ko = 10F800h  254 Ko = 13F800h
63 Ko = 10FC00h  255 Ko = 13FC00h
64 Ko = 110000h  256 Ko = 140000h
65 Ko = 110400h  257 Ko = 140400h
66 Ko = 110800h  258 Ko = 140800h
67 Ko = 110C00h  259 Ko = 140400h
68 Ko = 111000h  260 Ko = 141000h
69 Ko = 111400h  261 Ko = 141400h
```

Codes d'erreur lors de l'accès à la Hi-RAM

Après l'appel de la fonction, le flag Carry informe si l'appel s'est déroulé correctement. S'il est mis, cela signifie qu'une erreur s'est produite et la valeur fournie par le registre AH précise la cause de l'erreur :

Numéro d'erreur	Cause de l'erreur
AH = 0	Pas d'erreur (flag Carry initialisé)
AH = 1	Erreur de parité RAM
AH = 2	GDT défectueuse lors de l'appel
AH = 3	Impossible d'initialiser correctement le mode protégé

Il convient d'évoquer l'inconvénient de cette fonction. Toutes les interruptions doivent être interdites pendant que le processeur se trouve notamment en mode protégé. Expliquons la raison de cette circonstance : en mode protégé, vous pouvez appeler des interruptions BIOS (du Timer ou du clavier), mais ces routines ne sont conçues que pour le mode réel. Par conséquent, elles ne fonctionnent pas correctement en mode protégé.

Le désagrément causé par ce processus apparaît nettement dans le Timer. Etant donné que ces interruptions sont interdites, l'heure ne peut pas être calculée en mode protégé. Elle reste inchangée pendant un laps de temps. Si on appelle trop souvent la fonction 87h, il arrive que l'heure retarde de 20 à 30 secondes au bout d'un jour. Cependant, elle peut toujours être réglée correctement par le logiciel si bien que ce léger inconvénient passe inaperçu par rapport aux avantages de cette fonction.

La vitesse de travail de cette fonction, qui laisse à désirer, est beaucoup plus gênante que l'interdiction des interruptions. Cela concerne surtout les AT équipés d'un processeur 80286. On peut certes les activer en mode protégé, mais pour accéder au mode réel à partir de ce mode, il est indispensable d'effectuer quelques opérations. Les responsables ne sont autres que les développeurs de ce processeur qui n'ont pas implanté une commande permettant de commuter vers le mode réel.

Il faut tout simplement admettre l'oubli de cette commande ou penser que personne ne souhaitera passer du confortable mode protégé vers le mode réel de structure relativement simple. Toujours est-il qu'une telle commande fait défaut dans le répertoire du 80286. Pour revenir par conséquent au mode réel, il existe une seule méthode draconienne consistant à réinitialiser le processeur.

Dans un AT, cette action ne peut être réalisée qu'à travers le contrôleur du clavier. Mais en attendant que la commande ne soit reçue et traitée, il s'écoule au moins 6 millisecondes, ce qui représente une petite éternité pour un ordinateur 10, 12 ou 16 MHz. Mais ce n'est pas tout ! Après le reset, le processeur démarre en mode réel et commence à exécuter le programme avec le code Boot du BIOS.

Pour que le BIOS opère néanmoins un démarrage à chaud tout à fait normal et efface d'abord toute la mémoire, il faut spécifier un code dans une cellule précise de la RAM BIOS avant le déclenchement du reset. Le code indique au BIOS la raison du reset. Ainsi, le BIOS peut retourner directement au niveau de l'appel de la fonction 87h sans oublier que ce va-et-vient a duré naturellement un long moment.

Les machines PS/2 ont simplifié considérablement cette opération même si elles sont équipées d'un 80286. Leur développement a donné naissance à un mécanisme spécial destiné à commuter vers le mode réel. Mais ces systèmes ne peuvent toutefois pas tenir tête aux ordinateurs i386 et i486 capables de commuter vers le mode réel en toute aisance parce les développeurs d'INTEL ont su corriger les fautes d'autrefois.

Programmes d'exemple

Pour mieux expliquer l'accès à la mémoire étendue à l'aide des fonctions BIOS présentées ci-dessus, j'ai porté en fin de ce chapitre les listings des deux programmes EXTP.PAS et EXTC.C. Ils contiennent une routine nommée ExtCopy qui se charge d'appeler la fonction BIOS 87h pour copier des blocs de mémoire. Elle a besoin de deux pointeurs sur l'adresse du bloc source et du bloc cible ainsi que du nombre d'octets à copier. Ces deux pointeurs doivent être des pointeurs linéaires de vingt-quatre bits.

Pour vous éviter la conversion manuelle, les deux programmes possèdent chacun une fonction nommée ExtAdrConv qui transforme un banal pointeur FAR en adresse linéaire de vingt-quatre bits. Cette fonction est utilisée dans les deux routines ExtRead et ExtWrite qui lisent et écrivent dans la mémoire étendue.

En outre, les deux programmes contiennent chacun une routine nommée ExtGetInfo qui, à l'aide de la fonction BIOS 88h, permet de savoir s'il reste de la mémoire disponible et combien. Ce n'est pas tout. Elle recherche dans la mémoire étendue la présence de disques virtuels de type VDISK car ils peuvent poser des problèmes pendant l'utilisation de la mémoire étendue, voir la section 12.2.2 sous le titre "Conflits dans la mémoire étendue".

Les deux programmes ne se contentent pas de mettre ces routines à votre disposition, ils font une démonstration de leur appel. La mémoire étendue est parcourue par pas de un Ko dans lesquels ils écrivent et lisent une série d'octets aléatoire afin de détecter un mauvais fonctionnement de la mémoire ou une erreur du programme à l'appel des diverses fonctions, ce qui ne devrait pas se produire.

Listing : EXTP.PAS

```
{*********************************************************************
*                            E X T P . P A S                         *
**-------------------------------------------------------------------**
*  Démonstration de l'accès à la mémoire étendue par les fonctions   *
*  BIOS de l'interruption 15h, en gérant les disques virtuels        *
**-------------------------------------------------------------------**
*  Auteur          : MICHAEL TISCHER                                 *
*  développé le    : 18.05.1989                                      *
*  Dernière m. à j. : 19.02.1992                                     *
*********************************************************************}

program ExtP;

uses Dos;

{-- Variables globales ---------------------------------------------}

var RdLen    : integer;              { Taille disques virtuels en Ko }
    ExtAvail : boolean;              { Mémoire étendue disponible? }
    ExtStart : longint;     { Adresse de la mémoire étendue : linéaire }
    ExtLen   : integer;              { Taille mémoire étendue en Ko }

{*********************************************************************
*  ExtAdrConv : Convertit un pointeur en adresse linéaire de 32 bits  *
*               qui sera retournée sous forme d'un LONGINT            *
**-------------------------------------------------------------------**
*  Entrée : Adr = Le pointeur à convertir                             *
*  Sortie : l'adresse convertie                                       *
*********************************************************************}

function ExtAdrConv ( Adr : pointer ) : longint;

type PTRREC = record                    { Sert à accéder aux compo- }
                Ofs : word;             { sants d'un pointeur au    }
                Seg : word;             { choix                     }
              end;

begin
  ExtAdrConv := longint( PTRREC( Adr ).seg ) shl 4 + PTRREC( Adr ).ofs;
end;

{*********************************************************************
*  ExtCopy : Copie des donnés entre deux tampons quelconque dans la  *
*            limite adressable de 16 Mo des 80286/1386/1486.         *
**-------------------------------------------------------------------**
*  Entrée : Start = Adresse du tampon Start sous forme d'adresse     *
*                   linéaire sur 32 bits.                            *
*           But   = Adresse du tampon But sous forme d'adresse       *
*                   linéaire sur 32 bits.                            *
*           Len   = Nombre d'octets à copier                         *
*  Info   : - Le nombre d'octets à copier doit être pair.            *
*           - L'usage de cette procédure est réservé à l'intérieur de*
*             cette Unité.                                           *
*********************************************************************}

procedure ExtCopy( Start, But : longint; Len : word );

{-- Structures de données pour l'accès à la mémoire étendue --------}

type SDES = record                              { Descripteur segment }
```

```
                Longueur   : word;        { Longueur du segment en octets }
                AdrLo      : word;        { Bits 0 à 15 de l'adr. du segm }
                AdrHi      : byte;        { Bits 16 à 23 de l'adr. du segm }
                Attribut   : byte;                { Attribut du segment }
                Res        : word;                { Réservé pour le 1386 }
              end;

    GDT = record                                  { Global Descriptor Table }
              Dummy : SDES;
              GDTS  : SDES;
              Start : SDES;                                  { copie de ... }
              But   : SDES;                                  { ... vers }
              Code  : SDES;
              Stack : SDES;
            end;

    LI = record                          { Sert à accéder aux éléments d'un }
           LoWord : word;                { LongInts représentant une adresse }
           HiByte : byte;                                  { linéaire sur 32 bits }
           dummy  : byte;
         end;

var GTab : GDT;                                   { Global Descriptor Table }
    Regs : Registers;          { Registres process. pr appel interruption }
    Adr  : longint;                           { pour conversion de l'adresse }

begin
  FillChar( GTab, SizeOf( GTab ), 0 );              { Tous les champs à 0 }

  {-- Construction du descripteur du segment Start ------------------}

  GTab.Start.AdrLo      := LI( Start ).LoWord;
  GTab.Start.AdrHi      := LI( Start ).HiByte;
  GTab.Start.Attribut   := $92;
  GTab.Start.Longueur   := Len;

  {-- Construction du descripteur du segment But --------------------}

  GTab.But.AdrLo      := LI( But ).LoWord;
  GTab.But.AdrHi      := LI( But ).HiByte;
  GTab.But.Attribut   := $92;
  GTab.But.Longueur   := Len;

  {-- Copie de blocs de mémoire à l'aide de la fonction $87 de  -----}
  {-- l'interruption $15 pour cassettes                         -----}

  Regs.AH := $87;                      { N° de la fonction 'copier mémoire' }
  Regs.ES := seg( GTab );                              { Adresse de GDT }
  Regs.SI := ofs( GTab );                              { vers ES:SI }
  Regs.CX := Len shr 1;                { Nbre de mots à copier dans CX }
  Intr( $15, Regs );                                   { Appeler fonction }
  if ( Regs.AH <> 0 ) then                                    { Erreur? }
    begin                                      { Oui, AH contient code Erreur }
      writeln('Erreur en accès à la mémoire étendue (', Regs.AH,')!');
      RunError;                          { Quitter programme avec Run Time Error }
    end;
end;

{*********************************************************************
*  ExtRead : Copie un certain nombre d'octets de la mémoire étendue  *
*            dans la mémoire principale.                             *
**--------------------------------------------------------------------*
*  Entrée :  ExtAdr = Adresse source (linéaire) dans mémoire étendue *
*            BuPtr  = Pointeur sur le tampon cible dans mémoire princ. *
*            Len    = Nombre d'octets à copier                       *
*********************************************************************}

procedure ExtRead( ExtAdr : longint;  BuPtr : pointer;  Len : word );

begin
  ExtCopy( ExtAdr, ExtAdrConv( BuPtr ), len );
end;

{*********************************************************************
*  ExtWrite : Copie un certain nombre d'octets de la mémoire principale*
*             dans la mémoire étendue.                                *
**--------------------------------------------------------------------*
*  Entrée :  BuPtr  = Ptr sur tampon source dans mémoire principale   *
*            ExtAdr = Adresse But (linéaire) en mémoire étendue       *
*            Len    = Nombre d'octets à copier                       *
*********************************************************************}

procedure ExtWrite( BuPtr : pointer;  ExtAdr : longint;  Len : word );

begin
  ExtCopy( ExtAdrConv( BuPtr ), ExtAdr, len );
end;

{*********************************************************************
*  ExtGetInfo : Retourne l'adresse de la mémoire étendue et sa taille *
*               en tenant compte des éventuels disques virtuels de    *
*               type VDISK s'y trouvant                               *
**--------------------------------------------------------------------*
*  Entrée : aucune                                                   *
*  Sortie : aucune                                                   *
*  Globals : ExtAvail/W, ExtStart/W, ExtLen/W                        *
*********************************************************************}

procedure ExtGetInfo;

type NAME_TYP    = array [1..5] of char;
type BOOT_SECTEUR = record              { Secteur boot d'un disque virtuel }
                      dummy1   : array [1..3] of byte;
                      Name     : NAME_TYP;
                      dummy2   : array [1..3] of byte;
                      BpS      : word;
                      dummy3   : array [1..6] of byte;
                      Secteurs : word;
                      dummy4   : byte;    { remplissage pr taille paire }
                    end;

const VdiskName : NAME_TYP = 'VDISK';

var BootSek : BOOT_SECTEUR;               { copie le Boot Sector supposé }
    Derniere : boolean;                                { marque fin boucle }
    Regs     : Registers;    { Registres process. pr appel interruption }

begin
    {-- Demander la taille de la mémoire étendue et en déduire ------}
    {-- la présence éventuelle de mémoire étendue                  ---}

  Regs.ah := $88;  { n° de fonction: "renvoyer taille mémoire étendue" }
  Intr( $15, Regs );                    { Appeler interruption Cassette }
  if ( Regs.AX = 0 ) then
    begin                                             { Pas de RAM étendue }
      ExtAvail := FALSE;
      ExtLen   := 0;
      ExtStart := 0;
      exit;                                           { Retour à l'appelant }
    end;

  ExtAvail := TRUE;                          { Mémoire étendue disponible }
  ExtLen   := Regs.AX;              { copier sa taille dans une variable }

  {-- Rechercher disques virtuels type VDISK ----------------------}

  ExtStart := $100000;                                  { Si 1 Mo, lancer }
  Derniere := FALSE;                                 { A partir du RAM disk }
  repeat                                             { boucle interrogation }
    ExtRead( ExtStart, @BootSek, SizeOf( BootSek ) );
    with BootSek do
      if Name = VDiskName then     { secteur boot d'un disque virtuel? }
        inc( ExtStart, longint( Secteurs ) * BpS ) {Oui, après RAM disk}
      else
        Derniere := TRUE;              { Pas d'autres RAM disks trouvés }
  until Derniere;

    {-- Soustraire la taille des disques virtuels -------------------}
    {-- de la mémoire étendue disponible ----------------------------}

  dec( ExtLen, integer( (ExtStart - longint($100000)) shr 10) );
end;

{*********************************************************************
*  CheckExt : Teste la continuité de la mémoire étendue libre        *
*********************************************************************}

procedure CheckExt;

var AdrTest   : longint;                          { Adresse du bloc test }
    i, j      : integer;                          { Compteur d'itérations }
    WriteBuf,                                           { Blocs de test }
    ReadBuf   : array [1..1024] of byte;
    Erreur    : boolean;                          { Pointeur Erreur mémoire }

begin
  Randomize;                        { Initialise générateur nbres aléatoires }
  AdrTest := ExtStart;
  for i := 1 to ExtLen do                { lit la mémoire par blocs de 1 Ko }
    begin
      for j := 1 to 1024 do          { Remplir bloc avec nbres aléatoires }
        WriteBuf[ j ] := Random( 255 );

      write(@13, AdrTest );     { Affiche adresse du bloc de 1 Ko testé }

                  {-- Lire tampon et le copier dans ReadBuf  -------}

      ExtWrite( @WriteBuf, AdrTest, 1024 );
      ExtRead( AdrTest, @ReadBuf, 1024 );

        {-- Définir identité de WriteBuf et de ReadBuf  --------------}

      for j := 1 to 1024 do
        if WriteBuf[j] <> ReadBuf[j] then  { Contenu tampon identique? }
          begin                                              { Non, erreur! }
            writeln( '  Erreur! Adresse ',
                     AdrTest + longint(j-1) );
            Erreur := TRUE;
```

```
        end;

    inc( AdrTest, longint( 1024 ) );    ( Positionner AdrTest sur le )
  end;                                  ( bloc suivant d'1 Ko )
 writeln;
 if not( Erreur ) then                                ( Erreur ? )
   writeln( 'Ça baigne !' );                            ( Non )
end;

{*********************************************************************
* P R O G R A M M E   P R I N C I P A L                              *
*********************************************************************}

begin
  writeln( #13#10'EXTDEMO - (c) 1989,92  by Michael Tischer'#13#10);
  ExtGetInfo;       ( Donne disponibilité et taille de mémoire étendue )
  if ExtAvail then                        ( Mémoire étendue es-tu là ? )
    begin                                                      ( Oui )
      RdLen := integer( (ExtStart - longint( $100000 ) ) shr 10 );
      if ( RdLen = 0 ) then                    ( RAM disks installés ? )
        begin                                                  ( Non )
          writeln( 'Aucun RAM disk installé. ');
          writeln( 'La mémoire étendue libre commence à la ',
                   'limite du 1er Mo. ');
        end
      else                                    ( Oui, il y a des RAM disks ! )

begin
        writeln( 'Un ou plusieurs RAM disks occupent ',
                 RdLen,'Ko de mémoire étendue..');
        writeln( 'La mémoire étendue libre commence ', RdLen,
                 ' Ko après la limite du 1er Mo. ', RdLen, RdLen);
      end;
      writeln( ' Taille de la mémoire étendue libre ',
               ExtLen, ' Ko.');
      writeln( #13#10'Test de la continuité de la mémoire étendue',
               ' en cours...'#13#10);
      CheckExt;
  end
  else
    writeln( 'Pas de mémoire étendue dans votre ordinateur !');
end.
```

Listing : EXTC.C

```
/*********************************************************************
*                              E X T C . C                           *
**-----------------------------------------------------------------**
*  Démonstration de l'accès à la mémoire étendue par les fonctions   *
*  BIOS de l'interruption 15h, en gérant les disques virtuels        *
**-----------------------------------------------------------------**
*  Auteur            : MICHAEL TISCHER                              *
*  développé le      : 18.05.1989                                   *
*  Dernière m. à j.  : 19.02.1992                                   *
**-----------------------------------------------------------------**
*  Modèle de mémoire: SMALL                                         *
**-----------------------------------------------------------------**
*  Microsoft C       : Le message d'avertissement "Segment lost in  *
*                      conversation" est malheureusement inévitable *
*********************************************************************/

/*-- Intégrer les fichiers Include ----------------------------------*/

#include <stdio.h>
#include <stdlib.h>
#include <conio.h>
#include <string.h>
#include <dos.h>

/*-- Typedefs -------------------------------------------------------*/

typedef unsigned char BYTE;             /* nous fabriquons notre Byte */
typedef unsigned int WORD;
typedef BYTE BOOL;                      /* comme BOOLEAN en Pascal */

#define TRUE  ( 0 == 0 )
#define FALSE ( 0 == 1 )

/*-- Macros ---------------------------------------------------------*/

#ifndef __TURBOC__
  #define random(x) rand()
  #define randomize() srand(1)
#endif

/*-- Variables globales ---------------------------------------------*/

int  RdLen;                             /* Taille disques virtuels en Ko */
BOOL ExtAvail;                          /* Mémoire étendue disponible? */
long ExtStart;          /* Adresse mémoire étendue -> adresse linéaire. */
int  ExtLen;                            /* Taille mémoire étendue en Ko */

/*********************************************************************
*  ExtAdrConv : Convertit un pointeur FAR en adresse linéaire de 32  *
*               bits qui est retournée sous forme d'un long          *
**-----------------------------------------------------------------**
*  Entrée : Adr = Le pointeur à convertir                           *
*  Sortie : l'adresse convertie                                     *
*********************************************************************/

long ExtAdrConv( void far *Adr )
{
  return( (((long) Adr >> 16) << 4 ) + (unsigned int) Adr );
}

/*********************************************************************
*  ExtCopy : Copie des données entre deux tampons quelconques dans la*
*            limite adressable de 16 Mo des 80286/1386/1486.        *
**-----------------------------------------------------------------**
*  Entrée :  Start = Adresse du tampon Start en adresse lin. 32 bits *
*            But   = Adresse du tampon But en adresse linéaire 32 bits*
*            Len   = nombre des octets à copier                     *
*  Sortie : aucune                                                  *
*  Info   : Le nombre d'octets à copier doit être pair              *
*********************************************************************/

void ExtCopy( long Start, long But, WORD Len )
{
  /*-- Structures de données servant à l'accès à la mémoire étendue ---*/

  typedef struct {                              /* Descripteur segment */
                   WORD Longueur,       /* Longueur du segment en octets */
                        AdrLo;          /* Bits 0 à 15 de l'adr. du segm.*/
                   BYTE AdrHi,          /* Bits 16 à 23 de l'adr. du segm */
                        Attribut;                /* Attribut du segment */
                   WORD Res;                    /* Réservé pour le 1386 */
                 } SDES;

  typedef struct {                          /* Global Descriptor Table */
                   SDES Dummy,
                        GDTS ,
                        Start,                          /* Copier de ... */
                        But ,                             /* ... vers */
                        Code ,
                        Stack;
                 } GDT;

  #define LOWORD(x) ((unsigned int) (x))
  #define HIBYTE(x) (*((BYTE *)&x+2))

  GDT         GTab;                         /* Global Descriptor Table */
  union REGS  Regs;       /* Registres process. pr appel interruption */
  struct SREGS SRegs;                                /* Registres segment */
  long        Adr;                        /* pour conversion de l'adresse */

  memset( &GTab, 0, sizeof GTab );              /* Tous les champs à 0 */

  /*-- Construction du descripteur du segment Start ------------------*/

  GTab.Start.AdrLo     = LOWORD(Start);
  GTab.Start.AdrHi     = HIBYTE(Start);
  GTab.Start.Attribut  = 0x92;
  GTab.Start.Longueur  = Len;

  /*-- Construction du descripteur du segment But --------------------*/

  GTab.But.AdrLo    = LOWORD(But);
  GTab.But.AdrHi    = HIBYTE(But);
  GTab.But.Attribut = 0x92;
  GTab.But.Longueur = Len;

  /*-- Copie de blocs de mémoire à l'aide de la fonction 0x87 de  -----*/
  /*-- Interruption 0x15 pour cassettes                           -----*/

  Regs.h.ah = 0x87;             /* N° de la fonction 'copier mémoire' */
```

```
 SRegs.es  = (long) (void far *) &GTab >> 16;       /* Adresse du GDT */
 Regs.x.si = (int) &GTab;                           /* après ES:SI */
 Regs.x.cx = Len >> 1;              /* Nombre des mots à copier dans CX */
 int86x( 0x15, &Regs, &Regs, &SRegs );          /* Appeler fonction */
 if( Regs.h.ah )                                        /* Erreur? */
 {                                  /* Oui, AH contient un code d'erreur */
   printf( "\nErreur pendant l'accès à la mémoire étendue (%d)\n",
           Regs.h.ah);
   exit( 1 );             /* Quitter le programme avec le code d'erreur */
 }
}

/*****************************************************************
*  ExtRead : Copie un certain nombre d'octets de la mémoire étendue  *
*            dans la mémoire principale.                             *
*----------------------------------------------------------------*
*  Entrée : ExtAdr = Adresse source (linéaire) dans mémoire étendue  *
*           BuPtr  = Pointeur sur le tampon But dans mémoire principale *
*           Len    = Nombre des octets à copier                     *
*  Sortie : aucune                                                 *
*****************************************************************/

void ExtRead( long  ExtAdr,  void far *BuPtr, WORD Len )
{
  ExtCopy( ExtAdr, ExtAdrConv( BuPtr ), Len );
}

/*****************************************************************
*  ExtWrite : Copie un certain nombre d'octets de la mémoire principale*
*             dans la mémoire étendue.                             *
*----------------------------------------------------------------*
*  Entrée : BuPtr  = Pointeur sur tampon source dans mémoire principale*
*           ExtAdr = Adresse But (linéaire) en mémoire étendue     *
*           Len    = Nombre d'octets à copier                      *
*  Sortie : aucune                                                 *
*****************************************************************/

void ExtWrite( void far *BuPtr, long ExtAdr, WORD Len)
{
  ExtCopy( ExtAdrConv( BuPtr ), ExtAdr, Len );
}

/*****************************************************************
*  ExtGetInfo : Retourne l'adresse de la mémoire étendue et sa taille  *
*               en tenant compte des éventuels disques virtuels de  *
*               type VDISK s'y trouvant                             *
*----------------------------------------------------------------*
*  Entrée : aucune                                                 *
*  Sortie : aucune                                                 *
*  Globals : ExtAvail/W, ExtStart/W, ExtLen/W                      *
*****************************************************************/

void ExtGetInfo( void )
{
 typedef struct {                   /* Secteur boot d'un disque virtuel */
                BYTE dummy1[3];
                char Name[5];
                BYTE dummy2[3];
                WORD BpS;
                BYTE dummy3[6];
                WORD Secteurs;
                BYTE dummy4;
               } BOOT_SECTEUR;

 static char VDiskName[5] = { 'V', 'D', 'I', 'S', 'K' };

 BOOT_SECTEUR BootSek;              /* copie le secteur de boot supposé */
 union REGS   Regs;         /* Registres process. pr appel interruption */

 /*-- Demander la taille de la mémoire étendue et en déduire la -----*/
 /*-- la présence éventuelle de mémoire étendue                  -*/

 Regs.h.ah = 0x88;                  /* n° fonc. : "renvoyer taille XMS" */
 int86( 0x15, &Regs, &Regs );       /* Appeler interruption Cassette */
 if( Regs.x.ax == 0 )
 {                                          /* Pas de RAM étendue */
   ExtAvail = FALSE;
   ExtLen   = ExtStart = 0;
   return;                                  /* Retour à l'appelant */
 }

 ExtAvail = TRUE;                           /* Mémoire étendue disponible */
 ExtLen   = Regs.x.ax;          /* copier sa taille dans une variable */

 /*-- Rechercher disques virtuels type VDISK -------------------------*/

 ExtStart = 0x100000L;                      /* Si 1 Mo, lancer */

 while( TRUE )                                  /* boucle interrogation */
 {
   ExtRead( ExtStart, &BootSek, sizeof BootSek );
   if( memcmp( BootSek.Name, VDiskName,
       sizeof VDiskName ) == 0 )   /* Secteur boot de disque virtuel? */
     ExtStart += (long) BootSek.Secteurs * BootSek.BpS;       /* Oui */
   else
     break;                                     /* Quitter boucle */
 }

 /*-- Soustraire tailles des disques virtuels de la mémoire ---------*/
 /*-- étendue disponible ---------------------------------------------*/

 ExtLen -= (int) ((ExtStart - 0x1000001) >> 10);
}

/*****************************************************************
*  CheckExt : Vérifie que la mémoire étendue disponible d'un seul bloc *
*----------------------------------------------------------------*
*  Entrée : aucune                                                 *
*  Sortie : aucune                                                 *
*****************************************************************/

void CheckExt( void )
{
 long  AdrTest;                             /* Adresse du bloc test */
 int   i, j;                                /* Compteur d'itérations */
 BYTE  WriteBuf[1024],                              /* Blocs tests */
       ReadBuf[1024];
 BOOL  Erreur = FALSE;                      /* Pointeur erreur mémoire */

 randomize();                       /* Initialiser générateur nbres aléat. */
 AdrTest = ExtStart;
 for( i = 1; i <= ExtLen; ++i, AdrTest += 1024 )
 {                                          /* Lire mémoire par blocs de 1 Ko */
   for ( j = 0; j < 1024; )         /* Remplir bloc de nbres aléatoires */
     WriteBuf[ j++ ] = random( 255 );

   printf("\r%ld", AdrTest );               /* Adresse du bloc testé */

           /*-- Copier dans tampon WriteBuf et ensuite lire ReadBuf -*/

   ExtWrite( WriteBuf, AdrTest, 1024 );
   ExtRead( AdrTest, ReadBuf, 1024 );

     /*-- Déterminer l'identité de WriteBuf et de ReadBuf ------------*/

   for( j = 0; j < 1024; ++j )  /* Remplir bloc avec nbres aléatoires */
     if( WriteBuf[j] != ReadBuf[j] )    /* Contenu tampon identique? */
     {                                                  /* Non, erreur! */
       printf( "\n  Erreur! Adresse %ld\n", AdrTest + j - 1);
       Erreur = TRUE;
     }
 }                                                      /* définir */
 printf( "\n\n" );
 if( !Erreur )                                  /* Erreur apparue ? */
   printf( "C'est bon !\n" );                           /* Non */
}

/*****************************************************************
*  P R O G R A M M E   P R I N C I P A L                          *
*****************************************************************/

void main( void )
{
 printf ("EXTC - (c) 1989, 92 by Michael Tischer\n\n");
 ExtGetInfo();      /*Donne disponibilité et taille de mémoire étendue */
 if( ExtAvail )                             /* Mémoire étendue es-tu là ? */
 {                                                          /* Oui! */
   RdLen = (int) ( (ExtStart - 0x1000001 ) >> 10 );
   if( RdLen == 0 )                         /* RAM disks installés? */
     printf( "Aucun disque virtuel installé.\nLa mémoire étendue"\
             " disponible commence à la limite du 1er Mo.\n" );
   else                             /* Mais oui, il y a des RAM disks ! */
     printf("Un ou plusieurs RAM disks occupent %d Ko de la mémoire "\
            "étendue.\nLa mémoire étendue libre commence %d Ko après"\
            "la limite du 1er Mo.\n", RdLen, RdLen);
   printf( "Taille de la mémoire étendue libre : %d Ko\n", ExtLen);
   printf( "\nTest de la continuité de la mémoire étendue en"\
           " cours...\n\n" );
   CheckExt();
 }
 else
   printf( "Pas de mémoire étendue dans votre ordinateur !\n" );
}
```

12.2.2. Conflits dans la mémoire étendue

"La mémoire étendue se tient à la disposition de tout le monde", ainsi pourrait s'énoncer le slogan de ce chapitre. En effet, on a rarement vu un programme s'accaparer exclusivement de la mémoire étendue constituée de plusieurs Mo. Les problèmes surviennent surtout lorsqu'on lance un programme Cache ou d'autres utilitaires en plus d'un disque RAM. Les raisons sont généralement à rechercher dans un manque de contrôle lors de l'accès à la mémoire étendue, conduisant les programmes à écraser mutuellement leurs données et provoquant ainsi un plantage.

Mais il ne faut pas croire que des utilitaires inoffensifs se transforment brusquement en programmes destructeurs dès qu'ils se rencontrent dans la mémoire étendue. Le coupable est évidemment le BIOS qui ne prévoit aucune technique visant à allouer les différents blocs de mémoire. Bien au contraire, avec la fonction 88h, il incite chaque programme à disposer de la totalité de la mémoire étendue.

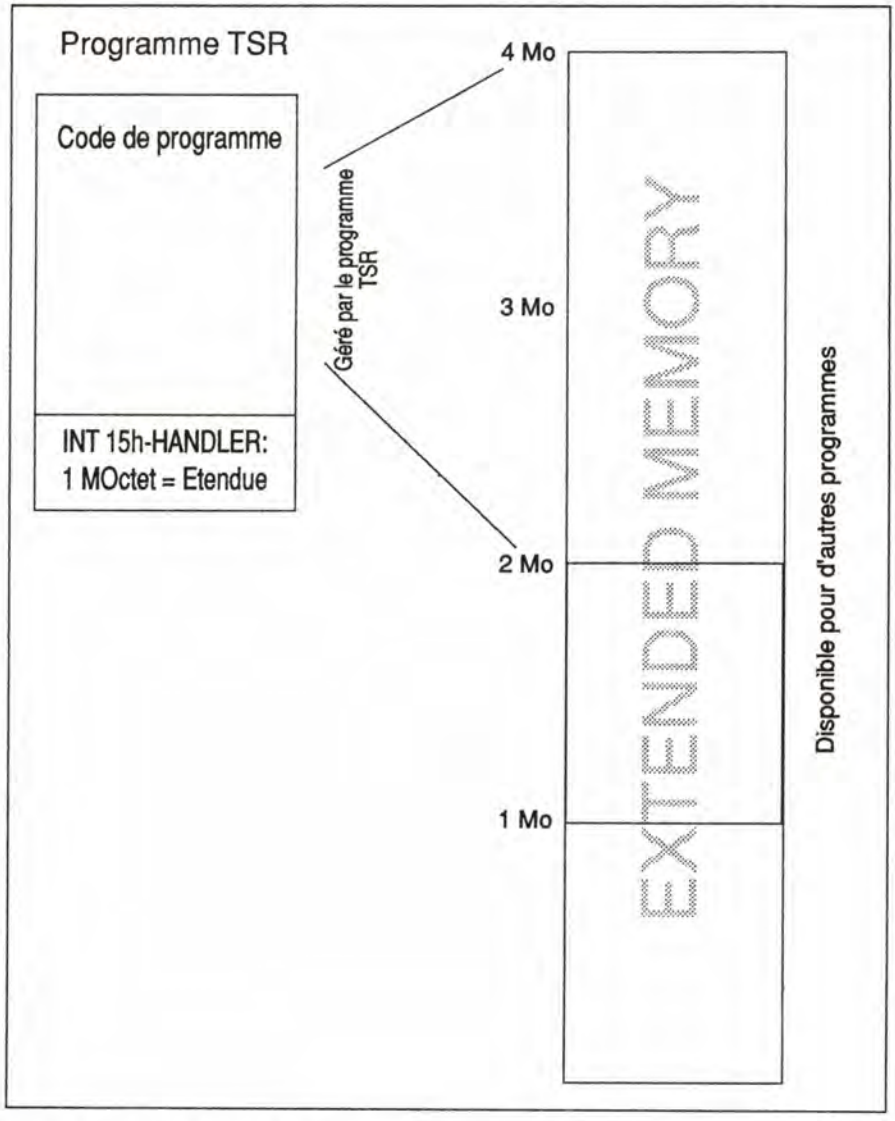
De ce point de vue, le standard XMS, faisant l'objet de la section suivante, permet de résoudre la situation. Il est apparu en 1988 à une époque où il existait déjà de nombreux utilitaires utilisant la mémoire étendue à des fins personnelles. Deux procédures différentes se sont développées autour du standard XMS. Elles permettent d'éviter la collision dans la mémoire étendue, mais cela reste encore insuffisant.

La première méthode, la plus performante, consiste à intervenir dans le fonctionnement de l'interruption 15h. A cet effet, un programme redirige le vecteur d'interruption de cette interruption dans la table des vecteurs d'interruption de telle manière qu'il ne pointe plus vers le handler d'interruption d'origine dans la ROM du BIOS, mais vers un handler personnel. Cette méthode doit d'ailleurs son nom à cette manière de procéder puisqu'elle est désignée par la méthode "INT 15".

Le nouveau handler de l'interruption 15h ne doit pas se servir de toutes les fonctions de l'interruption, mais seulement de la fonction 88h. Elle permet de déterminer la taille de la mémoire étendue. C'est pourquoi, il redirige l'appel de l'interruption vers le handler d'origine dès qu'il apprend -grâce au numéro de fonction- qu'une autre fonction doit être appelée.

Si l'appel concerne précisément la fonction 88h, elle retourne la taille de la mémoire étendue encore libre et non la taille réelle. La taille de la mémoire occupée découle tout simplement de la taille réelle. Ainsi, le programme d'appel ne peut plus s'approprier la totalité de la mémoire étendue puisqu'il ne dispose que de la zone disponible.

Etant donné que la mémoire étendue commence, comme précédemment, à la limite des 1 Mo, le programme installé préalablement doit se protéger contre le programme d'appel. A cet effet, il utilise non seulement la mémoire à partir des 1 Mo, mais à partir de la mémoire étendue. Du point de vue du programme d'appel, cette zone se situe notamment après la fin de la mémoire étendue si bien qu'il ne vient pas la déranger.



Utilisation de la mémoire étendue en redirigeant l'interruption 15h

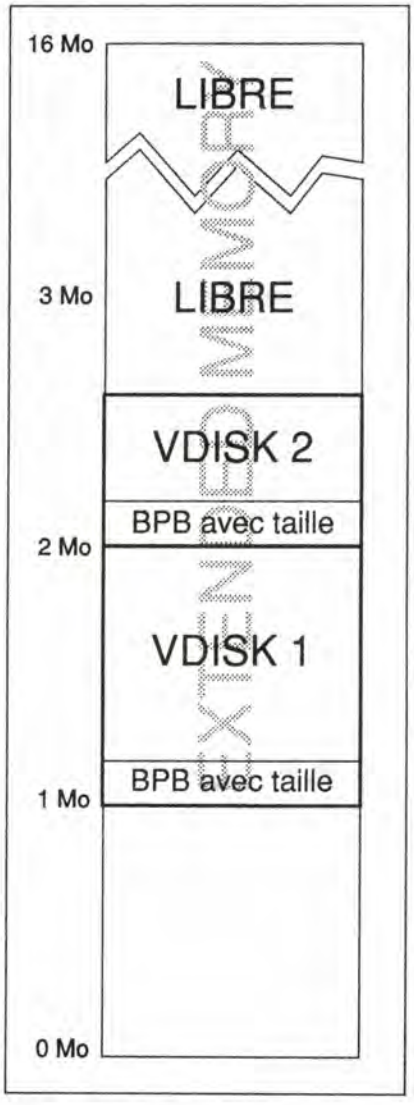
Si tous les programmes adoptaient cette procédure, il ne faudrait plus craindre des problèmes quant à l'utilisation totale de la mémoire étendue. Malheureusement, il existe une autre méthode qui complique à vrai dire l'ordre des choses.

Dans la littérature spécialisée, cette méthode est désignée par la méthode VDISK parce qu'elle a été d'abord introduite dans le driver de périphérique VDISK. Il s'agit d'un disque RAM utilisable par la mémoire étendue pour stocker les fichiers. A partir de la version 3.0 de PC DOS, il était fourni avec toutes les versions DOS. VDISK ne se donne pas

la peine de se soucier du détournement de l'interruption 15h opéré par les autres programmes, mais place directement les données dans la mémoire étendue à partir de la limite de 1 Mo.

VDISK ne réécrit pas la mémoire protégée par les programmes lancés précédemment à l'aide de la méthode INT 15. Mais les programmes appelés ultérieurement écrase irrémédiablement le disque RAM s'ils ne découvrent pas au préalable son existence et utilisent la mémoire située après sa fin. Tout cela se réalise grâce à un contrôle approfondi nécessitant une connaissance poussée de la structure d'un support mémoire sous DOS. La tête d'un tel disque RAM présente une structure très précise propre à toutes les mémoires de masse reconnues sous DOS.

Outre des informations d'état, l'en-tête contient une petite structure de données appelée bloc de paramètres. Il permet de calculer la taille d'un support de mémoire et, dans le cas d'un VDISK, la taille du RAM Disk dans la mémoire étendue.



Structure de la mémoire étendue avec un RAM disk de type VDISK

L'affaire n'est pas si simple que cela. Dans certains cas, plusieurs disques RAM se suivent dans la mémoire et il faut par conséquent consulter plusieurs disques RAM avant d'arriver au début de la mémoire étendue libre.

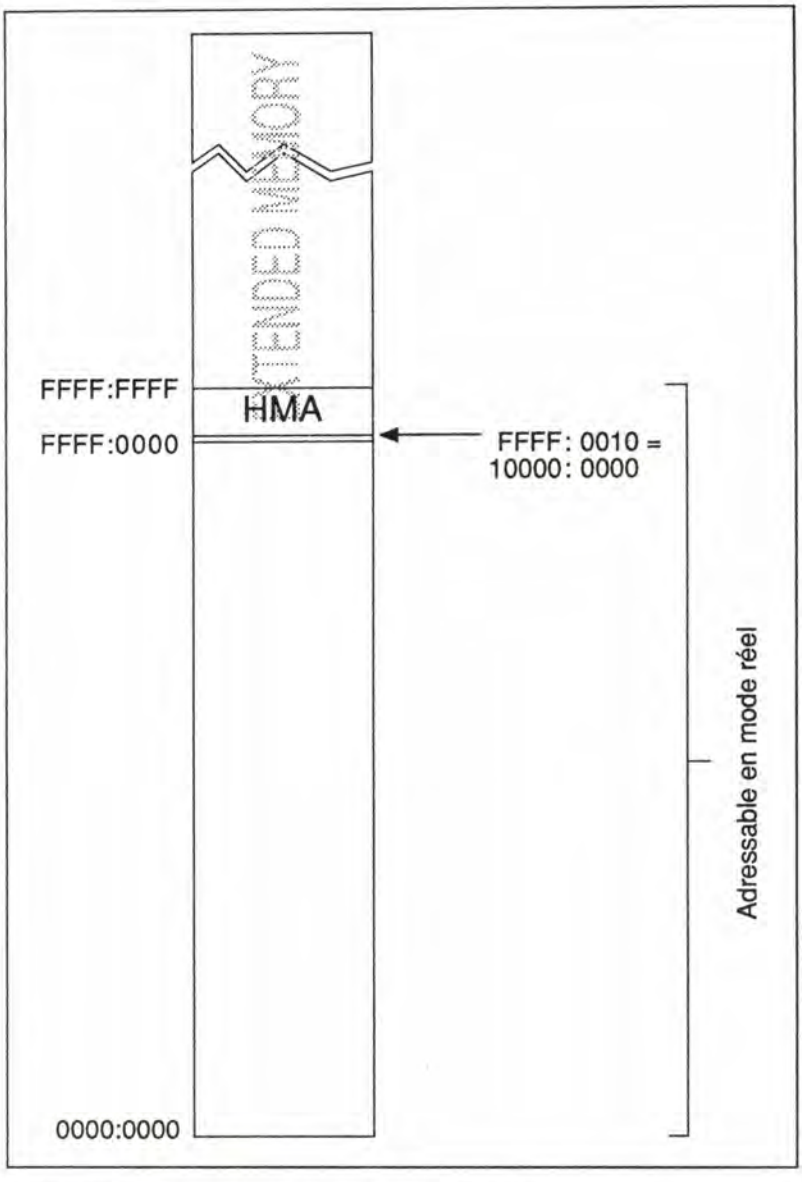
Comme cette procédure semble particulièrement complexe et ne fonctionne d'ailleurs pas sur tous les types de disques RAM convenant à la mémoire étendue, nous vous conseillons d'utiliser un driver XMS pour accéder à la mémoire étendue. Ce driver résout tous ces problèmes comme vous l'apprendrez dans les sections suivantes.

12.2.3. Accès direct à la HMA à partir du mode réel

La HMA (High Memory Area) représente les 64 premiers Ko de la mémoire étendue. Ils ont eu droit à un nom spécifique car ils correspondent à la seule et unique partie de la mémoire étendue pouvant être réclamée à partir du mode réel avec une petite astuce sans commuter vers le mode protégé.

La découverte de cette astuce est le fruit de la société Microsoft. Elle a introduit pour la première fois la HMA dans la version Windows 2.1 afin de mieux répondre à la gourmandise mémoire de cette interface utilisateur.

L'allocation de 64 Ko au mode réel constitue effectivement une découverte qu'il convient d'examiner à la loupe. Comment s'effectue l'accès aux cellules de mémoire situées à l'extérieur de la zone d'adressage du processeur ?



Limite des 1 Mo franchie en mode réel à travers le segment FFFFh

La réponse est relativement simple et dépend de la manière dont le 80286 et les modèles suivants construisent des adresses physiques en mode réel à partir d'une adresse de segment et d'offset. Comme vous le savez sans doute, les adresses de segment et d'offset sont tout simplement additionnées en multipliant préalablement l'adresse de segment par 16. En guise d'adresse de segment, si on précise maintenant le dernier segment de la zone d'adressage du PC, le segment FFFFh, on aboutit alors à l'adresse d'offset 000Fh parmi les adresses physiques situées au-delà de la limite de 1 Mo. On se trouve d'ores

et déjà dans la mémoire étendue bien que le processeur fonctionne en mode réel. De plus, la totalité de la mémoire peut être adressée sans problème au-dessous de la limite de 1 Mo.

Etant donné que la HMA ne commence pas à l'adresse d'offset 0000h mais 00010h, sa taille ne s'élève pas précisément à 64 Ko, mais comporte 16 octets de moins. Elle comporte exactement 65520 octets qui sont amplement suffisants pour y stocker des données ainsi que de petits utilitaires tels que les programmes TSR.

Etapes pour accéder à la HMA

Avant d'accéder à la HMA, le BIOS a pris soin d'activer la commutation vers la ligne d'adresse A20. Normalement, cette ligne est verrouillée parce que certaines applications PC ont tendance à franchir la limite de 1 Mo. En principe, depuis l'adresse FFFF:0010, on peut ainsi atteindre à nouveau le début de la mémoire principale, c'est-à-dire l'adresse 0000:0000.

Débordement d'adresse par désactivation de la ligne A20

Bien qu'il n'ait rien à voir dans ce contexte, le contrôleur de clavier prend une importance significative dans le cadre de la libération de la ligne d'adresse A20. Le phénomène ressemble à celui rencontré lors de la réactivation du processeur à partir du mode protégé.

En fait, cela se passe comme lors du reset du système où le port de sortie de ce contrôleur permet de bloquer ou libérer la ligne d'adresse A20. Le responsable de cette action est le bit 1 de ce port. Il doit être réglé pour rendre la ligne d'adresse A20 accessible et remis à 0 si un saut doit être effectué vers le début de la mémoire au niveau de la frontière 1 Mo.

Cependant, il n'est pas du tout facile d'arriver jusqu'à ce bit. Il faut d'abord envoyer une commande au clavier autorisant l'accès au port de sortie. Par ailleurs, il convient de respecter des règles strictes liées à la communication avec le contrôleur. Ainsi, il ne faut pas se contenter d'envoyer deux octets au port I/O correspondant. Nous n'étudierons pas ici en détail le fonctionnement de la communication.

Avant de s'aventurer à libérer la ligne d'adresse A20, il convient de s'assurer qu'un accès à la HMA est bel et bien autorisé. A cet effet, il faut d'abord déterminer le type du processeur. Après avoir obtenu la confirmation qu'un 80286 ou un modèle ultérieur est effectivement installé sur le système, on peut se permettre de spécifier une ligne d'adresse A20 et par voie de conséquence une mémoire supérieure à 1 Mo.

Si le processeur a effectué ce test, il faut déterminer la taille de la mémoire étendue avec la fonction BIOS 88h de l'interruption de cassettes 15h. Un accès à la HMA n'est autorisé que s'il existe au moins 64 Ko.

Cela pourrait très bien commencer avec la libération de la ligne d'adresse A20, mais on doit tout de même s'assurer que la ligne est effectivement libérée une fois le bit correspondant réglé dans le port de sortie du clavier. Le port de sortie du clavier ne constitue le domicile du bit contrôlant l'état de la ligne d'adresse A20 que dans les PC équipés d'un bus ISA (Industry Standard Architecture). Quant aux PS/2 et AT qui ont été adaptés au i386 au moyen d'un InBoard INTEL, il faut suivre une autre démarche pour libérer cette ligne.

Par conséquent, un test aiderait à confirmer l'état de la ligne d'adresse A20. Même si cette ligne n'est pas libre, tout accès à la HMA reste pourtant interdit. Une petite astuce aide néanmoins à éclaircir la situation : lorsque la ligne est verrouillée, les zones mémoire comprises par exemple entre FFFF:0010 et 0000:0000 sont identiques en raison du débordement au-delà des 1 Mo. Etant donné que la comparaison a porté tout simplement sur les 256 premiers octets de ces zones, il est possible d'établir une concordance entre ces derniers et prouver ainsi le verrouillage de l'A20.

Trois routines sont à utiliser avec la HMA :

- ✓ une routine pour démontrer l'existence d'une mémoire étendue de 64 Ko minimum,
- ✓ une routine pour actionner la ligne d'adresse A20,
- ✓ une routine pour contrôler la ligne d'adresse A20 en effectuant une comparaison de mémoire.

Programmes de démonstration

Les deux programmes d'exemple sont écrits en Pascal et C et offrent un cadre intéressant pour expérimenter vos connaissances en HMA. Sachant qu'à l'avenir l'accès à la HMA ne doit s'effectuer qu'à travers le driver XMS (décrit dans la section suivante), le programme présente une petite insuffisance : il ne contrôle pas l'existence d'un disque RAM de type VDISK. Un programme de ce genre ne doit d'ailleurs jamais présenter un tel défaut car l'accès à la HMA peut tout simplement provoquer la destruction du disque RAM. Avant de lancer ces programmes, n'oubliez donc pas de vérifier qu'un disque RAM ou un programme Cache n'est pas installé dans la mémoire étendue.

Les deux programmes utilisent essentiellement les routines citées plus haut. Elles portent les noms HMAAvail, Gate20 et IsA20On. Elles ne peuvent pas être conçues dans un langage évolué mais uniquement en Assembleur. Pour le programme C, elles sont

contenues dans un module Assembleur individuel (HMACA.ASM). Une fois assemblé avec le programme compilé C (HMAC.C), il doit être fusionné.

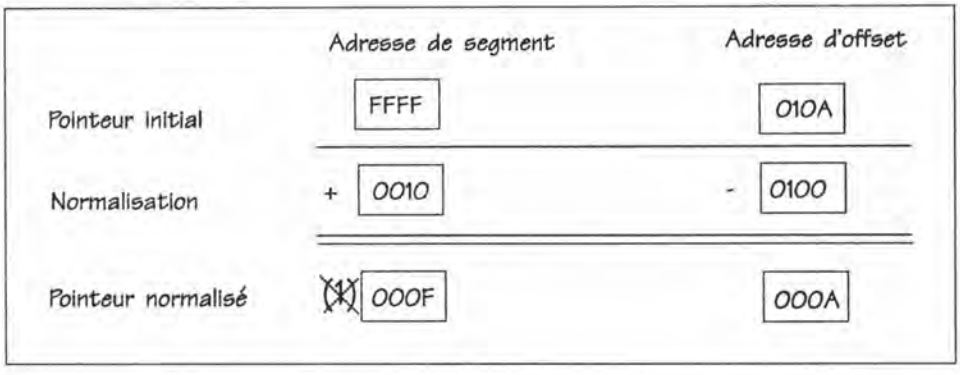
Dans la version Pascal du programme (HMAP.PAS), il n'est pas nécessaire d'avoir recours au module Assembleur car dans ce cas, les routines Assembleur sont intégrées directement dans le programme Pascal via les instructions INLINE. Cela permet d'accélérer considérablement les modifications futures, mais la manipulation devient plus souple puisque l'Assembleur s'avère inutile et toutes les routines en Pascal et en Assembleur sont regroupées dans un seul fichier.

Les deux programmes essayent d'abord d'accéder à la HMA et renvoient un message d'erreur quand cela est impossible. Un pointeur se place ensuite sur le début de la HMA. Il doit naturellement être de type FAR, ce qui est du moins le cas des programmes Pascal sous Turbo Pascal. Dans le programme C, le pointeur est déclaré en tant que tel au moyen du mot-clé FAR.

Si l'adresse FFFF:0010 est affectée au pointeur, cela signifie qu'un accès à la HMA peut avoir lieu à n'importe quel moment. Les deux programmes utilisent cette méthode pour effectuer un test de mémoire simple où les diverses cellules de mémoire de la HMA sont écrites avec une valeur, puis sont lues et les erreurs prises en compte. A vrai dire, il est rare de rencontrer des erreurs car le BIOS n'aurait pas tardé à signaler l'échec d'une fonction des Chips RAM correspondants dès le lancement du système.

La ligne d'adresse A20 est à nouveau verrouillée avant la fin des programmes interdisant ainsi l'accès à la HMA. Cette opération doit avoir lieu dans tous les cas parce que certains programmes sont amenés à franchir la limite des 1 Mo et on ne doit pas les en empêcher.

Pour vos connaissances personnelles en matière de HMA, il convient de souligner deux lignes directives à respecter obligatoirement. D'une part, vous devez éviter d'utiliser des routines DOS et des routines issues de la bibliothèque du compilateur pour transmettre des pointeurs conduisant à la HMA. Dans les deux cas, les routines concernées tendent à normaliser les pointeurs transmis.



Normalisation des pointeurs HMA

Il s'agit là d'un processus où la partie segment d'un pointeur est tellement modifiée que la partie offset est inférieure à 16. Mais comme auparavant, c'est la cellule mémoire spécifiée à l'origine qui est adressée. En ce qui concerne les cellules de mémoire de la HMA, cela provoque un débordement de l'adresse de segment qui se fixe sur 0000h. Il est donc absolument nécessaire d'empêcher la normalisation des pointeurs HMA.

D'autre part, il ne faut effectuer aucune opération de disquette ou disque dur adressant directement la HMA. Ici, le logiciel joue parfois un mauvais tour au contrôleur DMA qui ne sait pas s'en sortir correctement avec les adresses situées au-delà des 1 Mo.

Si vous respectez ces deux directives, vous ne rencontrerez aucune difficulté à accéder directement à la HMA. Mais n'oubliez pas que les deux programmes d'exemple ne cherchent à démontrer que le principe. Dans les programmes professionnels, vous devrez utiliser le driver XMS pour accéder à la HMA. La section suivante décrit les fonctions de ce driver.

Listing : HMAP.PAS

```
{*********************************************************************
*                          H M A P . P A S                           *
**-------------------------------------------------------------------**
* Sujet       : Démonstration d'accès direct au HMA sans            *
*               l'assistance d'un pilote spécial.                   *
**-------------------------------------------------------------------**
* Auteur          : MICHAEL TISCHER                                 *
* Développé le    : 27.07.1990                                      *
* Dernière m. à j : 29.07.1990                                      *
*********************************************************************}

program HMAP;

uses Crt;                                              { pour ClrScr }

{*********************************************************************
* HMAAvail : Détecte la présence d'un processeur 80286 ou suivant et *
*            la présence d'au moins 64 Ko de mémoire étendue         *
**-------------------------------------------------------------------**
* Entrée : aucune                                                   *
* Sortie : TRUE si HMA disponible, sinon FALSE                      *
* Info   : - Cette fonction doit être appelée avant l'appel de toutes *
*            les autres procédures et fonctions de ce programme.    *
*********************************************************************}

function HMAAvail : boolean;

begin
  inline (
        $33/$C0/                        { xor   ax,ax           *)
        $50/                            { push  ax              *)
        $90/                            { popf                  *)
        $9C/                            { pushf                 *)
        $58/                            { pop   ax              *)
        $25/$00/$F0/                    { and   ax,0F000h       *)
        $30/$00/$F0/                    { cmp   ax,0F000h       *)
        $74/$0E/                        { je    pashma > *)
        $B4/$88/                        { mov   ah,88h          *)
        $CD/$15/                        { int   15h             *)
        $30/$40/$00/                    { cmp   ax,64           *)
        $72/$05/                        { jb    pashma > *)
        $B8/$01/$00/                    { mov   ax,0001h        *)
        $EB/$02/                        { jmp   ende            *)
        $33/$C0/                        { xor   ax,ax < *)
        $88/$46/$FF                     { mov   [bp-1],al       *)
      );
end;

{*********************************************************************
* GateA20 : Bloque ou libère le canal d'adresses A20                 *
**-------------------------------------------------------------------**
* Entrée : libre = TRUE s'il faut libérer le canal                  *
* Sortie : TRUE si l'accès au contrôleur de clavier réussit, sinon  *
*          FALSE                                                    *
* Info   : - Après l'appel de cette fonction, il est recommandé de  *
*            s'assurer à l'aide de la fonction IsA200n si le canal a *
*            réellement été libéré car le procédé utilisé ici ne    *
*            libère le canal que sur les machines équipées du bus ISA *
*********************************************************************}

function GateA20( libre : boolean ) : boolean;

begin
  inline (
        $84/$DD/                        { mov   ah,11011101b    *)
        $83/$7E/$04/$00/                { cmp   FREI,0          *)
        $74/$02/                        { je    gl      *)
        $84/$DF/                        { mov   ah,11011111b    *)
        $33/$C9/                        { xor   cx,cx <     *)
        $FA/                            { cli                   *)
        $E4/$64/                        { in    al,64 <     *)
        $A8/$02/                        { test  al,02           *)
        $ED/$FA/                        { loopnz        *)
        $75/$1D/                        { jne   gerr > *)
        $80/$D1/                        { mov   al,WO_COMMAND   *)
        $E6/$64/                        { out   KB_COMMAND,al   *)
        $E4/$64/                        { in    al,64 <     *)
        $A8/$02/                        { test  al,02           *)
        $ED/$FA/                        { loopnz        *)
        $75/$11/                        { jne   gerr > *)
        $8A/$C4/                        { mov   al,ah           *)
        $E6/$60/                        { out   KB_DATA,al      *)
        $E4/$64/                        { in    al,64 <     *)
        $A8/$02/                        { test  al,02           *)
        $ED/$FA/                        { loopnz        *)
        $75/$05/                        { jne   gerr > *)
        $B8/$01/$00/                    { mov   ax,0001h        *)
        $EB/$02/                        { jmp   ende            *)
        $33/$C0/                        { xor   ax,ax < *)
        $FB/                            { sti                   *)
        $88/$46/$FF                     { mov   [bp-1],al       *)
      );
end;

{*********************************************************************
* IsA200n : Indique si le canal d'adresses A20 est libéré            *
**-------------------------------------------------------------------**
* Entrée : aucune                                                   *
* Sortie : TRUE si le canal est libre, FALSE sinon                  *
*********************************************************************}

function IsA200n : boolean;

begin
  inline (
        $1E/                            { push  ds              *)
```

```
        $06/                                    { push   es             *)
        $33/$F6/                                { xor    si,si          *)
        $8E/$0E/                                { mov    ds,si          *)
        $BF/$10/$00/                            { mov    di,0010        *)
        $B8/$FF/$FF/                            { mov    ax,FFFF        *)
        $8E/$C0/                                { mov    es,ax          *)
        $B9/$40/$00/                            { mov    cx,64          *)
        $FC/                                    { cld                   *)
        $F3/$A7/                                { repe   cmpsw          *)
        $07/                                    { pop    es             *)
        $1F/                                    { pop    ds             *)
        $E3/$05/                                { jcxz   a20off  *}
        $B8/$01/$00/                            { mov    ax,0001h       *)
        $EB/$02/                                { jmp    ende           *)
        $33/$C0/                                { xor    ax,ax  < *}
        $B8/$46/$FF                             { mov    [bp-1],al      *)
      );
end;

{*********************************************************************
* HMATest : Démonstration de l'accès au HMA                          *
**--------------------------------------------------------------------**
* Entrée : aucune                                                    *
*********************************************************************}

procedure HMATest;

type HMAR    = array [1..65520] of BYTE;              { le tableau HMA }
     HMARPTR = ^HMAR;                        { pointeur sur le tableau HMA }

var hmap  : HMARPTR;                                    { pointeur sur HMA }
    i,                                                { Compteur d'itérations }
    err   : word;                { Nombre d'erreurs pendant l'accès au HMA }
    dummy : boolean;

begin
  if ( IsA20On ) then
    writeln( 'Le canal A20 est déjà libéré !' )
  else
    if ( GateA20( TRUE ) = FALSE ) or ( IsA20On = FALSE ) then
      begin
        writeln( 'Attention! Le canal d''adresses A20 n''a pu être' +
                 'libéré.' );
        exit;
      end
  else
    writeln( 'L''accès à l''HMA vient d''être libéré.' );

hmap := HMARPTR(Ptr( $FFFF, $0010 ));                  { pointeur sur HMA }

err := 0;                                        { Aucune erreur jusqu'à présent)
for i := 1 to 65520 do                        { Teste chaque adresse séparément }
  begin
    write( #13, 'Adresse: ', i );
    hmap^[i] := i mod 256;                          { Ecrire dans l'adresse }
    if ( hmap^[i] <> i mod 256 ) then                       { et la relire }
      begin                                                    { Erreur! }
        writeln( ' ERREUR!' );
        inc( err );
      end;
  end;

writeln( #13 );
if ( err = 0 ) then                          { Evaluation du résultat du test }
  writeln( 'HMA ok, aucune adresse incorrecte.')
else
  writeln( 'ATTENTION! ', err, ' Des adresses incorrectes ont été ' +
           'découvertes dans le HMA !');

dummy := GateA20( FALSE );                      { Désactiver canal d'adresses }
end;

{*********************************************************************
*                    P R O G R A M M E   P R I N C I P A L           *
*********************************************************************}

begin
writeln( 'HMAP  -  Programme démo HMA par MICHAEL TISCHER'#10 );
if HMAAvail then
  begin
    HMATest;                                                  { test HMA }
    writeln;
  end
else
  writeln( 'Aucun accès possible à l''HMA.' );
end.
```

Listing : HMAC.C

```
/*********************************************************************
*                            H M A C . C                             *
**--------------------------------------------------------------------**
*  Sujet          : Démonstration d'accès direct au HMA sans l'aide   *
*                   d'aucun pilote particulier.                       *
**--------------------------------------------------------------------**
*  Auteur         : MICHAEL TISCHER                                   *
*  Développé le   : 27.07.1990                                        *
*  Dernière m. à j: 29.07.1990                                        *
**--------------------------------------------------------------------**
*  (MICROSOFT C)                                                      *
*  Création       : CL /AS /Zp hmac.c hmaca                           *
*  Appel          : hmac                                              *
**--------------------------------------------------------------------**
*  (BORLAND TURBO C)                                                  *
*  Création       : avec un fichier de projet contenant les fichiers  *
*                     hmac.c                                          *
*                     hmaca.obj                                       *
*********************************************************************/

/*-- Intégrer les fichiers include ---------------------------------*/

#include <dos.h>                              /* Pour appels interruptions */
#include <stdio.h>

#ifdef __TURBOC__
  #include <alloc.h>
#else
  #include <malloc.h>
#endif

/*-- Constantes ----------------------------------------------------*/


#define TRUE  ( 0 == 0 )
#define FALSE ( 0 == 1 )

/*-- Macros --------------------------------------------------------*/

#ifndef MK_FP
  #define MK_FP(seg,ofs) \
          ((void far *) (((unsigned long)(seg) << 16) | (unsigned)(ofs)))
#endif

#define Hi(x) (*((BYTE *) &x+1))                    /* Hi Byte d'un int */
#define Lo(x) (*((BYTE *) &x))                      /* Lo Byte d'un int */

/*-- Déclarations des types ----------------------------------------*/

typedef unsigned char BYTE;
typedef BYTE BOOL;
typedef unsigned WORD;

/*-- Déclarations externes -----------------------------------------*/

extern BOOL HMAAvail( void );                        /* HMA dispo ? */
extern BOOL GateA20( BOOL libre );                /* libérer/bloquer A20 */
extern BOOL IsA20On( void );                         /* A20 libéré ? */

/*********************************************************************
* HMATest : Démonstration de l'accès au HMA                          *
**--------------------------------------------------------------------**
* Entrée : aucune                                                    *
*********************************************************************/

void HMATest( void)

{
  BYTE far * hmap;                                  /* Pointeur sur HMA */
  WORD i,                                         /* Compteur d'itérations */
       err;                                /* Nombre d'erreurs en accès à l'HMA */

  if( IsA20On() )
    printf( "Le canal d'adresse A20 est déjà libéré!\n" );
  else
    if( GateA20( TRUE ) == FALSE  || IsA20On() == FALSE )
    {
      printf( "Attention! Le canal d'adresses A20 n'a pu être" \
```

```
                "libéré." );
        return;
    }
    else
        printf( "L'accès à l'HMA vient d'être libéré.\n" );

hmap = MK_FP( 0xFFFF, 0x0010 );                 /* pointeur sur HMA */
err  = 0;                                       /* aucune erreur jusqu'ici */
for( i = 1; i < 65520; ++i, ++hmap )
{                                               /* Teste chaque adresse séparément */
    printf( "\rAdresse: %u", i );
    *hmap = i % 256;                            /* Ecrire dans l'adresse */
    if( *hmap != i % 256 )                      /* et la relire */
    {                                           /* Erreur! */
        printf( " ERREUR!\n" );
        ++err;
    }
}

printf( "\n" );
if ( err == 0 )                                 /* Evaluer résultat du test */
    printf( "HMA ok, aucune adresse incorrecte.\n" );
else
    printf( "ATTENTION! %d adresses incorrectes découvertes " \
            "dans le HMA!\n", err );

GateA20( FALSE );                               /* Désactiver canal d'adresses */
}

/*****************************************************************
*                    P R O G R A M M E   P R I N C I P A L       *
*****************************************************************/

void main( void )
{
  int   i;                                      /* Compteur d'itérations */

  for( i = 1; i < 25; ++i )                     /* Effacer l'écran */
    printf ( "\n" );

  printf("HMAC  -  Programme démo HMA par MICHAEL TISCHER\n\n" );
  if( HMAAvail() )
  {
    HMATest();                                  /* Test HMA */
    printf( "\n" );
  }
  else
    printf( "Aucun accès possible à l'HMA.\n" );
}
```

Listing : HMACA.ASM

```
;*****************************************************************
;*                          H M A C A . A S M                    *
;*---------------------------------------------------------------*
;*  Sujet       : Routine en assembleur à intégrer au programme C *
;*                HMAC.C. Contient une routine d'interrogation   *
;*                de la mémoire étendue et de libération/blocage *
;*                du canal d'adresses A20.                       *
;*                Cette implémentation est destinée au modèle    *
;*                de mémoire SMALL.                              *
;*---------------------------------------------------------------*
;*  Auteur      : MICHAEL TISCHER                                *
;*  Développé le: 28.07.1990                                     *
;*  Dernière m. à j : 28.07.1990                                 *
;*---------------------------------------------------------------*
;*  assembler   : MASM HAMCA;   ou   TASM HMACA;                 *
;*                ... et ensuite lier au programme C HMAC.C      *
;*                compilé.                                       *
;*****************************************************************

;== Constantes ================================================

KB_COMMAND equ 64h                      ;Port commandes clavier
KB_STATUS  equ 64h                      ;est aussi port d'état
KB_DATA    equ 60h                      ;Port données du clavier
IB_FREE    equ 2                        ;Masque pour test d'Input Buf vide
WO_COMMAND equ 0D1h                     ;Commande d'écriture de la Output Port

GATE_ON    equ 11011111b                ;Code de canal adresse A20 libre
GATE_OFF   equ 11011101b                ;Code de canal adresse A20 bloqué

;== Déclarations de segments pour le programme en C ===========

IGROUP group _text                      ;Résumé des segments programmes
DGROUP group _bss, _data                ;Résumé des segments données
        assume CS:IGROUP, DS:DGROUP, ES:DGROUP, SS:DGROUP

_BSS    segment word public 'BSS' ;ce segment récupère ttes les variables
_BSS    ends                            ;statiques non initialisées

_DATA   segment word public 'DATA'      ;Toutes les variables initialisées
                                        ;globales et statiques sont regroupées
                                        ;dans ce segment
_DATA   ends

;== Programme =================================================

_TEXT   segment byte public 'CODE'      ;le segment programme

public    _HMAAvail
public    _GateA20
public    _IsA20On

;---------------------------------------------------------------
;-- HMAAvail : indiquere si le HMA existe
;-- Appel du C: BOOL HMAAvail( void )
;-- Valeur de retour : TRUE, si HMA disponible, sinon FALSE

_HMAAvail  proc near

        ;-- Tester d'abord si AT ou 386 ------------------------

        xor     ax,ax                   ;Ces instructions sont recommandées
        push    ax                      ;par Intel pour détecter le type de
        popf                            ;processeur. Elles prouvent l'existence
        pushf                           ;d'un 8088 ou d'un 8086
        pop     ax
        and     ax,0F000h
        cmp     ax,0F000h
        je      pashma                  ;PC normal PC ---> pashma

        ;-- AT ou 386, mais avec mémoire étendue? --------------

        mov     ah,88h                  ;Taille mémoire étendue par le BIOS
        int     15h                     ;
        cmp     ax,64                   ;Plus de 64 Ko en mémoire étendue ?
        jb      pashma                  ;Non, aucune HMA

        mov     ax,0FFFFh               ;Oui, HMA prêt
        ret                             ;alors on retourne TRUE

pashma:
        xor     ax,ax                   ;Retourner 0 pour FALSE
        ret                             ;retour au programme C

_HMAAvail  endp

;---------------------------------------------------------------
;-- GateA20 : Libérer ou bloquer le canal d'adresses A20
;-- Appel par C: BOOL GateA20( BOOL libre )
;-- Valeur de retour: TRUE si l'opération est réussie, sinon FALSE

_GateA20   proc near

           ;-- Structure pour accéder facilement au paramètre ---

sframe    struc                         ;Structure pour accéder à la pile
bp0       dw ?                                    ;stocke BP
ret_adr   dw ?                          ;Adresse de retour à l'appelant
libre     dw ?                          ;bloque ou libère le canal
sframe    ends                                    ;fin de la structure

frame equ [ bp - bp0 ]                  ;adresse les éléments de la structure

        ;-- Macro demandant l'état du contrôleur du clavier ------

kbc_ready  macro                        ;Contrôleur clavier prêt ?
        local  notready

notready:
        in      al,KB_STATUS                    ;Interroge port d'état
        test    al,IB_FREE                      ;Tampon Input vide ?
        loopne notready                 ;Non, et CX encore différent de zéro -->
```

```
        endm

    ;-- Voici le début du code de la procédure ----------------------

    push   bp                                   ;Stocke BP sur la pile
    mov    bp,sp                                  ;Copier SP après BP

    mov    ah,11011101b                 ;Cette valeur = "bloquer canal"
    cmp    frame.libre,0                                ;bloquer canal?
    je     g1                                      ;Oui, le code est OK.

    mov    ah,11011111b          ;Non, charger le code de "canal libre"

g1: xor    cx,cx                                    ;compteur pour Time Out
    cli                                ;aucune interruption en ce moment
    kbc_ready                                  ;attendre contrôleur clavier
    jne    gerr                                      ;Time Out? oui ---> GERR

    mov    al,WO_COMMAND          ;Envoyer le code d'accès à Output-P.
    out    KB_COMMAND,al                          ;au port d'instruction

    kbc_ready                                  ;attendre contrôleur clavier
    jne    gerr                                      ;Time Out? oui ---> GERR

    mov    al,ah          ;Envoyer instruction de libération/blocage
    out    KB_DATA,al                             ;au port de données

    kbc_ready                                  ;attendre contrôleur clavier
    sti                                              ;Interruption activée
    jne    gerr                                      ;Time Out? oui ---> GERR

    mov    ax,0FFFFh                      ;Pas de Time Out, tout va bien
    pop    bp                                      ;Lire BP sur la pile
    ret

gerr: xor  ax,ax                                  ;aucune bascule possible
    pop    bp                                     ;Relire BP sur la pile
    ret

_GateA20  endp
```

```
;-----------------------------------------------------------------------
;-- IsA20On : Indique si le canal d'adresses A20 est libéré
;-- Appel par C: BOOL IsA20( void )
;-- Valeur de retour : TRUE, si canal libre sinon FALSE

ramptr     dd 000000000h        ;Pointeur dans la mémoire conventionnelle
extptr     dd 0FFFF0010h                ;Pointeur dans la mémoire étendue

_IsA20On   proc near

    push   ds
    push   es

    lds    si,cs:ramptr     ;Si le canal est bloqué, les deux pointeurs
    les    di,cs:extptr                 ;désignent une adresse identique

    mov    cx,64                                        ;Comparer 128 octets
    cld                  ;Si instruction de chaîne compter vers le haut
    repe   cmpsw                    ;compare les deux blocs de mémoire

    pop    es                                    ;Récupère les registres
    pop    ds
    jcxz   a20off                  ;CX = 0 --> Blocs mémoire identiques

    mov    ax,0FFFFh                  ;Bloc différent ---> A20 est actif
    ret

a20off:
    xor    ax,ax                   ;Blocs identique ---> A20 est bloqué
    ret

_IsA20On   endp

;-----------------------------------------------------------------------

_text      ends                                      ;fin du segment de code
           end                                          ;fin du programme
```

12.2.4. Le standard XMS

Depuis toujours, une interface logicielle spécifique est associée au standard EMS. Quant à la mémoire étendue, elle a longtemps supporté l'absence d'une telle interface si bien que les différents fabricants de logiciels ont fini par concocter leur propre mixture. Ainsi depuis 1988, il existe un homologue du standard EMS qui le XMS (Extended Memory Specification). Les sociétés impliquées dans cette invention étaient encore une fois Microsoft, INTEL ainsi que AST Research qui, en collaboration avec Quadram pour le standard EEMS, ont tenté de former une partie adverse contre le consortium LIM.

Le standard XMS définit une interface logicielle permettant simultanément à plusieurs programmes d'accéder à la mémoire étendue et à d'autres zones mémoire. Concrètement, voici les différents types de mémoire reconnus :

- ✓ la HMA qui intercepte les 64 premiers Ko de la mémoire étendue et s'étend de l'adresse 1024 Ko à 1088 Ko,

- ✓ les blocs de mémoire étendue (EMB) logés dans la mémoire étendue depuis l'adresse 1088 Ko et n'entrant pas ainsi en collision avec la HMA,

- ✓ les blocs de mémoire supérieurs (UMB) situés dans la zone RAM conventionnelle entre 640 Ko et 1024 Ko. Ils ont pris une signification importante avec DOS 5.0 et sont désormais disponibles pour les programmes DOS.

Le driver XMS le plus connu est sans aucun doute le driver HIMEM.SYS de Microsoft. Il est fourni avec les diverses versions DOS ainsi que Windows. Il est intégré au système à travers le fichier de configuration CONFIG.SYS. Ainsi, il reste obligatoirement disponible après la procédure de lancement.

Les drivers XMS se rencontrent également dans de nombreux programmes chargés de gérer la mémoire sur un i386 ou i486. Outre l'interface XMS, ils reconnaissent généralement le standard LIM. En guise d'exemple, on peut citer 386Max de Qualitas, QEMM386 de Quaterdeck et naturellement EMM386.EXE de DOS 5.0. En principe, ils sont également capables d'intégrer les cellules issues de la mémoire étendue dans la zone située entre 640 Ko et 1024 Ko sous la forme de blocs de mémoire supérieure (UMB = Upper Memory block). Dans ce cas, cette zone n'est pas occupée par la RAM vidéo, les extensions électroniques ou le BIOS en ROM. Le programmeur dispose ainsi d'une RAM supplémentaire. Elle peut être adressée comme une RAM conventionnelle à l'aide des techniques d'adressage normales.

Les UMB ne seraient disponibles que sur de rares périphériques si ces drivers et les 386 n'existaient pas. De même, ils ne seraient pas soutenus par tous les drivers XMS. Mais les ordinateurs équipés des fameux NEAT Chips de Chips & Technologies constituent la seule exception. Grâce à un confortable programme SETUP, la mémoire peut être configurée ici de telle manière qu'une quantité précise peut être rendue disponible sous la forme d'UMB.

Tout comme pour la mémoire paginée, il faut d'abord prouver l'existence d'un driver XMS avant d'utiliser les fonctions XMS. Cette action s'effectue à l'aide de l'interruption 2Fh. Elle offre un abri à tous les programmes résidents et elle est utilisée par tous les programmes DOS résidents tels que SHARE, APPEND ou PRINT. Lors de son installation, le driver XMS vient se nicher dans cette interruption et peut ainsi être réclamé.

Appelez d'abord l'interruption 2Fh avec le code de fonction 4300h dans le registre AX. Si un driver XMS est implanté dans le système, cette fonction retourne la valeur 80h dans le registre AL. Une autre valeur signifie qu'il n'existe aucun driver XMS et les fonctions XMS ne peuvent donc pas être utilisées.

Cela n'est absolument pas possible sans l'appel de l'interruption 2Fh. En effet, au lieu par exemple de l'interface logicielle du standard LIM, les fonctions du XMS ne seront pas appelées par une interruption mais une instruction FAR Call. Pour cela, il nous faut l'adresse du handler XMS à appeler qui est désigné également comme un gestionnaire de mémoire étendue (XMM). L'intérêt de l'interruption 2Fh ne se reconnaît que lorsqu'un driver XMS est effectivement installé. Le premier appel de cette interruption a donc retourné la valeur 80h dans le registre AL.

Dans ce cas, appelez l'interruption 2Fh une seconde fois avec la valeur 4310h dans le registre AX. En guise de résultat, vous obtenez l'adresse de segment dans le registre ES et l'adresse d'offset dans le registre BX du XMM. Ces adresses permettent d'appeler toutes les routines du standard XMS.

Fonction	Utilisation
00h	Déterminer le numéro de version XMS
01h	Allouer la HMA (High Memory Area)
02h	Libérer la HMA
03h	Activation globale de la ligne d'adresse A20
04h	Désactivation globale de la ligne d'adresse A20
05h	Activation locale de la ligne d'adresse A20
06h	Désactivation locale de la ligne d'adresse A20
07h	Déterminer l'état de la ligne d'adresse A20
08h	Déterminer la taille de la mémoire étendue libre
09h	Allouer un bloc de mémoire étendu (EMB)
0Ah	Libérer un bloc de mémoire étendu alloué (EMB)
0Bh	Déplacer un bloc de mémoire étendu alloué (EMB)
0Ch	Verrouiller un bloc de mémoire étendu (EMB)
0Dh	Déverrouiller un bloc de mémoire étendu (EMB)
0Eh	Lire les informations sur un handle EMB
0Fh	Agrandir ou réduire le bloc de mémoire étendu alloué (EMB)
10h	Allouer un bloc de mémoire supérieure (UMB)
11h	Libérer un bloc de mémoire supérieure (UMB) alloué

Comme le montre le tableau précédent, il existe en tout 18 fonctions XMS dont les trois dernières ne sont pas reconnues par tous les drivers XMS.

Lors de l'appel des différentes fonctions XMS, le numéro de fonction est en principe transmis dans le registre AH. D'autres informations peuvent également être retournées dans les registres du processeur, mais cela varie évidemment d'une fonction à l'autre.

Pratiquement toutes les fonctions retournent un code d'état dans le registre AX. Il donne des informations sur le déroulement de l'opération concernée. La valeur 0001h indique par exemple "OK", la valeur 0000h signale une erreur. Dans ce cas, le registre BL contient un code d'erreur qui donne des indications précises sur la cause de l'erreur. Vous trouverez en annexe la liste de tous les codes d'erreur ainsi qu'une description détaillée des fonctions XMS.

Mais avant d'adresser les différentes fonctions, vous devez d'abord déterminer le numéro de version du driver à l'aide de la fonction 00h. Dans le registre AX, elle retourne le numéro de version XMS et dans le registre BX, le numéro de révision interne variant d'un driver à l'autre. Son importance est toutefois moindre.

Les deux informations sont retournées en tant que nombres BCD où l'octet de poids fort contient le numéro de version principal (avant le point) et l'octet de poids faible le numéro de version secondaire (derrière le point). Par exemple, la valeur 0200h correspond au numéro de version 2.0. Evitez d'utiliser un numéro de version inférieur avec les fonctions XMS car les drivers de la version 1.0 ne disposent généralement que des fonctions d'accès à la HMA. Si vous rencontrez un tel driver, n'hésitez pas à inviter l'utilisateur à se procurer un driver XMS en cours.

A l'heure actuelle, les drivers de la version 2.0 occupent la première place. Mais n'excluons pas les drivers implantés par les programmes de gestion de mémoire sur les i386. Ils retournent généralement le numéro de version 3.0. Nous ne connaissons toutefois pas la différence entre ces deux versions.

Si vous souhaitez utiliser la mémoire étendue dans votre programme sous la forme d'EMB, vous devez d'abord déterminer la taille disponible dans la mémoire étendue avec la fonction 08h. Après son appel, elle retourne la taille totale de la mémoire étendue libre dans le registre DX et la quantité d'EMB libres dans le registre AX. Les valeurs sont indiquées en Ko. Elles doivent être maniées avec précaution parce que les 64 Ko sont compris dans le calcul des deux valeurs, l'allocation des EMB commençant toujours après la HMA.

Si votre système dispose par exemple de 4 Mo de mémoire dont 3 Mo sont réservés pour la mémoire étendue, la fonction 08h retourne le nombre 3072 pour 3 Mo dans les registres AX et DX. Si vous appelez ensuite la fonction 09h, le XMM serait ainsi prêt à allouer 3 Mo de mémoire étendue à partir de la fin de la HMA uniquement. Les 64 derniers Ko de cette zone vont au-delà de la fin de la mémoire étendue et ne sont donc pas du tout disponibles physiquement. Par conséquent, vous devez à chaque fois ôter 64 Ko des valeurs retournées par la fonction 08h, ce qui permet d'inclure la taille de la HMA dans le calcul.

Après avoir obtenu la taille du plus grand EMB disponible selon cette méthode, vous pouvez réclamer une quantité correspondante de mémoire étendue avec la fonction 09h. Outre le numéro de fonction, spécifiez l'ampleur du bloc souhaité en Ko dans le registre DX. Après l'appel de la fonction, ce registre contient également le handle nécessaire à un autre accès à l'EMB à condition que tout soit déroulé correctement lors de l'allocation.

Mais dans tous les cas, vous devez contrôler la situation à l'aide du code d'état situé dans le registre AX. Même si une mémoire étendue suffisante n'est pas encore transmise à d'autres programmes, il arrive que l'allocation d'un bloc de mémoire échoue. La responsabilité peut par exemple incomber aux handles définis par le XMM pour gérer les EMB et qui sont retournés dans le registre DX. Vu que le nombre de handles octroyés

au XMM est notamment limité, il arrive parfois que la mémoire étendue soit encore libre. Mais à cet effet, tous les handles sont déjà occupés parce que plusieurs petits EMB ont été alloués. Pour éviter cela, vous devez toujours réclamer des EMB aussi grands que possible et les sectionner ensuite en plusieurs zones mémoire utilisées différemment.

Une question reste encore sans réponse. Pourquoi le XMM utilise-t-il les handles et ne retourne pas l'adresse de la zone allouée ? La réponse se trouve dans la gestion de la mémoire du XMM car elle est entièrement conçue pour empêcher la fragmentation de la mémoire étendue. Par conséquent, le XMM tend à déplacer les divers EMB çà et là dans la mémoire étendue. Le but est de construire une zone plus grande à partir de plusieurs petits EMB libérés, cette zone pouvant être retournée comme résultat de l'appel de la fonction 09h. Cette procédure déplace également les adresses des EMB déjà allouées. Il est donc judicieux de fournir un handle logique au lieu de l'adresse physique concrète d'un EMB à tous ceux qui ont émis un appel.

Pensez à conserver correctement ce handle car il sera nécessaire pour tout autre accès à un EMB. Si l'EMB doit être libéré, agrandi ou du moins copié partiellement dans la mémoire conventionnelle, le handle cité entre en action car il permet d'identifier un EMB par rapport au XMM.

Libérer des EMB

Le plus simple est de libérer un EMB pour le rendre accessible à d'autres programmes. Cette action ne se réalise pas automatiquement avec la fin d'un programme parce que les programmes TSR ont également le droit d'utiliser les EMB. N'oubliez donc jamais de libérer les EMB alloués avec la fonction 0Ah avant de terminer un programme, sinon ils restent en possession de votre programme jusqu'à un reset de l'ordinateur. En dehors du numéro de fonction, il suffit de transmettre le handle EMB dans le registre DX.

Copier des EMB

Sachant que dans la mémoire étendue, les données peuvent uniquement être stockées et non manipulées, il convient de copier régulièrement les EMB en totalité ou en partie dans la RAM conventionnelle ou de là dans la mémoire étendue. Avec la fonction 0Bh, le standard XMS fournit une fonction conçue spécialement à cet effet. Dans la paire de registres DS:SI, il faut lui transmettre le pointeur sur une structure appelée "Extended Memory Move" contenant toutes les informations sur les zones source et cible ainsi que le nombre d'octets à copier.

Configuration de la structure Extended Memory Move		
Adresse	**Contenu**	**Type**
+00h	Longueur du bloc en octets	1 DWORD
+04h	Handle du bloc source	1 WORD
+06h	Offset dans le bloc source à partir duquel la copie doit commencer	1 DWORD
+0Ah	Handle du bloc source	1 WORD
+0Ch	Offset dans le bloc source à partir duquel la copie doit commencer	1 DWORD
Longueur : 18 octets		

Les informations concernant le handle et l'adresse d'offset des deux zones doivent être traitées différemment selon que la zone provient de la mémoire étendue ou de la RAM conventionnelle. Le handle doit être spécifié si un EMB est réclamé. Le handle est retourné par la fonction 09h lors de l'allocation de l'EMB. L'adresse d'offset représente l'offset par rapport au début du bloc. Au contraire, si une zone de la RAM conventionnelle est adressée, il faut alors spécifier la valeur 0 en guise de handle et les adresses de segment et d'offset du début du bloc au format habituel (d'abord l'adresse d'offset puis l'adresse de segment) en tant qu'offset.

En ce qui concerne la vitesse de cette procédure de copie, les mêmes restrictions que celles évoquées dans le cadre des fonctions BIOS 87h s'appliquent ici, du moins aux ordinateurs 80286.

Outre les quatre fonctions déjà mentionnées, le standard XMS dispose de quatre autres fonctions portant sur les EMB. Elles portent les numéros 0Ch à 0Fh mais sont utilisées dans des cas exceptionnels. Reportez-vous à l'annexe pour obtenir de plus amples informations sur ces fonctions.

Accès à la HMA

XMM permet non seulement d'accéder à la mémoire étendue mais aussi à la HMA. Cela évite des conflits avec d'autres programmes et crée une indépendance électronique grâce aux fonctions permettant de contrôler la ligne d'adresse A20. Cela est important car tous les systèmes PC ne gèrent pas cette ligne à travers un bit dans le port de sortie du contrôleur de clavier comme il est décrit dans les paragraphes suivants.

Contrairement à la mémoire étendue, l'utilisation de la HMA ne peut pas être répartie entre plusieurs programmes en raison de sa taille. Ainsi, un programme ayant obtenu l'accès à la HMA à l'aide de la fonction 01h reçoit la totalité de la HMA ou rien du tout. Le dernier cas se présente lorsque la HMA est déjà allouée ou lorsque la quantité de mémoire HMA nécessaire est inférieure au paramètre /HMAMIN. Il s'agit d'un paramètre optionnel spécifié lors de l'appel du driver. Il empêche d'accorder un droit

d'accès exclusif à la HMA à un programme TSR nécessitant peu de Ko alors qu'un autre programme ayant besoin de plus de mémoire serait lésé.

Cela ne concerne pas les applications normales libérant la HMA avant de prendre congé. Elles doivent toujours spécifier avec FFFFh la quantité de mémoire HMA dont elles ont besoin lors de l'appel de la fonction 02h. Au contraire, les programmes TSR doivent déclarer explicitement en octets la quantité nécessaire dans le registre DX.

Le contenu du registre AX obtenu après l'appel de la fonction indique si le droit d'accès a été accordé ou non. Ici, la valeur 0001h indique que la HMA a été libérée alors que la valeur 0000h signale l'échec de la demande.

Bien que son allocation soit effectuée correctement, il n'est pas encore possible d'accéder à la HMA car il faut auparavant libérer la ligne d'adresse A20. Dans ce contexte, le XMM fournit en tout quatre fonctions servant à libérer et verrouiller cette ligne. Pour leur appel, ces fonctions se contentent de leur numéro qui doit être placé dans le registre AX avant l'appel.

Il existe quatre fonctions pour remplir cette tâche relativement simple parce que les fonctions chargées de libérer et verrouiller la ligne A20 se présentent respectivement deux fois : une fois sous forme locale et une autre fois sous forme globale. Cet attribut décrit l'effet produit par les fonctions. Les fonctions globales agissent toujours sur la ligne d'adresse A20, alors que cela dépend d'un compteur interne dans le cas des fonctions locales. Les deux fonctions influent sur le compteur. Ce dernier veille à ce qu'elles ne viennent agir sur la ligne d'adresse A20 que lorsque leurs appels soient équilibrés. A la suite de deux appels consécutifs de la fonction d'activation locale, la ligne est donc d'abord désactivée lorsque deux appels consécutifs de la fonction de désactivation surviennent immédiatement après.

Vous avez rencontré une procédure similaire dans le contexte de la souris et l'affichage ou le masquage du pointeur de souris. Cette technique est intéressante dans le cadre des appels répétitifs. Partout où chaque procédure active ou désactive la ligne d'adresse avant de lancer l'action appropriée, il devient possible d'annuler à tout moment cette action. Mais comme cela n'arrive que très rarement, vous pouvez vous limiter aux deux fonctions globales lorsque vous utilisez la HMA.

N'oubliez pas de désactiver la ligne d'adresse A20 avec la fonction 04h avant de terminer votre programme et libérer la HMA avec la fonction 02h. Sinon, l'accès de la HMA reste interdit aux programmes appelés ultérieurement ou ils se plantent parce qu'ils espéraient un débordement de segment au-delà de la limite des 1 Mo.

Bloc de mémoire supérieure (UMB)

Tout comme la HMA, les blocs de mémoire supérieure présentent un avantage important. Ils se situent dans la zone d'adresse du processeur ce qui permet à un

programme d'y accéder directement. En contrepartie, la mémoire RAM entre 640 Ko et 1 Mo est très rare si l'on ne dispose pas en particulier d'un i386. Dans ce cas, des programmes appropriés aident à rajouter une mémoire RAM dans la partie supérieure.

Si vous êtes sûr que le système cible pour lequel vous développez le programme dispose d'UMB ou du moins comporte une option concernant cette mémoire, vous soutenez alors le standard XMS avec deux fonctions d'accès à la mémoire supérieure. Notez toutefois que depuis la version 5.0, DOS est également capable d'utiliser les blocs de mémoire supérieure même s'il ne sert seulement que comme intermédiaire entre un programme et le driver XMS.

Contrairement à la HMA, il existe ici une véritable gestion de la mémoire permettant à plusieurs programmes de participer à la mémoire supérieure. Au lieu d'utiliser les handles, le travail s'effectue avec l'adresse de segment du bloc alloué. Elle est utilisée comme une clé pour identifier un UMB par rapport à un XMM.

Allouer des UMB

Pour obtenir l'adresse de segment, appelez la fonction 10h qui alloue un UMB et adaptez le XMM à vos besoins. Si cela n'est pas possible parce que la mémoire disponible est insuffisante, consultez le registre DX qui indique l'ampleur du plus grand UMB libre mesuré en paragraphes. Généralement, le registre DX est celui où vous devez stocker la taille du bloc souhaité avant l'appel de la fonction. L'unité utilisée ici est le paragraphe (16 octets).

Libérer des UMB

Si le XMM permet d'allouer des UMB, il autorise également à les libérer pour qu'ils ne soient pas liés à un autre programme. La fonction 11h remplit cette tâche. Mis à part le numéro de fonction, elle attend l'adresse de segment de l'UMB réclamé dans le registre DX.

Programmes de démonstration

Nous avons conçu deux programmes pour décrire le travail lié au standard XMS. Ils peuvent servir à intégrer la mémoire XMS dans vos propres programmes parce qu'ils contiennent d'ores et déjà toutes les routines nécessaires à l'appel des fonctions du standard XMS. Les deux programmes sont écrits en Pascal et C. Ils n'existent pas en Assembleur parce que l'appel du XMM pose quelque problème dans un langage évolué. La bibliothèque contient des routines modernes de compilation en C et Pascal. Elles sont prévues pour appeler n'importe quelle fonction d'interruption, mais l'appel et la transmission du contenu des registres à des sous-programmes tout à fait normaux ne

sont pas prévus. Une routine en Assembleur a donc été développée pour l'appel du XMM. Elle peut être importée dans le programme C (XMSC.C) à l'aide d'un module en Assembleur (XMSCA.ASM). Dans le programme Pascal (XMSP.PAS), elle est codifiée avec l'instruction INLINE.

Dans les deux cas, cette routine fonctionne comme les commandes d'appel d'interruptions. Une structure contenant les registres du processeur pour l'appel de la fonction XMM lui est transmise. Cette structure contient également les valeurs retournées après l'appel de la fonction, ce qui permet d'obtenir l'état d'erreur et d'autres informations.

Dans les deux programmes, cette routine n'est pas appelée directement, mais à l'aide des diverses routines servant d'interface aux 18 fonctions du XMM. Cela vous soulage énormément comme le montre d'ailleurs le programme principal dans les deux cas. Il vérifie notamment l'existence d'un driver XMM et en cas de succès appelle deux procédures contrôlant la taille de la HMA et de la mémoire étendue.

Listing : XMSP.PAS

```
{*********************************************************************
*                          X M S P . P A S                           *
**-------------------------------------------------------------------**
* Sujet       : Démonstration de l'accès à la mémoire étendue et de  *
*               la zone High Memory à l'aide des fonctions XMS telles*
*               qu'elles sont appliquées par exemple avec le pilote  *
*               de périphériques HIMEM.SYS.                          *
**-------------------------------------------------------------------**
* Auteur      : MICHAEL TISCHER                                      *
* Développé le : 27.07.1990                                          *
* Dernière MAJ : 29.03.1992                                          *
*********************************************************************}

program XMSP;

uses Crt, Dos;                  { Pour appel interruption et accès clavier }

const ERR_NOERR          = $00;                    { Pas d'erreur }
      ERR_NOTIMPLEMENTED = $80;          { Fonction appelée inconnue }
      ERR_VDISKFOUND     = $81;             { Ram disk VDISK détecté }
      ERR_A20            = $82;     { Erreur sur canal d'adresses A20 }
      ERR_GENERAL        = $8E;   { Erreur pilote à caractère général }
      ERR_UNRECOVERABLE  = $8F;              { Erreur irratrapable }
      ERR_HMANOTEXIST    = $90;                 { HMA introuvable }
      ERR_HMAINUSE       = $91;               { HMA déjà appelée }
      ERR_HMAMINSIZE     = $92;           { Taille HMA insuffisante }
      ERR_HMANOTALLOCED  = $93;               { HMA non affectée }
      ERR_A20STILLON     = $94;   { Canal d'adresses A20 encore actif }
      ERR_OUTOMEMORY     = $A0;  { Plus de mémoire étendue disponible }
      ERR_OUTOHANDLES    = $A1;       { Ts les identifs XMS sont pris }
      ERR_INVALIDHANDLE  = $A2;          { Identificateur invalide }
      ERR_SHINVALID      = $A3;    { Identificateur source invalide }
      ERR_SOINVALID      = $A4;          { Décalage source invalide }
      ERR_DHINVALID      = $A5; { Identificateur destination invalide }
      ERR_DOINVALID      = $A6;     { Décalage destination invalide }
      ERR_LENINVALID     = $A7; { Longueur invalide pour fonction Move }
      ERR_OVERLAP        = $A8;            { Recouvrement interdit }
      ERR_PARITY         = $A9;               { Erreur de parité }
      ERR_EMBUNLOCKED    = $AA;           { UMB n'est pas bloqué }
      ERR_EMBLOCKED      = $AB;          { UMB est encore bloqué }
      ERR_LOCKOVERFLOW   = $AC;  { Débordement du compteur de blocage }
      ERR_LOCKFAIL       = $AD;     { L'UMB ne peut pas être bloqué }
      ERR_UMBSIZETOOBIG  = $B0;       { Plus petit UMB disponible }
      ERR_NOUMBS         = $B1;          { Plus d'UMB disponible }
      ERR_INVALIDUMB     = $B2;{ L'adresse du segment UMB est invalide }

type XMSRegs = record                   { Informations pour appel XMS }
                 AX,                  { Seuls les registres AX, BX, DX et }
                 BX,                   { SI sont requis selon la fonction }
                 DX,                    { appelée, il faut donc une autre }
                 SI,                              { adresse de segment }
                 Segment : word
               end;

{-- Variables globales ------------------------------------------------}

var XMSPtr : pointer;     { Pointeur sur Extended Memory Manager (XMM) }
    XMSErr : BYTE;          { Code d'erreur de la dernière opération }

{*********************************************************************
* XMSInit : Initialise les routines d'appel des fonctions XMS        *
**-------------------------------------------------------------------**
* Entrée : aucune                                                    *
* Sortie : TRUE si pilote XMS identifié sinon FALSE                  *
* Info   : - L'appel de cette fonction doit précéder celui de toutes *
*            les autres procédures et fonctions émanant de ce        *
*            programme.                                              *
*********************************************************************}

function XMSInit : boolean;

var Regs : Registers;             { Registre pour appel d'interruption }
    xr : XMSRegs;

begin
  Regs.AX := $4300;             { Détecte disponibilité du XMS Manager }
  intr( $2F, Regs );
  if ( Regs.AL = $80 ) then                    { Détecté XMS-Manager? }
    begin                                                  { Oui }
      Regs.AX := $4310;             { Retourner point d'accès au XMM }
      intr( $2F, Regs );
      XMSPtr := ptr( Regs.ES, Regs.BX );  { Copie adresse ds var glob. }
      XMSErr := ERR_NOERR;                          { Pas d'erreur apparue }
      XMSInit := true;            { Trouvé handler, module initialisé }
    end
  else                                  { XMS handler n'est pas installé }
    XMSInit := false;
end;

{*********************************************************************
* XMSCall : Routine générale d'appel d'une fonction XMS              *
**-------------------------------------------------------------------**
* Entrée : NumFonc = Numéro de la fonction XMS appelante             *
*          XRegs = Structure de registres pour appel de fonction     *
* Info   : - Avant l'appel de cette procédure, charger exclusivement *
*            les registres effectivement nécessaires à l'appel de la *
*            fonction.                                               *
*          - Après l'appel de la fonction XMS, les contenus des      *
*            registres du processeur sont copiés dans les composants *
*            correspondants de la structure renvoyée.                *
*          - Le premier appel de cette procédure doit être précédé   *
*            d'un appel en bon ordre de la procédure XMSInit.        *
*********************************************************************}

procedure XMSCall( NumFonc : byte; var XRegs : XMSRegs );

begin
  inline ( $8C / $D9 /                          { mov   cx,ds       *)
```

```
!          $51 /                                { push   cx          *)
!          $C5 / $BE / $04 / $00 /              { lds    di,[bp+0004] *)
!          $8A / $66 / $08 /                    { mov    ah,[bp+0008] *)
!          $8B / $9D / $02 / $00 /              { mov    bx,[di+0002] *)
!          $8B / $95 / $04 / $00 /              { mov    dx,[di+0004] *)
!          $8B / $B5 / $06 / $00 /              { mov    si,[di+0006] *)
!          $8E / $5D / $08 /                    { mov    ds,[di+08]   *)
!          $8E / $C1 /                          { mov    es,cx        *)
!          $26 / $FF / $1E / XMSPtr /           { call   es:[XMSPTr]  *)
!          $8C / $D9 /                          { mov    cx,ds        *)
!          $C5 / $7E / $04 /                    { lds    di,[bp+04]   *)
!          $89 / $05 /                          { mov    [di],ax      *)
!          $89 / $50 / $02 /                    { mov    [di+02],bx   *)
!          $89 / $55 / $04 /                    { mov    [di+04],dx   *)
!          $89 / $75 / $06 /                    { mov    [di+06],si   *)
!          $89 / $4D / $08 /                    { mov    [di+08],cx   *)
!          $1F                                  { pop    ds           *)
!        );
!
!      {-- Tests codes erreur ------------------------------------------}
!
!   if ( XRegs.AX = 0 ) and ( XRegs.BX >= 128 ) then
!     begin
!       XMSErr := Lo(XRegs.BX)              { Erreur, stocker code erreur }
!       {
!         .
!         .
!         .
!         emplacement éventuel pour un autre traitement d'erreur
!         .
!         .
!         .
!       }
!     end
!   else
!     XMSErr := ERR_NOERR;                  { Pas d'erreur, ça baigne }
!end;
!
!{*********************************************************************
!* XMSQueryVer: Renvoie le n° de version de l'XMS et autres informations*
!*              sur l'état                                             *
!**-----------------------------------------------------------------**
!* Entrée : VerNr = Stocke n° de version après appel de la fonction   *
!*                  (Format: 235 = 2.35)                              *
!*          RevNr = Stocke n° de révision après appel de la fonction  *
!* Sortie : TRUEsi un HMA est disponible sinon FALSE                  *
!*********************************************************************}
!
!function XMSQueryVer( var VerNr, RevNr : integer ): boolean;
!
!var XR : XMSRegs;                         { Registre de communication avec XMS }
!
!begin
!  XmsCall( 0, XR );
!  VerNr := Hi(XR.AX)*100 + ( Lo(XR.AX) shr 4 ) * 10 +
!           ( Lo(XR.AX) and 15 );
!  RevNr := Hi(XR.BX)*100 + ( Lo(XR.BX) shr 4 ) * 10 +
!           ( Lo(XR.BX) and 15 );
!  XMSQueryVer := ( XR.DX = 1 );
!end;
!
!{*********************************************************************
!* XMSGetHMA : Retourner à l'appelant le droit d'accès à la HMA.      *
!**-----------------------------------------------------------------**
!* Entrée : LenB = Nombre d'octets à allouer                          *
!* Info   : Les programmes résidents devraient réserver exclusivement *
!*          la mémoire strictement requise. Par contre, donner $FFFF  *
!*          aux applications.                                         *
!* Sortie : TRUE si la HMA a pu être rendue disponible sinon FALSE;   *
!*********************************************************************}
!
!function XMSGetHMA( LenB : word ) : boolean;
!
!var Xr : XMSRegs;                         { Registre de communication avec XMS }
!
!begin
!  XR.DX := LenB;                          { Stocke longueur dans registre DX }
!  XmsCall( 1, Xr );                                  { Appel fonction XMS #1 }
!  XMSGetHMA := ( XMSErr = ERR_NOERR );
!end;
!
!{*********************************************************************
!* XMSReleaseHMA : Libère l'HMA et permet ainsi sa transmission à     *
!*                 d'autres programmes.                               *
!**-----------------------------------------------------------------**
!* Entrée : aucune                                                    *
!* Info   : - Appeler cette procédure avant de quitter un programme   *
!*            si la HMA a été allouée par un appel de XMSGetHMA pour  *
!*            pouvoir la transmettre aux programmes appelés plus tard *
!*          - L'appel de cette procédure entraîne la perte des        *
!*            données stockées dans la HAM.                           *
!*********************************************************************}
!
!procedure XMSReleaseHMA;
!
!var Xr : XMSRegs;                         { Registre de communication avec XMS }
!
!begin
!  XmsCall( 2, Xr );                                   { Appel fonction XMS #2 }
!end;
!
!{*********************************************************************
!* XMSA20OnGlobal: Libère le canal d'adresses A20, permettant l'accès *
!*                 direct à la HMA..                                  *
!**-----------------------------------------------------------------**
!* Entrée : aucune                                                    *
!* Info   : - La libération du canal d'adresses A20 est relativement  *
!*            lente sur de nombreux ordinateurs. Veillez à ne pas     *
!*            abuser de l'usage de cette procédure.                   *
!*********************************************************************}
!
!procedure XMSA20OnGlobal;
!
!var Xr : XMSRegs;                         { Registre de communication avec XMS }
!
!begin
!  XmsCall( 3, Xr );                                   { Appel fonction XMS #3 }
!end;
!
!{*********************************************************************
!* XMSA20OffGlobal: Pendant de la procédure XMSA20OnGlobal, celle-ci  *
!*                  bloque à nouveau le canal d'adresses A20, inter-  *
!*                  sant l'accès direct à la HMA.                     *
!**-----------------------------------------------------------------**
!* Entrée : aucune                                                    *
!* Info   : - Appelez toujours cette procédure avant de quitter un    *
!*            programme si le canal d'adresses A20 a été libéré par   *
!*            un appel de XMSA20OnGlobal.                             *
!*********************************************************************}
!
!procedure XMSA20OffGlobal;
!
!var Xr : XMSRegs;                         { Registre de communication avec XMS }
!
!begin
!  XmsCall( 4, Xr );                                   { Appel fonction XMS #4 }
!end;
!
!{*********************************************************************
!* XMSA20OnLocal: Voir XMSA20OnGlobal                                 *
!**-----------------------------------------------------------------**
!* Entrée : aucune                                                    *
!* Info   : - Cette procédure locale se distingue de sa variante      *
!*            globale par la libération du canal: elle n'est possible *
!*            que si elle n'a pas été effectuée par un appel précédent *
!*********************************************************************}
!
!procedure XMSA20OnLocal;
!
!var Xr : XMSRegs;                         { Registre de communication avec XMS }
!
!begin
!  XmsCall( 5, Xr );                                   { Appel fonction XMS #5 }
!end;
!
!{*********************************************************************
!* XMSA20OffLocal : Voir XMSA29OffGlobal                              *
!**-----------------------------------------------------------------**
!* Entrée : aucune                                                    *
!* Info   : - Cette procédure locale se distingue de sa variante      *
!*            globale par la libération du canal: elle n'est possible *
!*            que si elle n'a pas été effectuée par un appel précédent *
!*********************************************************************}
!
!procedure XMSA20OffLocal;
!
!var Xr : XMSRegs;                         { Registre de communication avec XMS }
!
!begin
!  XmsCall( 6, Xr );                                   { Appel fonction XMS #6 }
!end;
!
!{*********************************************************************
!* XMSIsA20On : Retourne l'état du canal d'adresse A20                *
!**-----------------------------------------------------------------**
!* Entrée : aucune                                                    *
!* Sortie : TRUE si le canal d'adresses A20 est libéré sinon  FALSE.  *
!*********************************************************************}
!
!function XMSIsA20On : boolean;
!
!var Xr : XMSRegs;                         { Registre de communication avec XMS }
!
!begin
!  XmsCall( 7, Xr );                                   { Appel fonction XMS #7 }
!  XMSIsA20On := ( Xr.AX = 1 );                  { AX = 1 ---> canal libre }
```

```
end;

{*****************************************************************
* XMSQueryFree : Renvoie la mémoire étendue disponible et la taille du *
*                plus gros bloc libre                                  *
*---------------------------------------------------------------------*
* Entrée : TotalLibre: Stocke la taille totale de l'EM libre.         *
*          MaxBl  : Stocke la taille du plus grand bloc libre.        *
* Info   : - Les deux valeurs sont en Ko                              *
*          - La taille de la HMA n'est pas comptabilisée même si elle *
*            n'a été affectée à aucun programme.                      *
*****************************************************************}

procedure XMSQueryFree( var TotalLibre, MaxBl : integer );

var Xr : XMSRegs;                    { Registre de communication avec XMS }

begin
  XmsCall( 8, Xr );                                  { Appel fonction XMS @8 }
  TotalLibre := Xr.AX;                          { La taille totale est dans AX }
  MaxBl   := Xr.DX;                             { La mémoire libre est dans DX }
end;

{*****************************************************************
* XMSGetMem : Alloue un bloc de mémoire étendue (EMB)             *
*---------------------------------------------------------------------*
* Entrée : LenKB : Taille en Ko du bloc interrogé                     *
* Sortie : Identificateur pour accès au bloc ou 0, si aucun bloc n'a  *
*          pu être alloué. Un code d'erreur se trouve alors dans la   *
*          variable globale XMSErr.                                   *
*****************************************************************}

function XMSGetMem( LenKb : integer ) : integer;

var Xr : XMSRegs;                    { Registre pour communiquer avec XMS }

begin
  Xr.DX := LenKB;                { Stockons la longueur dans le registre DX }
  XmsCall( 9, Xr );                                  { Appel fonction XMS @9 }
  XMSGetMem := Xr.DX                                 { Retourne l'identificateur }
end;

{*****************************************************************
* XMSFreeMem : Libère un bloc de mémoire étendue (EMB) précédemment *
*              alloué                                             *
*---------------------------------------------------------------------*
* Entrée : Handle : L'identificateur pour accéder au bloc. Il a été  *
*                   obtenu en appelant XMSGetMem.                    *
* Info   : - Le contenu de l'EMB est définitivement détruit par cet   *
*            appel, l'identificateur devient invalide.              *
*          - Avant de quitter un programme, libérez à l'aide de cette *
*            procédure toutes les zones précédemment allouées pour    *
*            pouvoir les allouer aux programmes suivants.           *
*****************************************************************}

procedure XMSFreeMem( Handle : integer );

var Xr : XMSRegs;                    { Registre pour communiquer avec XMS }

begin
  Xr.DX := Handle;          { Copions l'identificateur dans le registre DX }
  XmsCall( 10, Xr );                                 { Appel fonction XMS @10 }
end;

{*****************************************************************
* XMSCopy : : Copie des zones de mémoire entre la mémoire étendue et la*
*           mémoire conventionnelle ou à l'intérieur de ces deux      *
*           groupes de mémoire.                                       *
*---------------------------------------------------------------------*
* Entrée : HandleOrig  : Identificateur du bloc de mémoire à déplacer *
*          OffsetOrig  : Offset dans ce bloc, à partir duquel le      *
*                        déplacement sera effectué.                   *
*          HandleDest  : Identificateur du bloc de mémoire cible      *
*          OffsetDest  : Offset dans le bloc cible.                   *
*          LenW        : Nombre de mots déplacés                      *
* Info   : - Pour utiliser la mémoire normale dans cette opération,   *
*            donnez la valeur 0 à l'identificateur ("Handle") et pour *
*            l'offset, le segment et l'adresse de l'offset dans sa    *
*            forme habituelle (offset avant le segment).              *
*****************************************************************}

procedure XMSCopy( HandleOrig  : integer; OffsetOrig  : longint;
                   HandleDest : integer; OffsetDest : longint;
                   LenW       : longint  );

type EMS = record                          { Structure Move mémoire étendue }
             LenB    : longint;                  { Nbre d'octets à décaler }
             SHandle : integer;                   { Identificateur source }
             SOffset : longint;                        { Décalage source }
             DHandle : integer;                  { Handle destination }
             DOffset : longint;                  { Décalage destination }
           end;

var Xr : XMSRegs;                    { Registre de communication avec XMS }
    MI : EMS;                                        { Stocke l'EMS }

begin
  with MI do                                 { Commencer par préparer l'EMS }
    begin
      LenB := 2 * LenW;
      SHandle := HandleOrig;
      SOffset := OffsetOrig;
      DHandle := HandleDest;
      DOffset := OffsetDest
    end;

  Xr.SI      := Ofs( MI );                  { Adresse de l'offset de l'EMS }
  Xr.Segment := Seg(MI);                    { Adresse du segment de l'EMS }
  XmsCall( 11, Xr );                              { Appel fonction XMS @11 }
end;

{*****************************************************************
* XMSLock : Interdit tout décalage d'un bloc de mémoire étendue par   *
*           l'XMM et retourne son adresse absolue.                    *
*---------------------------------------------------------------------*
* Entrée : Handle : Identificateur du bloc de mémoire retourné lors de *
*          l'appel précédent de XMSGetMem.                            *
* Sortie : Adresse linéaire du bloc de mémoire.                       *
*****************************************************************}

function XMSLock( Handle : integer ) : longint;

var Xr : XMSRegs;                    { Registre de communication avec XMS }

begin
  Xr.DX := Handle;                                   { Handle de l'EMB }
  XmsCall( 12, Xr );                            { Appel fonction XMS @12 }
  XMSLock := longint(Xr.DX) shl 16 + Xr.BX;  { Calcule adr.sur 32 bits }
end;

{*****************************************************************
* XMSUnlock : Libère à nouveau un bloc de mémoire étendu pour une     *
*             opération de décalage.                                  *
*---------------------------------------------------------------------*
* Entrée : Handle : Identificateur de la zone de mémoire retourné lors *
*          d'un appel précédent de XMSGetMem.                         *
*****************************************************************}

procedure XMSUnLock( Handle : integer );

var Xr : XMSRegs;                    { Registre de communication avec XMS }

begin
  Xr.DX := Handle;                                   { Handle de l'EMB }
  XmsCall( 13, Xr );                            { Appel fonction XMS @13 }
end;

{*****************************************************************
* XMSQueryInfo : Retourne diverses informations sur un bloc de        *
*                mémoire étendue préalablement alloué.                *
*---------------------------------------------------------------------*
* Entrée :  Handle : Identificateur de la zone de mémoire             *
*           Lock   : Variable de stockage du compteur de Lock         *
*           LenKB  : Variable de stockage de la longueur du bloc en Ko *
*           FreeH  : Variable de stockage du nombre d'identificateurs *
*                    restant libres.                                  *
* Info    : Cette procédure ne permet pas de connaître l'adresse d'un *
*           bloc. Utilisez la fonction XMSLock pour cette information. *
*****************************************************************}

procedure XMSQueryInfo( Handle    : integer; var Lock, LenKB : integer;
                        var FreeH : integer );

var Xr : XMSRegs;                    { Registre de communication avec XMS }

begin
  Xr.DX := Handle;                                   { Handle de l'EMB }
  XmsCall( 14, Xr );                            { Appel fonction XMS @14 }
  Lock  := Hi( Xr.BX );                              { Lit les registres }
  FreeH := Lo( Xr.BX );
  LenKB := Xr.DX;
end;

{*****************************************************************
* XMSRealloc : Agrandit ou réduit la taille d'un bloc de mémoire      *
*              étendue alloué par XMSGetMem                           *
*---------------------------------------------------------------------*
* Entrée :  Handle   : Identificateur de la zone de mémoire           *
*           NewLenKB : Nouvelle taille du bloc, en Ko                 *
* Sortie :  TRUE si la taille du bloc a été modifiée sinon FALSE      *
* Info   :  Ce bloc ne doit pas être verrouillé!                      *
*****************************************************************}

function XMSRealloc( Handle, NewLenKB : integer ) : boolean;

var Xr : XMSRegs;                    { Registre de communication avec XMS }
```

```
{
begin
  Xr.DX := Handle;                                    { Handle de l'EMB }
  Xr.BX := NewLenKB;          { La nouvelle longueur dans le registre BX }
  XmsCall( 15, Xr );                                  { Appel fonction XMS @15 }
  XMSRealloc := ( XMSErr = ERR_NOERR );
end;

{*****************************************************************
* XMSGetUMB : Alloue un bloc de Upper Memory (UMB)               *
*----------------------------------------------------------------**
* Entrée :  LenPara : Taille de la zone allouée en paragraphes de 16  *
*                     octets chacun.                                  *
*           Seg     : Variable de stockage de l'adresse du segment de *
*                     l'UMB alloué (si tout va bien)                  *
*           MaxPara : Variable contenant la taille du plus gros bloc  *
*                     UMB en cas d'échec.                             *
* Sortie : TRUE si un UMB a été alloué sinon FALSE                    *
* Info   : Attention! Cette fonction n'est pas compatible avec tous   *
*          les pilotes XMS. Elle est extrêmement dépendante du        *
*          matériel.                                                  *
*****************************************************************}

function XMSGetUMB( LenPara          : integer;
                    var Seg, MaxPara : word ) : boolean;

var Xr : XMSRegs;                          { Registre de communication avec XMS }

begin
  Xr.DX := LenPara;                              { Longueur demandée selon DX }
  XmsCall( 16, Xr );                                  { Appel fonction XMS @16 }
  Seg := Xr.BX;                                { Retourne l'adresse du segment }
  MaxPara := Xr.DX;                              { Longueur du plus gros UMB }
  XMSGetUMB := ( XMSErr = ERR_NOERR );
end;

{*****************************************************************
* XMSFreeUMB : Libère un UMB alloué par XMSGetUMB.                *
*----------------------------------------------------------------**
* Entrée  : Seg : Adresse du segment de l'UMB à libérer           *
* Info    : Attention! Cette fonction n'est pas compatible avec tous  *
*           les pilotes XMS. Elle est extrêmement dépendante du       *
*           matériel.                                                 *
*****************************************************************}

procedure XMSFreeUMB( var Seg : word );

var Xr : XMSRegs;                          { Registre de communication avec XMS }

begin
  Xr.DX := Seg;                        { Adresse du segment de l'UMB selon DX }
  XmsCall( 17, Xr );                                  { Appel fonction XMS @17 }
end;

{-----------------------------------------------------------------}
{-- Procédures de test et de démonstration                      --}
{-----------------------------------------------------------------}

{*****************************************************************
* HMATest : Teste la disponibilité de l'HMA et démontre son maniement. *
*----------------------------------------------------------------**
* Entrée : aucune                                                 *
*****************************************************************}

procedure HMATest;

type HMAR = array [1..65520] of BYTE;                { Le tableau HMA }
     HMARPTR = ^HMAR;                          { Pointeur sur le tableau HMA }

var ch   : char;                                  { pour interroger touches }
    A20  : boolean;                  { Etat courant du canal d'adresse A20 }
    hmap : HMARPTR;                                    { Pointeur sur l'HMA }
    i,                                            { Compteur d'itérations }
    err  : word;                          { Nombre d'erreurs d'accès à la HMA }

begin
  write( 'Test HMA : Veuillez taper une touche pour lancer ' +
         'le test...' );
  ch := ReadKey;
  writeln( #10 );

  {-- Allouer HMA et tester chaque adresse de la mémoire ------------}

  if ( XMSGetHMA( $FFFF) ) then                  { Contrôlons-nous l'HMA? }
    begin                                                          { Oui }
      A20 := XMSIsA20On;                           { Donner l'état du canal }
      If ( A20 = FALSE ) then                          { Canal A20 libre ? }
        XMSA20OnGlobal;                        { Non, le libérer maintenant }

      hmap := HMARPTR(Ptr( $FFFF, $0010 ));          { Pointeur sur HMA }

      err := 0;                                { Jusque là encore aucune erreur }
      for i := 1 to 65520 do          { Tester chaque adresse séparément }
        begin
          write( #13, 'Cellule mémoire : ', i );
          hmap^[i] := i mod 256;                   { Ecrire dans l'adresse }
          if ( hmap^[i] <> i mod 256 ) then                { et la relire }
            begin                                                  { Erreur! }
              writeln( ' ERREUR! ' );
              inc( err );
            end;
        end;

      XMSReleaseHMA;                                       { Libérons l'HMA }
      If ( A20 = FALSE ) then   { Est-ce que le canal A20 était libre? }
        XMSA20OffGlobal;                                  { Non, libérons-le }

      writeln( #13 );
      if ( err = 0 ) then                  { Analyse du résultat du test }
        writeln( 'HMA ok, aucune cellule de la mémoire défectueuse.')
      else
        writeln( 'ATTENTION! ', err, ' cellules défectueuses dans',
                 ' l''HMA ! ');
    end
  else
    writeln( 'ATTENTION! Accès impossible à l''HMA.' );
end;

{*****************************************************************
* EMBTest : Teste la mémoire étendue et montre l'appel de plusieurs   *
*           fonctions XMS                                             *
*----------------------------------------------------------------**
* Entrée : aucune                                                 *
*****************************************************************}

procedure EMBTest;

type BAR = array [1..1024] of BYTE;          { tableau "Byte array" d'1 Ko }
     BARPTR = ^BAR;                  { Pointeur sur le tableau "Byte array" }

var ch      : char;                                  { Pour interroger touches }
    Adr     : longint;                                    { Adresse de l'EMB }
    barp    : BARPTR;                          { Pointeur sur tampon d'1 Ko }
    i, j,                                          { Compteur d'itérations }
    err,                                  { Nombre d'erreurs en accès à l'HMA }
    Handle,                               { Identificateur pour l'accès à l'EMB }
    TotalLibre,                           { Taille de toute la mémoire étendue }
    MaxBl   : integer;                                { Plus gros bloc libre }

begin
  write( 'Test EMB : Veuillez taper une touche pour ' +
         'lancer le test...' );
  ch := ReadKey;
  writeln( #10 );

  XMSQueryFree( TotalLibre, MaxBl ); { Ret. taille tot. de mém. étend. }
  writeln( 'Taille totale de la mémoire étendue (avec HMA) : ',
           TotalLibre, ' KB' );
  writeln( '                    Dont le plus gros bloc libre : ',
           MaxBl, ' KB' );

  TotalLibre := TotalLibre - 64;   { Calcul taille effective sans HMA. }
  if ( MaxBl >= TotalLibre ) then                { Valeur vraisemblable? }
    MaxBl := MaxBl - 64;                                          { Non }

  if ( MaxBl <> 0 ) then                        { Encore assez de mémoire? }
    begin                                                          { Oui }
      Handle := XMSGetMem( MaxBl );
      writeln( MaxBl, ' Ko alloués.' );
      writeln( 'Identificateur = ', Handle );
      Adr := XMSLock( Handle );                          { Retourner l'adresse }
      XMSUnlock( Handle );                                { Supprimer blocage }
      writeln( 'Adresse de départ = ', Adr, ' (', Adr div 1024, ' Ko)');

      GetMem( barp, 1024 );                          { Tampon sur Turbo Heap }
      err := 0;                                { Jusque là encore aucune erreur }

      {-- Vérifier l'EMB Ko après Ko ------------------------------}

      for  i := 0 to MaxBl-1 do
        begin
          write( #13, 'Test Ko: ', i+1 );
          FillChar( barp^, 1024, i mod 255 );
          XMSCopy( 0, longint(barp), Handle, longint(i)*1024, 512 );
          FillChar( barp^, 1024, 255 );
          XMSCopy( Handle, longint(i)*1024, 0, longint(barp), 512 );

          {-- Compare le tampon recopié avec le résultat attendu    ---}

          j := 1;
          while ( j <= 1024 ) do
            if ( barp^[j] <> i mod 255 ) then
              begin                                                { Erreur! }
                writeln( ' ERREUR! ' );
                inc( err );
                j := 1025;
```

```
!            end
!          else                ( Pas d'erreur, cellule mémoire suivante )
!            inc( j );
!        end;
!
!      writeln( #13 );
!      if ( err = 0 ) then                ( Analyse du résultat du test )
!        writeln( 'EMB ok, aucun des blocs d''1 Ko vérifiés n''était ' +
!                 'défectueux.')
!      else
!        writeln( 'ATTENTION! ', err, ' blocs d''1 Ko défectueux' +
!                 ' dans l''EMB' );
!
!      FreeMem( barp, 1024 );                        ( Libérer le tampon )
!      XMSFreeMem( Handle );                          ( Libérer l'EMB )
!    end;
!end;
!
!{*********************************************************************
!*                  P R O G R A M M E   P R I N C I P A L            *
!*********************************************************************}
!
!var VerNr,
!    RevNr  : integer;
!
```

```
!begin
!  ClrScr;
!  writeln( 'XMSP - (c) 1990, 92 by MICHAEL TISCHER' );
!  writeln;
!  if XMSinit then
!    begin
!      if XMSQueryVer( VerNr, RevNr ) then
!        writeln( 'Accès possible à l''HMA.' )
!      else
!        writeln( 'Aucun accès à l''HMA.' );
!      writeln( 'N° de version XMS : ', VerNr div 100,
!               '.', VerNr mod 100 );
!      writeln( 'N° de révision    : ', RevNr div 100,
!               '.', RevNr mod 100 );
!      writeln;
!      HMATest;                                                ( Test HMA )
!      writeln;
!      EMBTest;                                  ( Test mémoire étendue )
!    end
!  else
!    writeln( 'Aucun pilote XMS installé!');
!end.
!
!
```

Listing : XMSC.C

```
!/*********************************************************************
!*                              X M S C . C                          *
!**-----------------------------------------------------------------**
!*  Sujet        : Démonstration de l'accès à la mémoire étendue et à *
!*                 la zone High Memory à l'aide des fonctions XMS telles*
!*                 qu'elles sont appliquées par exemple avec le pilote *
!*                 de périphériques HIMEM.SYS.                       *
!**-----------------------------------------------------------------**
!*  Auteur       : MICHAEL TISCHER                                    *
!*  Développé le : 27.07.1990                                         *
!*  Dernière MAJ : 29.03.1992                                         *
!**-----------------------------------------------------------------**
!*  (MICROSOFT C)                                                     *
!*  Création     : CL /AS /Zp xmsc.c xmsca                            *
!*  Appel        : xmsc                                               *
!**-----------------------------------------------------------------**
!*  (BORLAND TURBO C)                                                 *
!*  Création     : par un fichier de projet contenant les noms:       *
!*                 xmsc.c                                            *
!*                 xmsca.obj                                         *
!*********************************************************************/
!
!/*-- Intégrer les fichiers include ------------------------------*/
!
!
!#include <dos.h>                        /* Pour appels d'interruptions */
!#include <stdio.h>
!
!#ifdef __TURBOC__
!  #include <alloc.h>
!#else
!  #include <malloc.h>
!#endif
!
!/*-- Constantes -------------------------------------------------*/
!
!#define ERR_NOERR          0x00                    /* Pas d'erreur */
!#define ERR_NOTIMPLEMENTED 0x80          /* Fonction appelée inconnue */
!#define ERR_VDISKFOUND     0x81             /* Ram disk VDISK détecté */
!#define ERR_A20            0x82   /* Erreur sur canal d'adresses A20 */
!#define ERR_GENERAL        0x8E  /* Erreur pilote à caractère général */
!#define ERR_UNRECOVERABLE  0x8F              /* Erreur irratrapable */
!#define ERR_HMANOTEXIST    0x90                /* HMA introuvable */
!#define ERR_HMAINUSE       0x91              /* HMA déjà appelée */
!#define ERR_HMAMINSIZE     0x92          /* Taille HMA insuffisante */
!#define ERR_HMANOTALLOCED  0x93              /* HMA non affectée */
!#define ERR_A20STILLON     0x94  /* Canal d'adresses A20 encore actif */
!#define ERR_OUTOMEMORY     0xA0 /* Plus de mémoire étendue disponible */
!#define ERR_OUTOHANDLES    0xA1 /* Tous identificateurs XMS sont pris */
!#define ERR_INVALIDHANDLE  0xA2          /* Identificateur invalide */
!#define ERR_SHINVALID      0xA3    /* Identificateur source invalide */
!#define ERR_SOINVALID      0xA4          /* Décalage source invalide */
!#define ERR_DHINVALID      0xA5     /* Identif destination invalide */
!#define ERR_DOINVALID      0xA6      /* Décalage destination invalide */
!#define ERR_LENINVALID     0xA7 /* Longueur invalide pr fonction Move */
!#define ERR_OVERLAP        0xA8              /* Recouvrement interdit */
!#define ERR_PARITY         0xA9                /* Erreur de parité */
```

```
!#define ERR_EMBUNLOCKED    0xAA              /* UMB n'est pas bloqué */
!#define ERR_EMBLOCKED      0xAB             /* UMB est encore bloqué */
!#define ERR_LOCKOVERFLOW   0xAC  /* Débordement compteur blocage UMB */
!#define ERR_LOCKFAIL       0xAD    /* L'UMB ne peut pas être bloqué */
!#define ERR_UMBSIZETOOBIG  0x80        /* Plus petit UMB disponible */
!#define ERR_NOUMBS         0x81             /* Plus d'UMB disponible */
!#define ERR_INVALIDUMB     0x82  /* adresse du segment UMB invalide */
!
!#define TRUE               ( 0 == 0 )
!#define FALSE              ( 0 == 1 )
!
!/*-- Macros -----------------------------------------------------*/
!
!#ifndef MK_FP
!  #define MK_FP(seg,ofs) \
!          ((void far *) (((unsigned long)(seg) << 16) | (unsigned)(ofs)))
!#endif
!
!#define HI(x) (*((BYTE *) &x+1))                  /* HI byte d'un int */
!#define LO(x) (*((BYTE *) &x))                    /* LO byte d'un int */
!
!/*-- Déclarations de types --------------------------------------*/
!
!typedef unsigned char  BYTE;
!typedef BYTE           BOOL;
!typedef unsigned       WORD;
!
!typedef struct                      /* Informations pour appel XMS */
!        {
!          WORD AX,                /* Seuls les registres AX, BX, DX et */
!               BX,                 /* SI sont requis selon la fonction */
!               DX,                  /* appelée, il faut donc une autre */
!               SI,                           /* adresse de segment */
!               Segment;
!        } XMSRegs;
!
!typedef struct                    /* Structure Move mémoire étendue */
!        {
!          long LenB;                    /* Nbre d'octets à décaler */
!          int  SHandle;                  /* Identificateur source */
!          long SOffset;                       /* Décalage source */
!          int  DHandle;                /* Handle destination */
!          long DOffset;                /* Décalage destination */
!        } EMMS;
!
!/*-- Déclarations externes --------------------------------------*/
!
!extern void XMSCall( BYTE NumFonc, XMSRegs *Xr );
!
!/*-- Variables globales -----------------------------------------*/
!
!void far   *XMSPtr;     /* Pointeur sur Extended Memory Manager (XMM) */
!BYTE       XMSErr;        /* Code d'erreur de la dernière opération */
!
!/*********************************************************************
!* XMSInit : Initialise les routines d'appel des fonctions XMS       *
!**-----------------------------------------------------------------**
```

```
/* Entrée : aucune                                                    */
/* Sortie : TRUE si pilote XMS identifié sinon FALSE                  */
/* Info   : - L'appel de cette fonction doit précéder celui de toutes */
/*            les autres procédures et fonctions émanant de ce        */
/*            programme.                                              */
/**********************************************************************/

BOOL XMSInit( void )
{
 union REGS  Regs;  /* Registre processeur pour appel interruptions */
 struct SREGS SRegs;                          /* Registre segment */
 XMSRegs      Xr;                        /* Registre pour appel XMS */

 Regs.x.ax = 0x4300;          /* Détecte disponibilité du XMS Manager */
 int86( 0x2F, &Regs, &Regs );              /* Appel du DOS Dispatcher */

 if( Regs.h.al == 0x80 )                     /* Détecté XMS-Manager? */
                                                          /* Oui */
   Regs.x.ax = 0x4310;                /* Retourner point d'accès au XMM */
   int86x( 0x2F, &Regs, &Regs, &SRegs );
   XMSPtr = MK_FP( SRegs.es, Regs.x.bx ); /*Copie adresse ds var glob.*/
   XMSErr = ERR_NOERR;                          /* Pas d'erreur apparue */
   return TRUE;                       /* Trouvé handler, module initialisé */
 }
 else                               /* XMS Handler n'est pas installé */
   return FALSE;
}

/**********************************************************************/
/* XMSQueryVer: Renvoie le nº de version de l'XMS et autres informations*/
/*              sur l'état                                            */
/**--------------------------------------------------------------------**/
/* Entrée : VerNr = Stocke nº de version après appel de la fonction   */
/*                  (Format: 235 == 2.35)                             */
/*          RevNr = Stocke nº de révision après appel de la fonction  */
/* Sortie : TRUE si un HMA est disponible sinon FALSE                 */
/**********************************************************************/

BOOL XMSQueryVer( int * VerNr, int * RevNr)
{
 XMSRegs Xr;                                  /* Registre pour appel XMS */

 XMSCall( 0, &Xr );                           /* Appeler fonction XMS #0 */
 *VerNr = Hi(Xr.AX)*100 + ( Lo(Xr.AX) >> 4 ) * 10 +
          ( Lo(Xr.AX) & 15 );
 *RevNr = Hi(Xr.BX)*100 + ( Lo(Xr.BX) >> 4 ) * 10 +
          ( Lo(Xr.BX) & 15 );
 return( Xr.DX == 1 );
}

/**********************************************************************/
/* XMSGetHMA : Retourner à l'appelant le droit d'accès à la HMA.      */
/**--------------------------------------------------------------------**/
/* Entrée  : LenB = Nombre d'octets à allouer                         */
/* Info    : Les programmes résidents devraient réserver exclusivement*/
/*           la mémoire strictement requise. Par contre, donner 0xFFFF*/
/*           aux applications.                                        */
/* Sortie  : TRUE si la HMA a pu être rendue disponible sinon FALSE;  */
/**********************************************************************/

BOOL XMSGetHMA( WORD LenB )
{
 XMSRegs Xr;                                  /* Registre pour appel XMS */

 Xr.DX = LenB;                        /* Stocke longueur dans registre DX */
 XMSCall( 1, &Xr );                           /* Appel fonction XMS #1 */
 return( (XMSErr = (Xr.AX==1) ? ERR_NOERR : Xr.BX) , Xr.AX==1 );
}

/**********************************************************************/
/* XMSReleaseHMA : Libère l'HMA et permet ainsi sa transmission à     */
/*                 d'autres programmes.                               */
/**--------------------------------------------------------------------**/
/* Entrée : aucune                                                    */
/* Info   : - Appeler cette procédure avant de quitter un programme   */
/*            si la HMA a été allouée par un appel de XMSGetHMA pour  */
/*            pouvoir la transmettre aux programmes appelés plus tard.*/
/*          - L'appel de cette procédure entraîne la perte des        */
/*            données stockées dans la HAM.                           */
/**********************************************************************/

void XMSReleaseHMA( void )
{
 XMSRegs Xr;                                  /* Registre pour appel XMS */

 XMSCall( 2, &Xr );                           /* Appel fonction XMS #2 */
}

/**********************************************************************/
/* XMSA20OnGlobal: Libère le canal d'adresses A20, permettant l'accès */
/*                 direct à la HMA.                                   */
/**--------------------------------------------------------------------**/
/* Entrée : aucune                                                    */
/* Info   : - La libération du canal d'adresses A20 est relativement  */
/*            lente sur de nombreux ordinateurs. Veillez à ne pas     */
/*            abuser de l'usage de cette procédure.                   */
/**********************************************************************/

void XMSA20OnGlobal( void )
{
 XMSRegs Xr;                                  /* Registre pour appel XMS */

 XMSCall( 3, &Xr );                           /* Appel fonction XMS #3 */
}

/**********************************************************************/
/* XMSA20OffGlobal: Pendant de la procédure XMSA20OnGlobal, celle-ci  */
/*                  bloque à nouveau le canal d'adresses A20, interdi-*/
/*                  sant l'accès direct à la HMA                      */
/**--------------------------------------------------------------------**/
/* Entrée : aucune                                                    */
/* Info   : - Appelez toujours cette procédure avant de quitter un    */
/*            programme si le canal d'adresses A20 a été libéré par   */
/*            un appel de XMSA20OnGlobal                              */
/**********************************************************************/

void XMSA20OffGlobal( void )
{
 XMSRegs Xr;                                  /* Registre pour appel XMS */

 XMSCall( 4, &Xr );                           /* Appel fonction XMS #4 */
}

/**********************************************************************/
/* XMSA20OnLocal: Voir XMSA20OnGlobal                                 */
/**--------------------------------------------------------------------**/
/* Entrée : aucune                                                    */
/* Info   : - Cette procédure locale se distingue de sa variante      */
/*            globale par la libération du canal: elle n'est possible */
/*            que si elle n'a pas été effectuée par un appel précédent*/
/**********************************************************************/

void XMSA20OnLocal( void )
{
 XMSRegs Xr;                                  /* Registre pour appel XMS */

 XMSCall( 5, &Xr );                           /* Appel fonction XMS #5 */
}

/**********************************************************************/
/* XMSA20OffLocal : Voir XMSA29OffGlobal                              */
/**--------------------------------------------------------------------**/
/* Entrée : aucune                                                    */
/* Info   : - Cette procédure locale se distingue de sa variante      */
/*            globale par la libération du canal: elle n'est possible */
/*            que si elle n'a pas été effectuée par un appel précédent*/
/**********************************************************************/

void XMSA20OffLocal( void )
{
 XMSRegs Xr;                                  /* Registre pour appel XMS */

 XMSCall( 6, &Xr );                           /* Appel fonction XMS #6 */
}

/**********************************************************************/
/* XMSIsA20On : Retourne l'état du canal d'adresse A20                */
/**--------------------------------------------------------------------**/
/* Entrée : aucune                                                    */
/* Sortie : TRUE si le canal d'adresses A20 est libéré sinon FALSE.   */
/**********************************************************************/

BOOL XMSIsA20On( void )
{
 XMSRegs Xr;                                  /* Registre pour appel XMS */

 XMSCall( 7, &Xr );                           /* Appel fonction XMS #7 */
 return( Xr.AX == 1 );                      /* AX == 1 ---> Canal libre */
}

/**********************************************************************/
/* XMSQueryFree : Renvoie la mémoire étendue disponible et la taille du*/
/*                plus gros bloc libre                                */
/**--------------------------------------------------------------------**/
/* Entrée : TotalLibre: Stocke la taille totale de l'EM libre.        */
/*          MaxBl   : Stocke la taille du plus grand bloc libre.      */
/* Info   : - Les deux valeurs sont en Ko                             */
/*          - La taille de la HMA n'est pas comptabilisée même si elle*/
/*            n'a été affectée à aucun programme.                     */
/**********************************************************************/

void XMSQueryFree( int * TotalLibre, int * MaxBl )
{
 XMSRegs Xr;                                  /* Registre pour appel XMS */

 XMSCall( 8, &Xr );                           /* Appel fonction XMS #8 */
 *TotalLibre = Xr.AX;                 /* La taille totale est dans AX */
 *MaxBl      = Xr.DX;                 /* La mémoire libre est dans DX */
```

```
!}
!
!/*****************************************************************
!* XMSGetMem : Alloue un bloc de mémoire étendue (EMB)           *
!**---------------------------------------------------------------**
!* Entrée : LenKB : Taille en Ko du bloc interrogé               *
!* Sortie : Identificateur pour accès au bloc ou 0, si aucun bloc n'a *
!*          pu être alloué. Un code d'erreur se trouve alors dans la *
!*          variable globale XMSErr.                             *
!*****************************************************************/
!
!int XMSGetMem( int LenKb )
!{
! XMSRegs Xr;                                  /* Registre pour appel XMS */
!
! Xr.DX = LenKb;            /* Stockons la longueur dans le registre DX */
! XMSCall( 9, &Xr );                          /* Appel fonction XMS @9 */
! return Xr.DX;                               /* Retourne l'identificateur */
!}
!
!/*****************************************************************
!* XMSFreeMem : Libère un bloc de mémoire étendue (EMB) précédemment *
!*              alloué                                           *
!**---------------------------------------------------------------**
!* Entrée : Handle : L'identificateur pour accéder au bloc. Il a été *
!*                   obtenu en appelant XMSGetMem.               *
!* Info   : - Le contenu de l'EMB est définitivement détruit par cet *
!*            appel, l'identificateur devient invalide.          *
!*          - Avant de quitter un programme, libérez à l'aide de cette *
!*            procédure toutes les zones précédemment allouées pour *
!*            pouvoir les allouer aux programmes suivants.       *
!*****************************************************************/
!
!void XMSFreeMem( int Handle )
!{
! XMSRegs Xr;                                  /* Registre pour appel XMS */
!
! Xr.DX = Handle;          /* Copions l'identificateur dans le registre DX */
! XMSCall( 10, &Xr );                         /* Appel fonction XMS @10 */
!}
!
!/*****************************************************************
!* XMSCopy : Copie des zones de mémoire entre la mémoire étendue et la *
!*           mémoire conventionnelle ou à l'intérieur de ces deux *
!*           groupes de mémoire.                                 *
!**---------------------------------------------------------------**
!* Entrée : HandleOrig  : Identificateur du bloc de mémoire à déplacer *
!*          OffsetOrig  : Offset dans ce bloc, à partir duquel le *
!*                        déplacement sera effectué.            *
!*          HandleDest  : Identifficateur du bloc de mémoire cible *
!*          OffsetDest  : Offset dans le bloc cible.            *
!*          LenW        : Nombre de mots déplacés               *
!* Info   : - Pour utiliser la mémoire normale dans cette opération, *
!*            donnez la valeur 0 à l'identificateur ("Handle") et pour *
!*            l'offset, le segment et l'adresse de l'offset dans sa *
!*            forme habituelle (offset avant le segment).       *
!*****************************************************************/
!
!void XMSCopy( int HandleOrig, long OffsetOrig, int HandleDest,
!              long OffsetDest, int LenW )
!
!{
! XMSRegs Xr;                                  /* Registre pour appel XMS */
! EMMS MI;                                     /* Stocke l'EMMS */
! void far * MIPtr;
!
! MI.LenB    = 2 * LenW;                  /* Commencer par préparer l'EMMS */
! MI.SHandle = HandleOrig;
! MI.SOffset = OffsetOrig;
! MI.DHandle = HandleDest;
! MI.DOffset = OffsetDest;
!
! MIPtr      = &MI;                        /* Pointeur FAR sur la structure */
! Xr.SI      = FP_OFF( MIPtr );            /* Adresse de l'offset de l'EMMS */
! Xr.Segment = FP_SEG( MIPtr );            /* Adresse du segment de l'EMMS */
! XMSCall( 11, &Xr );                         /* Appel fonction XMS @11 */
!}
!
!/*****************************************************************
!* XMSLock : Interdit tout décalage d'un bloc de mémoire étendue par *
!*           l'XMM et retourne son adresse absolue.              *
!**---------------------------------------------------------------**
!* Entrée : Handle : Identificateur du bloc de mémoire retourné lors de *
!*          l'appel précédent de XMSGetMem.                      *
!* Sortie : Adresse linéaire du bloc de mémoire.                 *
!*****************************************************************/
!
!long XMSLock( int Handle )
!{
! XMSRegs Xr;                                  /* Registre pour appel XMS */
!
! Xr.DX = Handle;                              /* Handle de l'EMB */
! XMSCall( 12, &Xr );                         /* Appel fonction XMS @12 */
! return( ((long) Xr.DX << 16) + Xr.BX);    /* Calcule adresse 32 bits */
!}
!
!/*****************************************************************
!* XMSUnlock : Libère à nouveau un bloc de mémoire étendu pour une *
!*             opération de décalage.                            *
!**---------------------------------------------------------------**
!* Entrée : Handle : Identificateur de la zone de mémoire retourné lors *
!*          d'un appel précédent de XMSGetMem.                   *
!*****************************************************************/
!
!void XMSUnlock( int Handle )
!{
! XMSRegs Xr;                                  /* Registre pour appel XMS */
!
! Xr.DX = Handle;                              /* Handle de l'EMB */
! XMSCall( 13, &Xr );                         /* Appel fonction XMS @13 */
!}
!
!/*****************************************************************
!* XMSQueryInfo : Retourne diverses informations sur un bloc de  *
!*                mémoire étendue préalablement alloué.          *
!**---------------------------------------------------------------**
!* Entrée :  Handle : Identificateur de la zone de mémoire       *
!*           Lock   : Variable de stockage du compteur de Lock   *
!*           LenKB  : Variable de stockage de la longueur du bloc en Ko *
!*           FreeH  : Variable de stockage du nombre d'identificateurs *
!*                    restant libres.                            *
!* Info   : Cette procédure ne permet pas de connaître l'adresse d'un *
!*          bloc. Utilisez la fonction XMSLock pour cette information. *
!*****************************************************************/
!
!void XMSQueryInfo( int Handle, int * Lock, int * LenKB, int * FreeH )
!
!{
! XMSRegs Xr;                                  /* Registre pour appel XMS */
!
! Xr.DX  = Handle;                             /* Handle de l'EMB */
! XMSCall( 14, &Xr );                         /* Appel fonction XMS @14 */
! *Lock  = Hi( Xr.BX );                       /* Lit les registres */
! *FreeH = Lo( Xr.BX );
! *LenKB = Xr.DX;
!}
!
!/*****************************************************************
!* XMSRealloc : Agrandit ou réduit la taille d'un bloc de mémoire *
!*              étendue alloué par XMSGetMem                     *
!**---------------------------------------------------------------**
!* Entrée :  Handle   : Identificateur de la zone de mémoire     *
!*           NewLenKB : Nouvelle taille du bloc, en Ko           *
!* Sortie :  TRUE si la taille du bloc a été modifiée sinon FALSE *
!* Info   :  Ce bloc ne doit pas être verrouillé!                *
!*****************************************************************/
!
!BOOL XMSRealloc( int Handle, int NewLenKB)
!{
! XMSRegs Xr;                                  /* Registre pour appel XMS */
!
! Xr.DX = Handle;                              /* Handle de l'EMB */
! Xr.BX = NewLenKB;        /* La nouvelle longueur dans le registre BX */
! XMSCall( 15, &Xr );                         /* Appel fonction XMS @15 */
! return( XMSErr == ERR_NOERR );
!}
!
!/*****************************************************************
!* XMSGetUMB : Allous un bloc de Upper Memory (UMB)              *
!**---------------------------------------------------------------**
!* Entrée :  LenPara : Taille de la zone allouée en paragraphes de 16 *
!*                     octets chacun.                            *
!*           Seg     : Variable de stockage de l'adresse du segment de *
!*                     l'UMB alloué (si tout va bien)            *
!*           MaxPara : Variable contenant la taille du plus gros bloc *
!*                     UMB en cas d'échec.                       *
!* Sortie : TRUE si un UMB a été alloué sinon FALSE              *
!* Info   : Attention! Cette fonction n'est pas compatible avec tous *
!*          les pilotes XMS. Elle est extrêmement dépendante du  *
!*          matériel.                                            *
!*****************************************************************/
!
!BOOL XMSGetUMB( int LenPara, WORD * Seg, WORD * MaxPara )
!{
! XMSRegs Xr;                                  /* Registre pour appel XMS */
!
! Xr.DX    = LenPara;                       /* Longueur demandée selon DX */
! XMSCall( 16, &Xr );                         /* Appel fonction XMS @16 */
! *Seg     = Xr.BX;                        /* Retourne l'adresse du segment */
! *MaxPara = Xr.DX;                           /* Longueur du plus gros UMB */
! return ( XMSErr == ERR_NOERR );
!}
!
!/*****************************************************************
!* XMSFreeUMB : Libère un UMB alloué par XMSGetUMB.              *
!**---------------------------------------------------------------**
```

```
/* Entrée  : Seg : Adresse du segment de l'UMB à libérer              */
/* Info    : Attention! Cette fonction n'est pas compatible avec tous */
/*           les pilotes XMS. Elle est extrêmement dépendante du      */
/*           matériel.                                                */
/*************************************************************************/

void XMSFreeUMB( WORD Seg )

{
 XMSRegs Xr;                                  /* Registre pour appel XMS */

 Xr.DX = Seg;                        /* Adresse du segment de l'UMB selon DX */
 XMSCall( 17, &Xr );                          /* Appel fonction XMS #17 */
}

/*-----------------------------------------------------------------------*/
/*-- Procédures de test et de démonstration                            --*/
/*-----------------------------------------------------------------------*/

/*************************************************************************/
/* HMATest : Teste la disponibilité de l'HMA et démontre son maniement.  */
/*-----------------------------------------------------------------------*/
/* Entrée : aucune                                                       */
/*************************************************************************/

void HMATest( void)

{
 BOOL A20;                          /* Etat courant du canal d'adresse A20 */
 BYTE far * hmap;                                 /* Pointeur sur l'HMA */
 WORD i,                                        /* Compteur d'itérations */
      err;                              /* Nombre d'erreurs d'accès à la HMA */

 printf( "Test HMA : Tapez une touche pour lancer le test..." );
 getch();
 printf("\n\n" );

 /*-- Allouer HMA et tester chaque adresse de la mémoire -------------*/

 if( XMSGetHMA(0xFFFF) )                          /* Contrôlons-nous l'HMA? */
 {                                                              /* Oui */
   if( ( A20 = XMSIsA20On() ) == FALSE )    /* Donner l'état du canal */
     XMSA20OnGlobal();                                  /* le libérer */

   hmap = MK_FP( 0xFFFF, 0x0010 );                  /* Pointeur sur HMA */
   err  = 0;                              /* Jusque là encore aucune erreur */
   for( i = 1; i < 65520; ++i, ++hmap )
   {                                  /* Tester chaque cellule séparément */
     printf( "\rCellule mémoire: %u", i );
     *hmap = i % 256;                          /* Ecrire à cette adresse */
     if( *hmap != i % 256 )                              /* et la relire */
     {                                                          /* Erreur! */
       printf( " ERREUR!\n" );
       ++err;
     }
   }

   XMSReleaseHMA();                                       /* Libérons l'HMA */
   if( A20 == FALSE )             /* Est-ce que le canal A20 était libre? */
     XMSA20OffGlobal();                                   /* Non, libérons-le */

   printf( "\n" );
   if( err == 0 )                              /* Analyse du résultat du test */
     printf( "HMA ok, aucune cellule de la mémoire défectueuse.\n" );
   else
     printf( "ATTENTION! %d cellules défectueuses dans l'HMA\n", err );
 }
 else
   printf( "ATTENTION! Accès impossible à l'HMA.\n" );
}

/*************************************************************************/
/* EMBTest : Teste la mémoire étendue et montre l'appel de plusieurs     */
/*           fonctions XMS                                               */
/*-----------------------------------------------------------------------*/
/* Entrée : aucune                                                       */
/*************************************************************************/

void EMBTest( void )

{
 long Adr;
 BYTE * barp;                                 /* Pointeur sur tampon d'un Ko */
 int  i, j,                                     /* Compteur d'itérations */
      err,                          /* Nombre d'erreurs pendant l'accès à l'HMA */
      Handle,                         /* Identificateur pour accéder à l'EMB */
      TotalLibre,                      /* Taille de toute la mémoire étendue */
      MaxBl;                                    /* Plus gros bloc libre */

 printf( "Test EMB : Tapez une touche pour lancer le test..." );
 getch();
 printf( "\n" );

 XMSQueryFree( &TotalLibre, &MaxBl ); /* Retourner taille mém. étend. */
 printf( "Taille totale de la mémoire étendue (avec HMA) : %d KB\n",
         TotalLibre );
 printf( "                    Dont le plus gros bloc libre : %d KB\n",
         MaxBl );

 TotalLibre -= 64;                    /* Calcul taille effective sans HMA. */
 if( MaxBl >= TotalLibre )                        /* Valeur vraisemblable? */
   MaxBl -= 64;                                                 /* Non */

 if( MaxBl > 0 )                                /* Encore assez de mémoire? */
 {                                                              /* Oui */
   Handle = XMSGetMem( MaxBl );
   printf( "%d Ko alloués.\n", MaxBl );
   printf( "Identificateur = %d\n", Handle );
   Adr    = XMSLock( Handle );                      /* Retourner l'adresse */
   XMSUnlock( Handle );                             /* Supprimer blocage */
   printf( "Adresse = %ld (%d KB)\n", Adr, Adr >> 10 );

   barp   = malloc( 1024 );                       /* Le tampon sur le tas... */
   err    = 0;                                /* Jusque là encore aucune erreur */

       /*-- Vérifier l'EMB Ko après Ko ------------------------------*/

   for( i = 0; i < MaxBl; ++i )
   {
     printf( "\rTest du Ko: %d", i+1 );
     memset( barp, i % 255, 1024 );
     XMSCopy( 0, (long) ((void far *) barp),
              Handle, (long) i*1024, 512 );
     memset( barp, 255, 1024 );
     XMSCopy( Handle, (long) i*1024, 0,
              (long) ((void far *) barp), 512 );

     /*-- Compare le tampon recopié avec le résultat attendu ---------*/

     for( j = 0; j < 1024; ++j )
       if( *(barp+j) != i % 255 )
       {                                                        /* Erreur! */
         printf( " ERREUR!\n" );
         ++err;
         break;
       }
   }

   printf( "\n" );
   if( err == 0 )                               /* Analyse du résultat du test */
     printf( "EMB ok, aucun des blocs testés d'un Ko était " \
             "défectueux.\n");
   else
     printf( "ATTENTION! %d blocs d'1 Ko défectueux dans l'EMB\n",err );

   free( barp );                                          /* Libérer le tampon */
   XMSFreeMem( Handle );                                   /* Libérer l'EMB */
 }
}

/*************************************************************************/
/*                          P R O G R A M M E   P R I N C I P A L        */
/*************************************************************************/

void main( void )
{
 int VerNr,                                                 /* N° de version */
     RevNr,                                                 /* N° de révision */
     i;                                          /* Compteur d'itérations */

 for( i = 1; i < 25; ++i )                                  /* Effacer l'écran */
   printf ( "\n" );

 printf("XMSC - (c) 1990, 92 by MICHAEL TISCHER\n\n" );
 if( XMSInit() )
 {
   if( XMSQueryVer( &VerNr, &RevNr ) )
     printf( "Accès possible à l'HMA.\n" );
   else
     printf( "Aucun accès à l'HMA.\n" );
   printf( "N° de version XMS: %d.%d\n", VerNr / 100, VerNr % 100 );
   printf( "N° de révision   : %d.%d\n\n", RevNr / 100, RevNr % 100 );
   HMATest();                                                   /* Test HMA */
   printf( "\n" );
   EMBTest();                                        /* Test mémoire étendue */
 }
 else
   printf( "Aucun pilote XMS installé!\n" );
}
```

Listing : XMSCA.ASM

```
;*****************************************************************;
;*                          X M S C A . A S M                    *;
;*---------------------------------------------------------------*;
;*   Sujet          : Routine en assembleur destinée à être intégrée *;
;*                    au programme XMSC.C. Permet de disposer d'une  *;
;*                    routine d'appel du pilote XMS.                 *;
;*                    Cette implémentation est destinée au modèle de *;
;*                    mémoire SMALL.                                 *;
;*---------------------------------------------------------------*;
;*   Auteur         : MICHAEL TISCHER                              *;
;*   développé le   : 27.07.1990                                   *;
;*   dernière m. à j: 27.07.1990                                   *;
;*---------------------------------------------------------------*;
;*   assembler      : MASM XMSCA;   ou    TASM XMSCA;             *;
;*                    ... ensuite lier au programme C compilé      *;
;*                    XMSC.C                                       *;
;*****************************************************************;

;-- Déclarations des segments pour le programme en C ------------

IGROUP group _text                  ;Regroupement des segments programme
DGROUP group _bss, _data            ;Regroupement des segments de données
       assume CS:IGROUP, DS:DGROUP, ES:DGROUP, SS:DGROUP

_BSS   segment word public 'BSS' ;Ce segment contient ttes les variables
_BSS   ends                             ;statiques non initialisées

_DATA  segment word public 'DATA'     ;Toutes les variables initialisées
                                 ;globales et statiques sont regroupées
                                                    ;dans ce segment

extrn _XMSPtr : dword                          ;Référence au pointeur XMS

_DATA  ends

;-- Programme ---------------------------------------------------

_TEXT  segment byte public 'CODE'                ;Le segment de programme

public    _XMSCall

;-----------------------------------------------------------------
;-- XMSCall : Routine générale d'appel d'une fonction XMS
;-- Appel C : void XMSCall( BYTE NumFonct, XMSRegs *Xr ) avec
;--           typedef struct ( WORD AX, BX, DX, SI, Segment ) XMSRegs;
;-- Retour  : aucun
;-- Info    : - Avant l'appel de cette procédure, charger excusivement
;--             les registres effectivement nécessaires à l'appel de la
;--             fonction.
;--       - Après l'appel de la fonction XMS, les contenus des registres
;--         du processeur sont copiés dans les composant correspondants
;--             de la structure renvoyée.
;--       - Le premier appel de cette procédure doit être précédé d'un
;--             appel en bon ordre de la procédure XMSInit.

_XMSCall   proc near

sframe     struc                        ;Structure pour accéder à la pile
bp0        dw ?                                            ;stocke BP
ret_adr    dw ?                      ;Adresse de retour pour l'appelant
NumFonc    dw ?                             ;Numéro de la fonction XMS
xrptr      dw ?                  ;Pointeur sur la structure des registres
sfram      ends

frame  equ [ bp - bp0 ]            ;adresse les éléments de la structure

       push   bp                                  ;Copions BP sur la pile
       mov    bp,sp                                 ;Passons SP après BP

       push   si                                ;Sauvegarde de SI et de DI
       push   di

       mov    cx,ds                                       ;Copions DS dans CX
       push   cx                                 ;et sauvegarder sur la pile
       mov    di,frame.xrptr                  ;Chargement du n° de fonction
       mov    ah,byte ptr frame.NumFonc  ;Chargement du ptr sur la struct
       mov    bx,[di+2]              ;Chargement des registres stockés dans
       mov    dx,[di+4]                    ;les composants de la structure
       mov    si,[di+6]
       mov    ds,[di+8]
       mov    es,cx                                    ;Charge ES avec DS
       call   es:[_XMSPtr]                          ;Appel du Handler XMS
       mov    cx,ds                                     ;Stocke DS dans CX
       pop    ds                                  ;Récupérons l'ancien DS
       mov    di,frame.xrptr      ;Chargement du pointeur sur la structure
       mov    [di],ax                        ;Entre les registres dans leurs
       mov    [di+02],bx                          ;composants respectifs de
       mov    [di+04],dx                                  ;de la structure
       mov    [di+06],si
       mov    [di+08],cx

       pop    di                                       ;Récupère SI et DI
       pop    si

       pop    bp                              ;Récupère BP qui était sur la pile
       ret                                          ;Retour au programme en C

_XMSCall   endp

;-----------------------------------------------------------------

_text      ends                                  ;Fin du segment de code
           end                                       ;Fin du programme
```

13. Le son sur le PC

Tous les PC disposent d'un haut-parleur intégré, grâce auquel est produit le bip, que vous connaissez certainement très bien, qui retentit lorsque le buffer clavier est plein et que vous frappez encore une touche. Le haut-parleur ne permet toutefois pas uniquement de produire des bips mais aussi toutes sortes de notes. Ce chapitre a pour but de vous expliquer comment cela peut être réalisé sur le plan logiciel.

Dans ce chapitre, vous apprendrez à :

- ✓ identifier l'électronique qui se cache derrière le haut-parleur pour créer le son,
- ✓ programmer l'électronique pour créer le son et
- ✓ saurez comment un haut-parleur crée un signal sonore.

Création d'un signal dans un haut-parleur

Pour faire retentir des bruits ou des sons sur une machine électronique, il faut que la membrane d'un haut-parleur avance et recule rapidement pour produire ce qu'on appelle une vibration. Une vibration isolée ne produira cependant aucune note mais un simple "clic" étouffé. Si par contre cette vibration est suivie d'autres vibrations se succédant à intervalles rapprochés, on obtiendra de véritables notes de musique. La hauteur de ces notes dépendra de la fréquence de ces vibrations, autrement dit du nombre de vibrations par seconde. L'unité de mesure des fréquences est le Hertz, qui correspond à une vibration par seconde. Si la membrane du haut-parleur avance et recule 440 fois en une seconde, elle produira par exemple une note d'une fréquence de 440 Hertz. Sur le plan musical, chaque note de la gamme correspond à une fréquence bien précise. C'est ainsi que les normes internationales définissent la note 'la' juste au-dessus de la clé de sol comme une fréquence de 440 Hertz. La table suivante vous présente les fréquences des notes de la gamme chromatique pour les 8 octaves du clavier :

Octave 0		Octave 1		Octave 2		Octave 3	
Do	16,35	Do	32,70	Do	65,41	Do	130,81
Do#	17,32	Do#	34,65	Do#	69,30	Do#	138,59
Ré	18,35	Ré	36,71	Ré	73,42	Ré	146,83
Ré#	19,45	Ré#	38,89	Ré#	77,78	Ré#	155,56
Mi	20,60	Mi	41,20	Mi	82,41	Mi	164,81
Fa	21,83	Fa	43,65	Fa	87,31	Fa	174,61
Fa#	23,12	Fa#	46,25	Fa#	92,50	Fa#	185,00
Sol	24,50	Sol	49,00	Sol	98,00	Sol	196,00
Sol#	25,96	Sol#	51,91	Sol#	103,83	Sol#	207,65
La	27,50	La	55,00	La	110,00	La	220,00
La#	29,14	La#	58,27	La#	116,54	La#	233,08
Si	30,87	Si	61,74	Si	123,47	Si	246,94
Octave 4		**Octave 5**		**Octave 6**		**Octave 7**	
Do	261,63	Do	523,25	Do	1046,50	Do	2093,00
Do#	277,18	Do#	554,37	Do#	1108,74	Do#	2217,46
Ré	293,66	Ré	587,33	Ré	1174,66	Ré	2349,32
Ré#	311,13	Ré#	622,25	Ré#	1244,51	Ré#	2489,02
Mi	329,63	Mi	659,26	Mi	1328,51	Mi	2637,02
Fa	349,23	Fa	698,46	Fa	1396,91	Fa	2793,83
Fa#	369,99	Fa#	739,99	Fa#	1479,98	Fa#	2959,96
Sol	392,00	Sol	783,99	Sol	1567,98	Sol	3135,96
Sol#	415,30	Sol#	830,61	Sol#	1661,22	Sol#	3322,44
La	440,00	La	880,00	La	1760,00	La	3520,00
La#	466,16	La#	923,33	La#	1864,66	La#	3729,31
Si	493,88	Si	987,77	Si	1975,53	Si	3951,07

Le haut-parleur de votre PC peut produire des fréquences de 1 Hertz à 1 000 000 Hertz mais notre ouïe ne perçoit que les fréquences comprises entre 20 Hertz et 20 000 Hertz. Malgré son spectre de fréquences impressionnant, le haut-parleur de votre PC ne se prête absolument pas à une exploitation musicale. C'est ainsi par exemple qu'il joue certaines notes (certaines fréquences) systématiquement plus fort que d'autres. Or cet effet ne peut être compensé car il n'est pas possible de régler le volume du son.

Programmation de l'électronique du son

Pour produire des notes, un programme devra donc faire vibrer la membrane de haut-parleur à la fréquence exigée par la note voulue. Le haut-parleur dispose d'un port (l'adresse 61h) à travers lequel on peut lui indiquer si la membrane doit se mouvoir en avant ou en arrière. Un programme de production de sons pourrait donc se présenter ainsi :

On donne tout d'abord l'ordre de déplacer la membrane vers l'avant. Peu après, l'ordre est rapporté, ce qui fait revenir la membrane en arrière. On revient ensuite à la première étape, à l'intérieur d'une boucle. Cette opération doit cependant être gérée dans le temps de façon à être répétée le nombre de fois nécessaire au cours d'une seconde pour produire la fréquence de la note voulue.

Cette méthode a cependant plusieurs inconvénients : la vitesse de traitement des différentes instructions dépend en effet de la fréquence d'horloge de l'ordinateur, de sorte que ce programme doit être adapté à la fréquence d'horloge de chaque ordinateur. D'autre part, cette boucle risque d'être interrompue par une interruption (par exemple lorsqu'une touche est actionnée), ce qui entraîne une distorsion du son. C'est pourquoi on utilise pour produire des sons un des nombreux circuits intégrés dont dispose chaque PC, le 8253, qui permet de résoudre le problème indiqué. Le 8253 est un temporisateur programmable qui est au départ chargé, sur le PC, de gérer l'horloge interne. Le fait qu'il puisse également permettre de faire de la musique n'est au fond qu'un heureux effet secondaire. Dans les deux cas, on utilise sa faculté de déclencher une action déterminée au bout d'un laps de temps donné. Pour "savoir" qu'un certain temps s'est écoulé, encore faut-il bien sûr qu'il connaisse l'heure. Cela est obtenu sur le PC en retransmettant au 8253 les vibrations du coeur du PC, c'est-à-dire d'un oscillateur du type 8284, qui produit 1 193 180 impulsions par seconde. Il suffit ensuite d'indiquer au 8253 après combien de ces impulsions une action déterminée devra être déclenchée. Dans le cas de la génération de sons, cette action consiste à envoyer une impulsion déterminée au haut-parleur. Avant de lui donner cet ordre, nous devons tout d'abord le programmer sur la fréquence à laquelle le signal devra être produit. L'unité de mesure qui nous sert à indiquer une fréquence, les "vibrations par seconde", ne signifie cependant rien pour lui. Nous devons donc tout d'abord convertir la fréquence en un nombre de vibrations de l'oscillateur. On utilise pour cela la formule suivante :

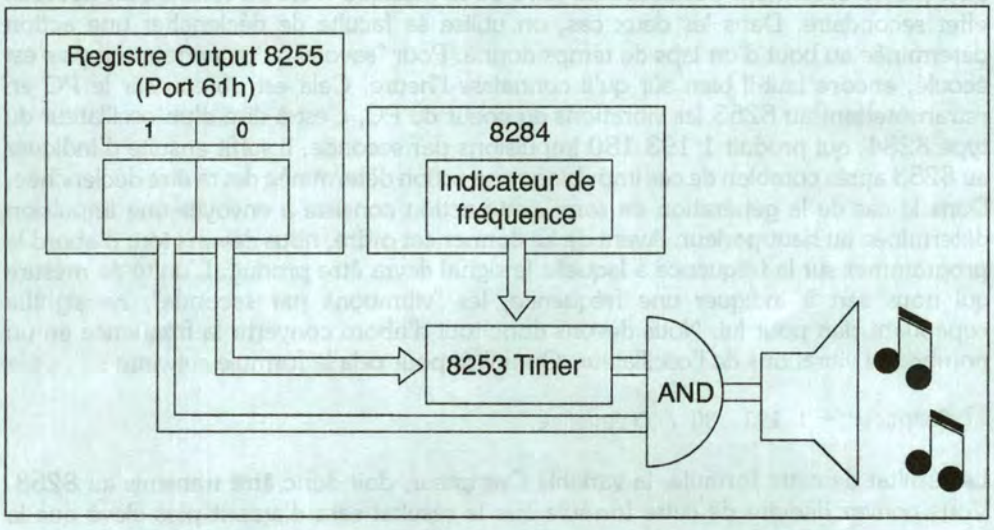
```
Compteur = 1 193 180 / fréquence
```

Le résultat de cette formule, la variable Compteur, doit donc être transmis au 8253. Vous pouvez déduire de cette formule que le résultat sera d'autant plus élevé que la fréquence sera basse et inversement. Cela est d'ailleurs parfaitement logique puisque cette valeur indique au 8253 combien de vibrations sur les 1 193 180 par seconde il devra attendre avant d'envoyer à nouveau un signal au haut-parleur. Plus cette valeur sera faible et plus le signal sera envoyé souvent, plus souvent donc la membrane avancera et reculera et plus la note sera haute.

Programmation du port 8253

La communication entre l'unité centrale se fait encore une fois à travers des ports. La valeur 182 est tout d'abord envoyée sur le port 43h. Le 8253 est ainsi prévenu qu'il doit commencer à générer un signal cyclique dès que l'intervalle entre deux signaux lui aura été communiqué dans la deuxième phase de l'opération. Cet intervalle est défini par la valeur calculée à l'aide de la formule indiquée plus haut. Comme le 8253 stocke cette valeur, sur un plan interne, sous la forme d'un nombre de 16 bits (soit une valeur entre 0 et 65535), le spectre des fréquences pouvant être produites est limité de 18 à 1 193 180 Hertz. Ce nombre 16 bits doit être transféré sur le port 42h. Mais comme ce port est un port 8 bits, les 16 bits de ce nombre doivent être transférés en deux temps, tout d'abord les 8 bits de plus faible poids, suivis des 8 bits de plus fort poids.

Nous en arrivons alors à la seconde étape, qui consiste à retransmettre le signal du 8253 au haut-parleur. L'accès au haut-parleur se fait à travers le port 61h qui est relié à un circuit périphérique programmable. Pour retransmettre le signal du 8253 au haut-parleur, les deux bits inférieurs de ce port doivent être fixés sur 1. Comme les 6 autres bits sont utilisés à d'autres fins, il ne faut cependant pas les modifier. C'est pourquoi il convient tout d'abord de lire le contenu du port 61h, de fixer les deux bits inférieurs sur 1 (ce qui équivaut à une combinaison OU avec 3) et de renvoyer la valeur résultante sur le port 61h. A partir de ce moment retentit un son qui ne s'arrêtera que lorsque les bits qui viennent juste d'être fixés sur 1 seront à nouveau fixés sur 0.



Génération de sons

Exemples de programmes

Si aussi bien le QuickBasic que Turbo Pascal disposent d'instructions pour sortir des sons, la bibliothèque du compilateur C de Microsoft ne comporte pas d'instruction de ce type et le programmeur en assembleur doit également les reconstituer.

C'est pourquoi nous proposons pour ces deux groupes d'utilisateurs des programmes d'exemple qui servent l'un à intégrer une fonction Sound en C, l'autre à organiser une sortie sonore commode en assembleur.

Ces deux programmes vont au-delà de la simple activation et désactivation de sons puisqu'ils permettent au programmeur de fixer la durée des notes produites.

Nous utilisons à nouveau à cet effet l'interruption 1Ch du temporisateur qui est appelée 18,2 fois par seconde par l'interruption 8h du temporisateur. Lorsque la routine de génération de sons est appelée, elle attend donc que lui soient communiquées aussi bien la note à produire que la durée de cette note. Cette durée est exprimée en 18èmes de seconde, la valeur 18 correspondant par conséquent à une seconde, la valeur 9 à une demie seconde. Cette valeur est stockée dans une variable.

Immédiatement avant que la sortie sonore ne soit activée, la routine de l'interruption 1Ch est détournée sur une routine utilisateur. Elle sera désormais appelée 18,2 fois par seconde et elle n'aura rien d'autre à faire que de décrémenter la durée de note transmise (qui figure maintenant dans une variable) chaque fois qu'elle sera appelée. Lorsqu'elle atteint la valeur 0, la durée de la note est écoulée et le son doit être désactivé.

C'est ce qu'elle indique alors à la routine Sound proprement dite en fixant une variable déterminée. La routine Sound s'en aperçoit immédiatement car depuis le détournement de l'interruption du temporisateur elle se trouvait dans une boucle d'attente permanente dont le seul objet était justement de tester le contenu de cette variable. Dès qu'elle détecte ainsi la fin de la durée de la note, elle arrête la sortie sonore et replace l'interruption du temporisateur sur son ancienne routine.

Un autre aspect très pratique de la routine Sound réside dans le fait qu'elle n'attend pas la fréquence de la note à sortir mais simplement le numéro de cette note. Ce numéro se réfère à une table dans laquelle Sound a stocké les fréquences des notes des octaves 3 à 5. La valeur 0 correspond ici au Do de la troisième octave, 1 à Do dièse, 2 à Ré, 3 à Ré dièse, 4 à Mi, 5 à Fa, etc...

Les deux programmes jouent, en guise de démonstration, la gamme chromatique des octaves 3 à 5, chaque note retentissant pendant une demie seconde. Cependant le programme de démonstration et la routine Sound sont placés dans le même fichier en assembleur alors qu'ils sont séparés en C. Le programme C contient simplement l'appel de la fonction Sound qui doit être reliée au programme C, en tant que programme assembleur indépendant, lors du linkage.

Voici tout d'abord le programme en C pour appeler la fonction Sound puis le listing du programmeur en Assembleur qui contient la fonction Sound pour la version C.

Listing : SOUNDC.C

```
/*****************************************************************/
/*                          S O U N D C                          */
/*---------------------------------------------------------------*/
/*   Fonction       : Sortie de la gamme chromatique des octaves 3 */
/*                    à 5 à l'aide d'une fonction assembleur     */
/*---------------------------------------------------------------*/
/*   Auteur         : MICHAEL TISCHER                            */
/*   développé le   : 15/08/1987                                 */
/*   dernière modif.: 29/04/1989                                 */
/*---------------------------------------------------------------*/
/*   (MICROSOFT C)                                               */
/*   Création       : CL /AS SOUNDC.C SOUNDA                     */
/*   Appel          : SOUNDc                                     */
/*---------------------------------------------------------------*/
/*   (BORLAND TURBO C)                                           */
/*   Création       : Avec fichier Project de la teneur suivante : */
/*                    soundc                                     */
/*                    soundca.obj                                */
/*****************************************************************/

/*-- Déclaration des fonctions du module assembleur ------------*/
extern void Sound();            /* Pour intégrer la routine assembleur */

/*****************************************************************/
/**                      PROGRAMME PRINCIPAL                    **/
/*****************************************************************/

void main()
{
 int Son;

 printf("\nSOUND (c) 1987 by Michael Tischer\n\n");
 printf("Si le haut-parleur de votre PC fonctionne correctement, vous");
 printf(" allez entendre\nmaintenant la gamme entre les octaves 3 et ");
 printf("5, chaque note étant jouée\npendant une demie seconde.\n\n");
 for (Son = 0; Son < 35; Sound(Son++, 9))    /* Jouer chaque note une */
   ;                                         /* demie seconde        */

 printf("Fin\n");
}
```

Listing : SOUNDCA.ASM

```
;*****************************************************************;
;*                          S O U N D C A                        *;
;*-------------------------------------------------------------- *;
;*   Fonction       : fournit une fonction à intégrer en C pour  *;
;*                    jouer les notes des octaves 3, 4 et 5.     *;
;*                                                               *;
;*-------------------------------------------------------------- *;
;*   Auteur         : MICHAEL TISCHER                            *;
;*   développé le   : 15.08.1987                                 *;
;*   Dernière modif.: 16.02.1991                                 *;
;*-------------------------------------------------------------- *;
;*   Assemblage     : MASM -mx SOUNDCA;                          *;
;*                    ... puis lier à SOUNDC                     *;
;*****************************************************************;

IGROUP group _text              ;Regroupement des segments de programme
DGROUP group const,_bss,  _data    ;Regroupement des segments de données
       assume CS:IGROUP, DS:DGROUP, ES:DGROUP, SS:DGROUP

       public _Sound                        ;Fonction rendue accessible
                                                  ;aux autres programmes

CONST  segment word public 'CONST'  ;Ce segment reçoit toutes constantes
CONST  ends                               ;qui peuvent seulement être lues

_BSS   segment word public 'BSS' ;Ce segment reçoit toutes les variables
_BSS   ends                                    ;statiques non initialisées

_DATA  segment word public 'DATA'      ;Toutes les variables globales et
                                      ;statiques initialisées sont logées
                                                  ;dans ce segment

time_anc  dw (?),(?)        ;Adresse ancienne interruption temporisateur
compteur_s db (?)                      ;Durée restante d'une note en
                                                  ;1/18èmes de seconde

fin_s     db (?)                    ;Indique si note a déjà été jouée
notes     dw 9121,8609,8126,7670    ;Valeurs de notes pour l'octave 3
          dw 7239,6833,6449,6087
          dw 5746,5423,5119,4831
          dw 4560,4304,4063,3834    ;Valeurs de notes pour l'octave 4
          dw 3619,3416,3224,3043
          dw 2873,2711,2559,2415
          dw 2280,2152,2031,1917    ;Valeurs de notes pour l'octave 5
          dw 1809,1715,1612,1521
          dw 1436,1355,1292,1207

_DATA  ends
```

```
;-- Programme -----------------------------------------------------

_TEXT  segment byte public 'CODE'                ;Le segment de programme

;-- SOUND: Jouer une note -----------------------------------------
;-- Appel en C  : Sound((int) Note, (int) Durée);
;-- Sortie      : Aucune
;-- Infos       : La note est le numéro de la note par rapport au Do
;--               de la troisième octave
;--               Durée est durée de note en 1/18èmes de seconde

_Sound    proc near

          push bp                                    ;Ranger BP sur la pile
          mov  bp,sp                                 ;Transférer SP dans BP

        ;-- Détourner interruption temporisateur sur routine utilisateur
          mov  word ptr cs:setds+1,ds ;Sauver DS pour interrupt. tempor.
          mov  ax,351ch     ;Lire adresse de l'interruption temporisateur
          int  21h                              ;Appeler interruption du DOS
          mov  time_anc,bx  ;Adresse d'offset de l'ancienne interruption
          mov  time_anc+2,es                    ;et ranger adresse de segment
          mov  word ptr cs:stjump+1,bx ;Pour nouvelle interrupt. tempor.
          mov  word ptr cs:stjump+3,es                    ;Sauvegarder
          mov  bx,ds                                 ;Ranger DS dans BX
          push cs                                   ;Amener pile sur pile
          pop  ds                                    ;et retirer pour DS
          mov  dx,offset sound_ti;Adresse offset nouvelle routine tempo.
          mov  ax,251ch              ;Fixer nouvelle routine temporisateur
          int  21h                              ;Appeler interruption du DOS
          mov  ds,bx                                    ;Restaurer DS

          mov  al,182                        ;Préparer génération d'un son
          out  43h,al           ;Envoyer valeur au Timer Command Register

          mov  bx,[bp+4]                                      ;Lire note
          xor  bh,bh                      ;BH pour adressage table notes = 0
          shl  bx,1                     ;Doubler n° note (car table de mots)
          mov  ax,[notes+bx]                            ;Lire valeur de note
          out  42h,al           ;Octet faible dans Timer Counter Register
          mov  al,ah                      ;Transférer octet fort dans AL
          out  42h,al               ;et aussi dans Timer Counter Register
          in   al,61h               ;Lire bit de contrôle du haut-parleur
          or   al,11b           ;Deux bits inférieurs activent haut-parleur
          mov  fin_s,1                  ;Note doit encore être jouée
          mov  dl,[bp+6]                                ;Lire durée de note
          mov  compteur_s,dl                              ;et sauvegarder
```

```
;         out  61h,al                              ;Activer haut par eur
;
;play:    cmp  fin_s,0                                  ;Note terminée ?
;         jne  play                            ;NON --> Attendre encore
;
;         in   al,61h              ;Lire bit de contrôle du haut-parleur
;         and  al,11111100b                ;Annuler deux bits inférieurs
;         out  61h,al                          ;Désactiver haut-parleur
;
;         ;-- Activer à nouveau ancienne interruption temporisateur ----
;         mov cx,ds                                          ;Ranger DS
;         mov ax,251ch      ;N° fonct. pour Fixer vecteur d'interruption
;         lds dx,dword ptr time_anc ;Charger ancienne adresse dans DS:DX
;         int 21h                              ;Appeler interruption du DOS
;         mov ds,cx                                        ;Ramener DS
;
;         mov sp,bp                          ;Restaurer pointeur de pile
;         pop bp                                    ;Retirer BP de la pile
;         ret                                ;Retour au programme d'appel
;
;_Sound   endp
;
;;-- Nouvelle interruption temporisateur ----------------------------------
;
;sound_ti  proc far                          ;Ist appelée 18 fois par seconde
;
;         push ax                                  ;Ranger AX et DS sur la pile
;         push ds
;setds:   mov  ax,0000h          ;C'est ici que doit être inscrit le DS C
;         mov  ds,ax
;         dec  compteur_s                                ;Décrémenter compteur
;         jne  fin_st                                  ;Si pas encore 0, alors fin
;         mov  fin_s,0                               ;Signaler durée de note écoulée
;fin_st:  pop  ds                                        ;Fixer à nouveau ancien DS
;         pop  ax                                      ;Retirer aussi AX de la pile
;
;stjump:  db   0EAh,0,0,0,0 ;FAR JUMP vers l'ancienne interruption tempor
;
;sound_ti  endp
;
;;-- Fin ------------------------------------------------------------------
;
;_text    ends                                ;Fin du segment de programme
;         end                                 ;Fin du source assembleur
;
```

Voici le programme complet en assembleur :

Listing : SOUNDA.ASM

```
;*****************************************************************;
;*                            S O U N D                          *;
;*----------------------------------------------------------------*;
;*   Fonction       : sort la gamme chromatique de l'octave 3 à   *;
;*                    l'octave 5 à l'aide d'une routine universelle *;
;*                    de génération de sons.                      *;
;*----------------------------------------------------------------*;
;*   Auteur         : MICHAEL TISCHER                             *;
;*   développé le   : 06.08.1987                                  *;
;*   Dernière modif.: 16.02.1992                                  *;
;*----------------------------------------------------------------*;
;*   Assemblage     : MASM SOUNDA;                                *;
;*                    LINK SOUNDA;                                *;
;*                    EXE2BIN SOUNDA SOUNDA.COM       ou          *;
;*                    TASM SOUNDA                                 *;
;*                    TLINK /TC SOUNDA                            *;
;*----------------------------------------------------------------*;
;*   Appel          : SOUND                                       *;
;*****************************************************************;
;
code     segment para 'CODE'              ;Définition du segment de CODE
;
         org 100h                         ;Il commence à l'adresse 100(h)
                                          ;directement à la suite du PSP
;
         assume cs:code, ds:code, es:code, ss:code
;
;-- Programme ---------------------------------------------------------
;
sound    proc near
;
         ;-- Sortir message ------------------------------------------
;
         mov  ah,9                ;Numéro de fonction pour Sortir chaîne
         mov  dx,offset initm              ;Adresse d'offset du message
         int  21h                          ;Appeler interruption du DOS
;
         ;-- Jouer gammes chromatiques -------------------------------
         xor  bl,bl           ;Commencer par le Do de la troisième octave
         mov  dl,9                ;Jouer une demie seconde chaque fois
nextune: call play_tune                                  ;Jouer note
         inc  bl                                        ;Note suivante
         cmp  bl,36           ;Toutes notes trois octaves déjà jouées ?
         jne  nextune                              ;NON --> Jouer note suivante
;
         ;-- Sortir message de fin -----------------------------------
;
         mov  ah,9                ;Numéro de fonction pour Sortir chaîne
         mov  dx,offset endmes             ;Adresse d'offset du message
         int  21h                          ;Appeler interruption du DOS
;
         mov  ax,4C00h            ;Terminer programme par appel d'une
         int  21h                          ;fonction DOS en renvoyant le
                                                        ;code d'erreur 0
;
sound    endp
;
;-- Données du programme principal -----------------------------------
;
initm    db 13,10,"SOUND (c) 1987 by Michael Tischer",13,10,13,10
         db "Si le haut-parleur de votre PC fonctionne correctement, "
         db "vous allez entendre",13,10,"maintenant la gamme chro"
         db "matique des octaves 3 à 5.",13,10,"$"
;
endmes   db 13,10,"Fin",13,10,"$"
;
;;-- PLAY_TUNE: Jouer une note ---------------------------------------
;;-- Entrée   : BL = Numéro de note par rapport au Do troisième octave
;;--            DL = Durée de note en 1/18èmes de seconde
;;-- Sortie   : Aucune
;;-- Registres: AX, CX, ES et FLAGS sont modifiés
;;-- Infos    : Le contrôle n'est rendu à la routine d'appel qu'une
;;--            fois la durée de note écoulée
;
play_tune proc near
         push dx                                  ;Ranger DX et BX sur la pile
         push bx
;
        ;-- Détourner interruption temporisateur sur routine utilisateur
         push dx                                  ;Ranger DX et BX sur la pile
         push bx
         mov  ax,351ch    ;Lire adresse de l'interruption temporisateur
         int  21h                                 ;Appeler interruption du DOS
         mov  time_anc,bx  ;Adresse d'offset de l'ancienne interruption
         mov  time_anc+2,es                  ;et ranger adresse de segment
;
         mov  dx,offset sound_ti;Adresse offset nouvelle routine tempo.
         mov  ax,251ch             ;Fixer nouvelle routine temporisateur
         int  21h                                 ;Appeler interruption du DOS
         pop  bx                                  ;Retirer BX et DX de la pile
         pop  dx
;
         mov  al,182                        ;Préparer génération d'un son
         out  43h,al             ;Envoyer valeur au Timer Command Register
         xor  bh,bh                    ;BH pour adressage table notes = 0
         shl  bx,1                  ;Doubler n° note (car table de mots)
         mov  ax,[notes+bx]                          ;Lire valeur de note
         out  42h,al             ;Octet faible dans Timer Counter Register
         mov  al,ah                        ;Transférer octet fort dans AL
         out  42h,al                ;et aussi dans Timer Counter Register
         in   al,61h                ;Lire bit de contrôle du haut-parleur
         or   al,11b          ;Deux bits inférieurs activent haut-parleur
         mov  fin_s,1                        ;Note doit encore être jouée
         mov  compteur_s,dl                        ;Sauver durée de note
         out  61h,al                               ;Activer haut-parleur
;
play:    cmp  fin_s,0                                  ;Note terminée ?
         jne  play                            ;NON --> Attendre encore
;
         in   al,61h              ;Lire bit de contrôle du haut-parleur
         and  al,11111100b                ;Annuler deux bits inférieurs
         out  61h,al                          ;Désactiver haut-parleur
;
```

```
;-- Activer à nouveau ancienne interruption temporisateur ----

        mov cx,ds                                       ;Ranger DS
        mov ax,251ch    ;N° fonct. pour Fixer vecteur d'interruption
        lds dx,dword ptr time_anc ;Charger ancienne adresse dans DS:DX
        int 21h                                 ;Appeler interruption du DOS
        mov ds,cx                                       ;Ramener DS

        pop  bx                                 ;Retirer BX et DX de la pile
        pop  dx
        ret                                     ;Retour au programme d'appel

play_tune endp

;-- Nouvelle interruption temporisateur -----------------------------

sound_ti  proc far                      ;Est appelée 18 fois par seconde

        dec  cs:compteur_s                      ;Décrémenter compteur
        jne  fin_st                     ;Si pas encore 0, alors fin
        mov  cs:fin_s,0                 ;Signaler durée de note écoulée
fin_st:  jmp  dword ptr cs:[time_anc] ;Vers ancienne interruption tempo

sound_ti  endp

;-- Voici les variables dont ont besoin les quatre routines ---------

time_anc  dw (?),(?)        ;Adresse de l'ancienne interruption tempor.
compteur_s db (?)                       ;Durée restante d'une note en
                                                ;1/18èmes de seconde
fin_s     db (?)                        ;Indique si note a déjà été jouée
notes     dw 9121,8609,8126,7670        ;Valeurs de notes pour l'octave 3
          dw 7239,6833,6449,6087
          dw 5746,5423,5119,4831
          dw 4560,4304,4063,3834        ;Valeurs de notes pour l'octave 4
          dw 3619,3416,3224,3043
          dw 2873,2711,2559,2415
          dw 2280,2152,2031,1917        ;Valeurs de notes pour l'octave 5
          dw 1809,1715,1612,1521
          dw 1436,1355,1292,1207

;-- Fin -------------------------------------------------------------

code      ends                                  ;Fin du segment de CODE
          end  sound                          ;Fin du programme assembleur
```

14. Configuration et type de processeur

Nombreuses sont les situations où la configuration de l'ordinateur joue un rôle important. Lorsqu'un programme doit connaître le nombre d'interfaces série, la taille de la mémoire RAM ou contrôler l'existence d'un co-processeur numérique, les informations sur la configuration doivent toujours être demandées dans ces cas.

Ce chapitre montre comment obtenir les innombrables données de configuration à l'aide de deux interruptions spécifiques de la ROM BIOS et comment les placer sur la piste du processeur et de son co-processeur à l'aide de programmes écrits en un langage approprié.

14.1. Définir la configuration à l'aide du BIOS

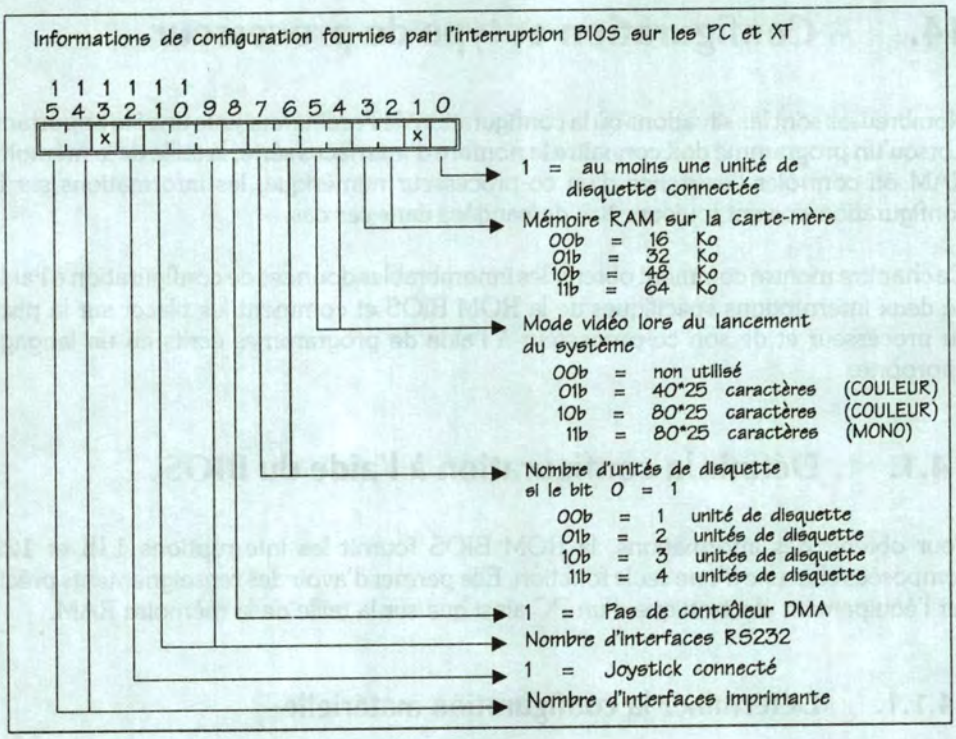
Pour obtenir des informations, la ROM BIOS fournit les interruptions 11h et 12h composées chacune d'une seule fonction. Elle permet d'avoir des renseignements précis sur l'équipement électronique d'un PC ainsi que sur la taille de la mémoire RAM.

14.1.1. Déterminer la configuration matérielle

L'appel de l'interruption BIOS 11h fournit des informations sur la configuration matérielle d'un ordinateur, le nombre d'interfaces série et parallèle, les lecteurs de disquettes et le contrôleur DMA.

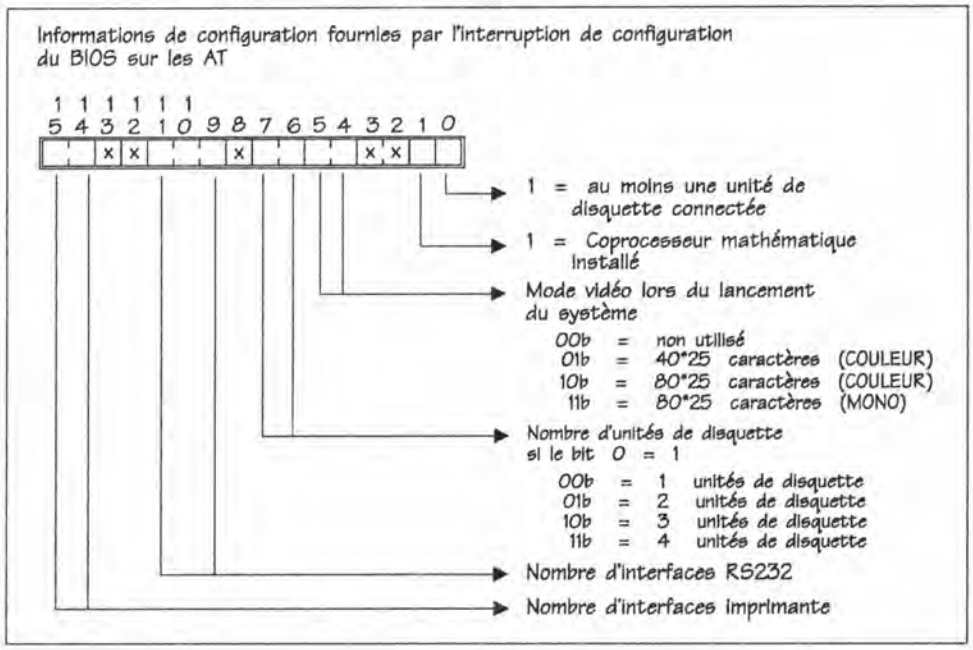
Contrairement aux autres interruptions BIOS, le contenu du registre du processeur ne joue aucun rôle lors de l'appel de cette interruption car ni un numéro de fonction ni un autre argument ne sont attendus. Cette interruption n'étant composée que d'une seule fonction, les informations souhaitées sont retournées dans le registre AX.

Dans la figure suivante, le contenu du registre AX représente un champ de bits après l'appel de la fonction. Les divers champs donnent des informations sur les différents éléments de l'électronique. Les informations ne sont pas très abondantes vu l'ancienneté de cette interruption. Elle existait déjà sur les premiers PC et leur BIOS si bien qu'elle ne reproduit que la configuration matérielle présente dans les PC de cette époque.



Cette définition des différents bits s'applique aussi bien au PC qu'au XT mais elle diffère légèrement de la structure du mot de configuration fourni par l'AT. Pour pouvoir interpréter correctement le contenu du registre AX, il est donc indispensable de savoir à quelle machine on a affaire.

Cette interruption communique néanmoins à l'utilisateur de nombreuses informations importantes telles que le type du processeur, du clavier, la disponibilité d'une souris ou d'une mémoire EMS. Dans les divers chapitres de ce livre, n'hésitez donc pas à vous en servir pour connaître par exemple si une souris est connectée ou le type de carte vidéo installée.



Si vous voulez connaître le mode vidéo actuel, ce n'est cependant pas cette fonction qu'il conviendra d'utiliser puisqu'elle indique simplement quel mode vidéo était activé lors du lancement du système. Vous serez mieux informé par la fonction 0Fh de l'interruption 10h qui renvoie le numéro du mode vidéo actuel. La section suivante vous présentera un programme d'exemple d'utilisation de cette interruption.

14.1.2. Déterminer la taille de la mémoire RAM à l'aide du BIOS

L'interruption 11h communique à l'utilisateur la taille de la mémoire RAM sur la carte mère, alors que l'interruption 12h indique la taille de la mémoire RAM dans le système complet. Pour pouvoir fournir cette information, la taille de la RAM sur la carte mère est comptabilisée et enregistrée, ainsi que celle de la RAM figurant sur d'éventuelles cartes d'extension de la mémoire. Sur le PC et le XT, ces informations sont tirées de la position des différents switchs DIP sur les différentes cartes de mémoire, alors que sur l'AT elles sont lues dans l'une des 64 cellules de mémoire de l'horloge CMOS. On ne peut toutefois, en procédant ainsi, déterminer que la taille de la mémoire RAM en dessous de la limite de 1 Mo. Cela suffit cependant largement pour le PC et le XT car la zone d'adressage de leur processeur (le 8088) est limitée à 1 Mo. Ces modèles ne peuvent donc disposer d'une autre mémoire RAM en plus de la RAM ainsi comptabilisée. Il en va bien sûr autrement pour l'AT, dont le processeur (80286) peut gérer jusqu'à 16 Mo de mémoire. Avec un ordinateur de ce type, une mémoire supplémentaire, dont

l'interruption 12h ne tient pas compte, peut donc être installée en plus de la RAM "normale" qui s'étend jusqu'à la limite de 1 Mo.

La taille de la mémoire est renvoyée dans le registre AX, après appel de cette interruption, exprimée en Ko (c'est-à-dire en unités de 1024 octets et non 1000 !).

Un résultat de 256 signifiera par exemple que votre PC comporte 256 Ko de mémoire RAM.

14.1.3. Exemples de programmes

En conclusion de ce chapitre, nous allons comme d'habitude vous présenter un exemple pratique d'utilisation de l'interruption que nous venons de décrire. Voici donc trois listings de programmes que nous avions déjà annoncés à la section précédente. La structure de base de ces trois programmes, qui sont écrits en BASIC, Pascal et C, reste la même.

Ils testent tout d'abord l'octet d'identification du modèle, dans la cellule de mémoire F000:FFFE, pour déterminer si on a affaire à un PC, XT ou à un AT. La désignation du modèle est sortie sur l'écran. C'est également sur cette identification du modèle que repose la sortie du type de processeur, en considérant systématiquement que l'AT dispose d'un processeur 80286 et que tous les autres PC disposent d'un processeur 8088. L'étape suivante consiste alors à déterminer puis à afficher la taille de la mémoire RAM sur la carte mère, à l'aide de l'interruption 12h qui a été décrite dans cette section. Comme nous l'avons toutefois indiqué, l'AT peut disposer de mémoire RAM supplémentaire au-delà de la limite de 1 Mo. C'est pourquoi, si l'ordinateur est un AT, la taille de cette mémoire RAM doit encore être déterminée. Le programme utilise à cet effet la fonction 08h de l'interruption 15h que nous décrirons plus précisément par la suite. Pour le moment, il nous suffira de savoir que la taille de la mémoire RAM au-delà de la limite de 1 Mo est renvoyée dans le registre AX, exprimée en Ko.

Ce n'est qu'après que cette information ait été affichée que l'interruption 11h sera appelée pour déterminer la configuration de la machine. La dernière tâche du programme consistera alors à décomposer le mot de configuration reçu en ses différents bits pour décoder les informations auxquelles ils correspondent afin d'afficher ces informations sur l'écran.

Pour que le programme ne soit pas trop long, nous nous sommes limités aux bits qui sont identiques aussi bien dans le mot de configuration du PC et du XT d'une part que dans celui de l'AT d'autre part. Nous avons ainsi négligé l'information indiquant, sur un AT, si l'ordinateur dispose ou non d'un coprocesseur mathématique.

Vous pouvez donc considérer ce programme comme un point de départ et le compléter de telle façon qu'il fournisse, suivant le modèle d'ordinateur, toutes les informations que recèle le mot de configuration.

Si vous vous posez encore certaines questions sur le fonctionnement des différents programmes, les commentaires très complets des listings vous renseigneront certainement.

Listing : CONFIGB.BAS

```
'*****************************************************************
'*                        C O N F I G                            *
'*---------------------------------------------------------------*
'*   Fonction       : affiche sur l'écran la configuration du PC *
'*---------------------------------------------------------------*
'*   Auteur         : MICHAEL TISCHER                            *
'*   Développé le   : 07/07/1987                                 *
'*   Dernière modif.: 18/10/1992                                 *
'*****************************************************************

DECLARE SUB PrintConfig ()
DECLARE FUNCTION GetWord& (Register AS INTEGER)

'$INCLUDE: 'QB.BI'              ' Inclure la déclaration des registres

CONST TRUE = -1
CONST FALSE = NOT TRUE

CALL PrintConfig                                ' Affiche la configuration
END

'*****************************************************************
'* GetWord: Récupère et valide les valeurs récupérées dans les   *
'*          pour la division entière et le modulo                *
'* Entrée : Valeur à convertir                                   *
'* Sortie : Valeur convertie                                     *
'*****************************************************************
'
'
FUNCTION GetWord& (Register AS INTEGER)

IF Register <= 0 THEN                           ' Bit 16 posé ? (signe)
  GetWord = 65536 + Register                      ' Complément positif
ELSE
  GetWord = Register
END IF
END FUNCTION

'*****************************************************************
'* PrintConfig: Affichage de la configuration d'un PC            *
'* Entrée : Aucune                                               *
'* Sortie : Aucune                                               *
'* Infos  : la configuration est sortie en tenant compte du type de *
'*          PC                                                   *
'*****************************************************************
'
SUB PrintConfig

DIM AT AS INTEGER                               ' Ce PC est-il un AT? '
DIM Word AS LONG
DIM Register AS RegType       ' Registres processeur pour interruption '

CLS                                                    ' Vider l'écran '
DEF SEG = &HF000
IF PEEK(&HFFFE) = &HFC THEN
  AT = TRUE                                    ' Tester si AT ou bien '
ELSE
  AT = FALSE                                             ' PC ou  XT '
END IF

PRINT "CONFIGB  -  (c) 1987, 1992 by Michael Tischer"
PRINT
PRINT "Configuration de votre PC"
PRINT "-------------------------------------------------------"
PRINT "Type de PC              : ";
SELECT CASE PEEK(&HFFFE)
  CASE &HFF                                            ' &FF, c'est un PC '
    PRINT "PC"
  CASE &HFE                                            ' &FE, c'est un XT '
    PRINT "XT"
  CASE &HFC                                            ' &FC, c'est un AT '
    PRINT "AT ou supérieur"
END SELECT
CALL Interrupt(&H12, Register, Register)  ' Taille RAM donnée par BIOS '
PRINT "Mémoire RAM             :"; Register.ax; "Ko"
IF AT THEN                                          ' Ce PC est-il un AT ? '
  Register.ax = &H8800       ' fonction pour lire taille mémoire suppl. '
  CALL Interrupt(&H15, Register, Register) 'Interrup. cassette du BIOS '
  PRINT "RAM supplémentaire      :"; Regs.ax; "Ko au dessus 1 Mo"
END IF
CALL Interrupt(&H11, Register, Register) ' Interruption de config BIOS '
PRINT "Mode vidéo initial      : ";
SELECT CASE (Register.ax MOD 256) AND 48    ' Déterminer le mode vidéo '
  CASE 0
    PRINT "Indéfini"
  CASE 16
    PRINT "40*25 caractères carte couleur"
  CASE 32
    PRINT "80*25 caractères carte couleur"
  CASE 48
    PRINT "80*25 caractères carte monochrome"
END SELECT
Word = GetWord(Register.ax)
PRINT "Lecteurs de disquette   :"; (((Word MOD 256) \ 64) AND 3) + 1
PRINT "Interfaces série        :"; ((Word \ 256) \ 2) AND 3
PRINT "Interfaces parallèle    :"; (Word \ 256) \ 64
END SUB
```

Listing : CONFIGP.PAS

```
{*****************************************************************}
{*                        C O N F I G                            *}
{*---------------------------------------------------------------*}
{*   Fonction       : affiche sur l'écran la configuration du PC *}
{*---------------------------------------------------------------*}
{*   Auteur         : MICHAEL TISCHER                            *}
{*   Développé le   : 07/07/1987                                 *}
{*   Dernière modif.: 18/10/1992                                 *}
{*****************************************************************}

program CONFIG;

Uses Crt, Dos;                          { Intégrer les unités CRT et DOS }

{*****************************************************************}
{* PrintConfig: Affichage de la configuration d'un PC            *}
{* Entrée : Aucune                                               *}
{* Sortie : Aucune                                               *}
{* Infos  : la configuration est sortie en tenant compte du type de  *}
{*          PC                                                   *}
{*****************************************************************}

procedure PrintConfig;

var AT       : boolean;                          { Ce PC est-il un AT? }
    Regs : Registers;          { Registres processeur pour interruption }

begin
  ClrScr;                                                { Vider l'écran }
  if Mem[$F000:$FFFE] = $FC then AT := true      { Tester si AT ou bien }
                            else AT := false;                { PC ou  XT }
  writeln('CONFIGP  -  (c) 1987, 1992 by Michael Tischer');
  writeln;
  writeln('Configuration de votre PC');
  writeln('-------------------------------------------------------');
  write('Type de PC              : ');
  case Mem[$F000:$FFFE] of                { Lire à nouveau le type de PC }
    $FF : writeln('PC');                                  { $FF, c'est un PC }
```

```
  $FE : writeln('XT');                                { $FE, c'est un XT }
else writeln('AT ou supérieur')                       { $FC, c'est un AT }
end;

Intr($12, Regs);      { Lire taille mémoire RAM avec interruption BIOS }
writeln('Mémoire RAM              : ',Regs.ax,' Ko');
if AT then                                  { Ce PC est-il un AT ? }
  begin                                                    { OUI }
    Regs.ah := $88;          { fonction pour lire taille mémoire suppl. }
    Intr($15, Regs );          { Appeler interruption cassette du BIOS }

    writeln('RAM supplémentaire       : ',Regs.ax,' Ko au dessus 1 Mo');
  end;
Intr($11, Regs);      { Appeler interruption de configuration du BIOS }
write('Mode vidéo initial         : ');
case Regs.al and 48 of                      { Déterminer le mode vidéo }
  0 : writeln('Indéfini');
 16 : writeln('40*25 caractères carte couleur');
 32 : writeln('80*25 caractères carte couleur');
 48 : writeln('80*25 caractères carte monochrome');
end;
writeln('Lecteurs de disquette    : ', succ(Regs.al shr 6 and 3));
writeln('Interfaces série         : ', Regs.ah shr 1 and 3);
writeln('Interfaces parallèle     : ', Regs.ah shr 6)
end;

{*********************************************************************}
{*                        PROGRAMME PRINCIPAL                        *}
{*********************************************************************}

begin
 PrintConfig;                                   { Sortir configuration }
end.
```

Listing : CONFIGC.C

```
/*********************************************************************/
/*                          C O N F I G C                            */
/*-------------------------------------------------------------------*/
/*    Fonction        : Affiche la configuration du PC sur l'écran   */
/*                                                                   */
/*-------------------------------------------------------------------*/
/*    Auteur          : MICHAEL TISCHER                              */
/*    Développé le    : 13.08.1987                                   */
/*    Dernière modif. : 19.02.1992                                   */
/*-------------------------------------------------------------------*/
/*    Modèle de mémoire : SMALL                                      */
/*********************************************************************/

/*-- Intégrer les fichiers Include ----------------------------------*/

#include <dos.h>                        /* Intégrer les fichiers Header */
#include <stdio.h>

/*-- Typedefs -------------------------------------------------------*/

typedef unsigned char BYTE;     /* Voilà comment se bricoler un OCTET */

/*-- Macros ---------------------------------------------------------*/

#ifdef MK_FP
  #undef MK_FP
#endif

#ifdef peekb
  #undef peekb
#endif

#define MK_FP(seg, ofs) ((void far *) ((unsigned long) (seg)<<16|(ofs)))
#define peekb(seg, ofs) *((BYTE far *) MK_FP(seg, ofs))

/*-- Constantes -----------------------------------------------------*/

#define TRUE  ( 0 == 0 )        /* Constantes pour faciliter la lecture */
#define FASLE ( 0 == 1 )        /* du texte du programme                */

/*********************************************************************/
/* CLS: Effacer l'écran et placer le curseur dans le coin supérieur  */
/*      gauche de l'écran                                            */
/* Entrée : Aucune                                                   */
/* Sortie : Aucune                                                   */
/*********************************************************************/

void Cls( void )
{
 union REGS Register;           /* Registres pour appeler l'interruption*/

 Register.h.ah = 6;                /* Numéro de fonction pour Scroll-UP */
 Register.h.al = 0;                                 /* 0 pour effacer */
 Register.h.bh = 7;                   /* Ecriture claire sur fond noir */
 Register.x.cx = 0;                /* Coin supérieur gauche de l'écran */
 Register.h.dh = 24;                  /* Coordonnées du coin inférieur */
 Register.h.dl = 79;                              /* droit de l'écran  */
 int86(0x10, &Register, &Register);      /* interruption vidéo du BIOS */

 Register.h.ah = 2;          /* Fonction pour fixer position du curseur */
 Register.h.bh = 0;                                    /* Page écran 0 */
 Register.x.dx = 0;    /* Coordonnées coin supérieur gauche de l'écran */
 int86(0x10, &Register, &Register);      /* interruption vidéo du BIOS */
}

/*********************************************************************/
/* PRINTCONFIG: Affichage de la configuration d'un PC                */
/* Entrée : Aucune                                                   */
/* Sortie : Aucune                                                   */
/* Infos    : La configuration est affichée en fonction du type de PC */
/*                                                                   */
/*********************************************************************/

void PrintConfig( void )
{
 union REGS Register;          /* Registres pour appeler l'interruption */
 BYTE AT;                                          /* AT ou supérieur? */

 Cls();                                            /* Effacer l'écran */
 AT = (peekb(0xF000, 0xFFFE) == 0xFC);
 printf("CONFIGC  -  (c) 1987, 92 by Michael Tischer\n\n");
 printf("Configuration de votre PC\n");
 printf("-------------------------------------------------------\n");
 printf("Type de PC                : ");

 switch( peekb(0xF000, 0xFFFE) )                        /* Type de PC */
 {
   case 0xFF : printf("PC\n");               /* 0xFF c'est un PC normal */
               break;
   case 0xFE : printf("XT\n");                     /* 0xFE c'est un XT */
               break;
   default   : printf("AT ou supérieur\n");        /* 0xFC c'est un AT */
               break;
 }
 printf("Mémoire RAM               : ");
 int86(0x12, &Register, &Register);        /* Lire la taille de la RAM */
 printf("%d Ko\n",Register.x.ax);                       /* et l'afficher */
 if ( AT )                                  /* Ce PC est-il un AT? */
 {                                                          /* OUI */
  Register.h.ah = 0x88;    /* n° de fonction pour RAM supplémentaire */
  int86(0x15, &Register, &Register);     /* Lire la taille de la RAM */
  printf("RAM supplémentaire        : %u Ko au-dessus de 1Mo\n",
                          Register.x.ax);
 }
 int86(0x11, &Register, &Register);          /* Configuration (BIOS) */
 printf("Mode vidéo initial        : ");
 switch(Register.x.ax & 48)
 {
   case  0 : printf("Indéfini\n");
             break;
   case 16 : printf("40*25 caractères couleur\n");
             break;
   case 32 : printf("80*25 caractères couleur\n");
             break;
   case 48 : printf("80*25 caractères mono\n");
             break;
 }
 printf("Lecteurs de disquette     : %d\n", (Register.x.ax >> 6 & 3) +
1);
 printf("Interfaces série          : %d\n", Register.x.ax >> 9 & 0x03);
 printf("Interfaces parallèle      : %d\n\n", Register.x.ax >> 14);
}

/*********************************************************************/
/**                        PROGRAMME PRINCIPAL                      **/
/*********************************************************************/

void main()
{
 PrintConfig();                          /* Afficher la configuration */
}
```

14.2. Déterminer le type de processeur et co-processeur

Sur le marché très encombré des programmes utilitaires pour le PC existent aujourd'hui toute une série de programmes permettant d'obtenir des informations sur la configuration ou l'équipement d'un PC. Ces programmes indiquent la taille de la mémoire RAM, le numéro de version DOS installée etc., ainsi que le type de processeur dont est doté le PC.

Cette information peut être très utile par exemple lors du développement de programmes en langage évolué car elle permet d'adapter le code produit à un processeur bien précis. C'est ainsi, par exemple, que Microsoft C et Turbo C permettent de produire un code spécifique pour le 8088, le 80286 ou le i386, ce qui permet d'exploiter au mieux les possibilités du processeur et son jeu d'instructions. Pour des programmes traitant des masses de données importantes, on peut ainsi obtenir une amélioration considérable des performances. Pour tirer parti de cet avantage, il faut compiler le programme séparément pour chacun des trois types de processeur. Il faut aussi développer un programme servant à charger le véritable programme, qui recherchera tout d'abord le type du processeur et exécutera celui des trois programmes qui avait été compilé pour ce processeur.

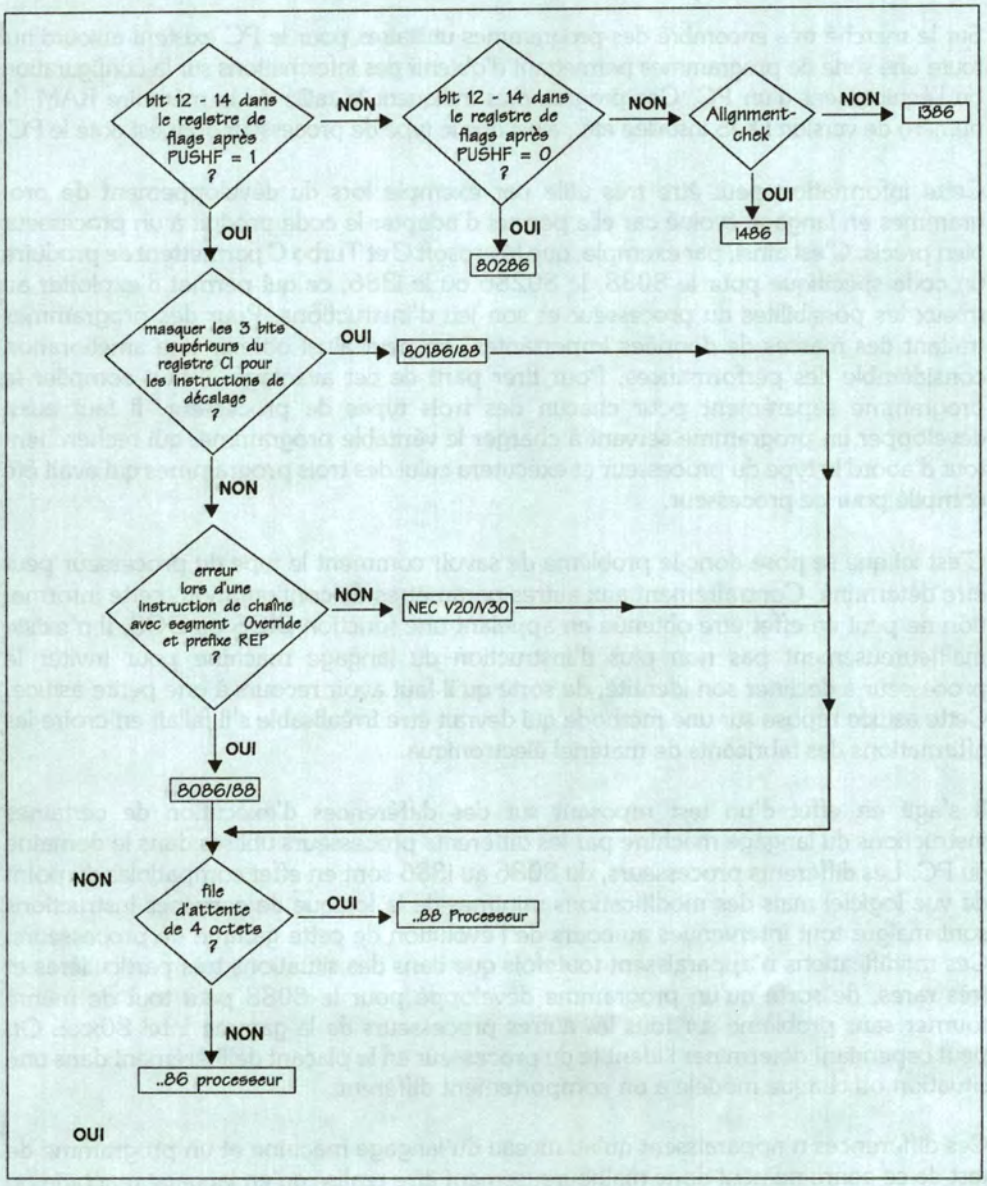
C'est ici que se pose donc le problème de savoir comment le type du processeur peut être déterminé. Contrairement aux autres paramètres de configuration, cette information ne peut en effet être obtenue en appelant une fonction BIOS ou DOS. Il n'existe malheureusement pas non plus d'instruction du langage machine pour inviter le processeur à décliner son identité, de sorte qu'il faut avoir recours à une petite astuce. Cette astuce repose sur une méthode qui devrait être irréalisable s'il fallait en croire les affirmations des fabricants de matériel électronique.

Il s'agit en effet d'un test reposant sur des différences d'exécution de certaines instructions du langage machine par les différents processeurs utilisés dans le domaine du PC. Les différents processeurs, du 8086 au i386 sont en effet compatibles du point de vue logiciel mais des modifications minimes de la logique de certaines instructions sont malgré tout intervenues au cours de l'évolution de cette gamme de processeurs. Ces modifications n'apparaissent toutefois que dans des situations très particulières et très rares, de sorte qu'un programme développé pour le 8088 peut tout de même tourner sans problème sur tous les autres processeurs de la gamme Intel 80xxx. On peut cependant déterminer l'identité du processeur en le plaçant délibérément dans une situation ou chaque modèle a un comportement différent.

Ces différences n'apparaissent qu'au niveau du langage machine et un programme de test de ce genre ne peut donc malheureusement être réalisé qu'en langage machine ou assembleur. A la fin de ce chapitre, vous trouverez deux modules en Assembleur dans lesquels la routine de test est intégrée dans des programmes en Pascal et C. Nous allons maintenant nous pencher sur le principe de fonctionnement de la routine de test et donc sur les différences entre les divers processeurs qui sont à la base de cette routine.

Remarque : Le i486 est l'assemblage sur la même puce d'un i386, i387 et 82385.

14.2.1. Déterminer le type de processeur



Tests pour déterminer le type du processeur d'un PC

Comme le montre l'organigramme, la routine de test se compose de plusieurs tests permettant de distinguer les différents types de processeur. Un test n'est exécuté que si le test précédent a échoué.

Différences d'affectation du registre de flags

Le premier test repose sur les différences d'affectation du registre de flags sur les divers processeurs. Le rôle des bits 0 à 11 de ce registre est en effet identique sur tous les processeurs mais les bits 12 à 15 ne sont utilisés qu'à partir du 80286 (du fait de l'introduction du mode protégé). Les instructions PUSHF (placer le contenu du registre de flags sur la pile) et POPF (retirer le contenu du registre de flags de la pile) ont donc dû en tenir compte. Alors que jusqu'au 80188, ces instructions fixaient toujours sur 1 les bits 12 à 15 du registre de flags, elles procèdent désormais différemment sur le 80286 et le i386. Le premier test de la routine de test s'appuie sur ce fait et place tout d'abord la valeur 0 sur la pile puis ramène dans le registre de flags, avec POPF, la valeur ainsi sauvegardée. Comme il n'existe aucune instruction permettant de tester directement le contenu des bits 12 à 15 du registre de flags, le registre de flags est immédiatement replacé sur la pile à l'aide de l'instruction PUSHF. Il s'agit toutefois simplement de pouvoir amener ensuite cette valeur dans le registre AX, avec l'instruction POP AX. On peut alors aisément tester la valeur des bits 12 à 15. Si les quatre bits sont mis, il ne peut s'agir ni d'un 80286, ni d'un i386 et on passe donc au test suivant.

Si par contre les quatre bits ne sont pas mis, on sait déjà que seuls le 80286 et le i386 sont encore possibles. Or il existe également une différence entre ces deux processeurs dans le traitement de l'instruction POPF, de sorte qu'il est aisé de les distinguer. On répète donc l'ensemble de l'opération, mais en plaçant cette fois la valeur 07000h sur la pile au lieu de la valeur 0. En chargeant ensuite cette valeur dans le registre de flags à l'aide de l'instruction POPF, on fixera donc sur 1 les bits 12 à 14 du registre de flags. Si ensuite, lorsqu'on retirera le registre de flags de la pile, il s'avère que ces bits ne contiennent plus 1, c'est qu'il s'agit d'un 80286 car ce processeur, contrairement au 80386, annule automatiquement ces trois bits. Le test est donc terminé pour ces deux modèles.

Différences entre le i386 et le i486

S'il s'agit maintenant d'un i386, une différenciation entre le i386 et le i486 doit avoir lieu éventuellement à l'aide du registre de flags. Avec le i486, un nouveau flag est en effet intégré dans le registre de flags étendu EFlags. Il porte le nom "Alignment Check" et se trouve à la position 18. Si ce flag est réglé, le processeur contrôle tous les accès en mémoire concernant l'adresse d'offset spécifiée dans ce cas. Si cette dernière n'est pas un multiple de 4, il déclenche une interruption spéciale désignée par Exception.

Si la routine Alignment Check est en service, cette circonstance atteint par exemple l'instruction de langage machine suivante qui lit le contenu de la cellule de mémoire 102. 102 n'est pas un multiple de 4 et de ce fait n'est pas aussi "aligned" que la cellule de mémoire 77 réclamée dans l'instruction qui suit immédiatement.

```
mov al, [102]
shl word ptr [77],3
```

Lors de leur excécution, les deux commandes opèrent une Alignment Exception mettant en oeuvre un Exception Handler sous la forme d'une interruption. Généralement, le système d'exploitation est réclamé dans ce cas pour prendre connaissance du Misalignment. Selon l'information obtenue, le programme concerné peut s'interrompre ou poursuivre normalement son exécution.

Cette procédure permet de vérifier que l'accès à la mémoire s'effectue plus rapidement dans un i486 quand la case à éditer se trouve dans une adresse d'offset divisible par quatre. Expliquons-nous : Pour accéder à la mémoire, le i486 nécessite un temps particulièrement important lorsque cette condition n'est pas remplie. Pour éviter ce désagrément, un système d'exploitation peut activer la routine Alignment Check pour s'enquérir sur les programmes accédant aux données et variables mal alignées lors de leur exécution.

Le système d'exploitation peut par exemple émettre un message d'avertissement destiné au développeur de l'application pour signaler le déclenchement de l'exception appropriée. La routine Alignment Check devient une aide intéressante lorsqu'il s'agit d'optimiser les programmes pour le 80486.

Après avoir pris connaissance de la routine Alignment Check, il devient facile de rabaisser le i386 par rapport au i486 puisque le i386 ne connaît pas ce flag et n'est pas capable de modifier son contenu. En commençant par déterminer et sauvegarder le contenu du registre EFlag, on peut faire pivoter le bit d'alignement. On demande ensuite le registre EFlag pour contrôler si le contenu de ce flag a réellement changé. Si tel est le cas et dans ce cas seulement, on sait qu'il s'agit d'un i486.

Il ne faut cependant pas oublier de régler le flag d'alignement sur son ancienne valeur sans oublier que ce processus comporte quelques dangers. Ils sont surtout liés à la manipulation du registre EFlag qui ne s'effectue qu'à travers la pile. Les deux instructions du langage machine suivantes démontrent ce processus dont le rôle est de charger le registre EFlag dans le registre EAX. A cet effet, son contenu est d'abord poussé sur la pile avant d'atterrir dans le registre AX.

```
pushfd
pop eax
```

Bien que les deux instructions ne soient pas directement visibles, deux accès à la mémoire ont lieu dans ce contexte puisque la pile se trouve effectivement à cet endroit. Si le bit d'alignement est réglé au moment de l'exécution de cette instruction, cela déclenche une exception lorsque le pointeur SP n'indique pas une adresse divisible par quatre. Mais il est tout à fait possible de provoquer cette action si l'instruction de langage machine

```
and esp, 0FFFCh
```

précède les instructions citées plus haut. Elle fixe tout simplement à 0 les deux bits inférieurs en arrondissant le pointeur de pile sur l'adresse suivante divisible par quatre.

Sachant qu'un tel test de processeur s'exécute généralement dans le cadre d'un sous-programme, on doit d'abord prendre connaissance du contenu actuel du pointeur de pile pour pouvoir le restaurer par la suite.

Versions SX

Bien que ce test fonctionne correctement quand il s'agit de faire la différence entre le i386 et le i486, il comporte toutefois un hic. Ce test ne permet pas en effet de différencier les versions SX et DX de ces processeurs et, à notre connaissance, il n'existe pas de test logiciel qui distingue un i386SX d'un i386DX. Le processus est un peu différent dans le cas du i486SX et du i486DX quand il s'agit d'effectuer une distinction de manière indirecte également. La version SX de ce processeur n'est pas équipée du co-processeur numérique i387 qui est intégré dans la version DX. Après avoir découvert un processeur i486, il devient aisé de déterminer la présence des instructions à virgule flottante à l'aide d'un test de co-processeur et de connaître ainsi la version du processeur qui est installée.

Si les instructions à virgule flottante sont soutenues, cela ne signifie nullement qu'un i486DX est également installé. On peut très bien avoir un i486SX étendu à l'aide d'un co-processeur numérique. On ne peut affirmer qu'un i486SX est effectivement installé que lorsqu'on découvre un 80486 ne soutenant pas les opérations numériques du processeur à virgule flottante.

Différence entre le 8086/8088 et le 80186/80188

Si cependant le premier test n'a pas été positif, le test suivant permet de déterminer si on a affaire à un 80188 ou à un 80186. Lors de l'introduction de ces deux modèles, le fonctionnement des instructions de décalage (SHL et SHR) a été modifié en liaison avec le registre CL utilisé comme compteur de décalage. Alors que sur les processeurs antérieurs le contenu du registre CL spécifiait un nombre de décalages compris entre 0 et 255, les trois bits supérieurs du registre CL sont désormais annulés avant le décalage, ce qui limite donc le nombre d'opérations de décalage. C'est d'ailleurs tout à fait logique car au bout de 16 décalages (ou de 17 en cas de décalage passant par le flag Carry), tous les bits d'un mot contiennent de toute façon toujours 0. Des décalages supplémentaires ne pourraient donc en rien changer la valeur de l'argument et ne serviraient qu'à faire perdre un temps précieux au processeur.

Le second test repose sur cette différence de comportement. Il consiste en effet à faire décaler de 021h positions sur la droite la valeur 0ffh dans le registre AL, à l'aide de l'instruction SHR. Si le processeur employé est le 80188 ou un des processeurs suivants, il masquera tout d'abord les 3 bits supérieurs du compteur de décalage et ne laissera donc subsister, sur les 021h décalages réclamés, qu'un seul décalage.

```
  021h (00100001b)    Nombre de décalages.
& 01fh (00011111b)    Masquer les 3 bits du haut
-----------------------------------------------
  001h (00000001b)    Nombre effectif de décalages.
```

Contrairement à leurs prédécesseurs, qui auraient effectivement décalé la valeur 0ffh 021h fois sur la droite et auraient donc renvoyé la valeur 0 en réponse, le 80188 et le 80186 renverront donc la valeur 07fh comme résultat. En examinant le registre AL après le décalage, on peut donc aisément déterminer si le processeur est un 80186 ou 80188 (auquel cas AL est différent de 0) ou un autre modèle (AL égal 0).

Différence entre le 8086/8088 et NEC V20/V30

Si le résultat de ce test est encore négatif, le processeur ne peut plus être qu'un 8088 ou 8086 ou bien un NEC V20 ou V30. Les deux derniers modèles cités sont des clones des 8088 ou 8086, dont le jeu d'instructions est identique à leurs homologues Intel mais qui travaillent nettement plus vite grâce à une optimisation de la logique interne et à une construction améliorée. Cet avantage est cependant contrebalancé par un coût nettement plus élevé qui dissuade la plupart de fabricants de PC d'implanter ces processeurs sur leurs ordinateurs.

En dehors de l'exécution plus rapide des instructions, ces processeurs éliminent aussi un petit défaut qui apparaît sur certains processeurs 8088 et 8086. En effet, lorsqu'une interruption électronique est déclenchée au cours de l'exécution d'une instruction de chaîne (par exemple LODS) avec le préfixe REP(eat) et un segment Override, l'exécution de cette instruction ne reprend pas une fois l'interruption terminée. On peut aisément s'en rendre compte au fait que le registre CX qui fonctionne comme compteur de boucle pour cette instruction ne contient pas la valeur 0 après exécution de cette instruction, contrairement à ce qu'on pourrait attendre.

Ce comportement est exploité par le programme de test qui charge la valeur 0ffffh dans le registre CX, faisant ainsi exécuter 65535 fois l'instruction de chaîne suivante avec préfixe REP et segment Override. Comme même un processeur rapide aura besoin d'un certain temps pour exécuter cette instruction, il est absolument certain qu'une interruption électronique sera déclenchée au cours des 65535 exécution de cette instruction. Avec un 8088 ou un 8086, l'exécution de l'instruction ne reprendra pas après cette interruption et le nombre de parcours de boucle restants ne sera plus exécuté. Le programme de test examine donc si c'est le cas d'après le contenu du registre CX après exécution de l'instruction.

Déterminer la largeur du bus de données

Si une distinction a pu être opérée de cette manière entre les 8088 ou 8086 d'un côté et les V20 ou V30 de l'autre, un dernier test est effectué pour tous les processeurs (à

l'exception du 80286 et du i386). Ce test a pour objet de déterminer si le processeur identifié correspond à une version dotée d'un bus 8 bits ou d'un bus 16 bits. Il s'agit donc ici de distinguer le 8088 du 8086, le V20 du V30, ou le 80186 du 80188. Il n'est pas possible de déterminer la largeur du bus de données à l'aide d'instructions du langage machine mais la longueur de la file d'attente (queue) à l'intérieur du processeur dépend directement de cette largeur.

Cette file d'attente a pour fonction de recevoir les instructions qui suivent l'instruction actuellement exécutée. Ces instructions n'ayant plus ensuite à être chargée à partir de la mémoire mais pouvant être retirées directement de la file d'attente, la vitesse d'exécution du processeur en est donc accrue.

Normalement, le remplissage de la queue s'effectue à l'insu du programme à condition qu'il utilise la méthode de la reprogrammation dynamique. Dans ce cas, le changement du code programme est réalisé par le programme où une instruction langage machine est par exemple réécrite par une autre dans un segment de code. C'est une technique rarement utilisée qui convient dans des situations particulières.

Cette procédure devient problématique quand il s'agit de modifier des commandes dont les codes sont déjà chargés dans la queue. Un tel cas se présente lorsque la commande à modifier se trouve très près de la commande chargée de modifier. Dans l'extrait de code suivant, une instruction INC DX est remplacée par une instruction NOP. L'instruction INC est tout de même exécutée parce que les deux instructions se trouvent tellement rapprochées qu'au moment de son exécution, l'instruction INC modifiée se trouvait déjà dans la queue.

```
mov byte ptr cs:queue,Code NOP
queue:inc dx
```

Cette méthode permet de tester facilement la longeur de la queue et du bus de données. Alors que cette file d'attente a une longueur de 6 octets sur les processeurs dotés d'un bus de données 16 bits, cette longueur n'est que de 4 octets sur les processeurs dotés d'un bus de données 8 bits.

Le dernier test de notre routine de test repose donc sur cette différence de longueur. A l'aide de l'instruction de chaîne STOSB (store string byte) avec le préfixe de répétition REP, nous modifions 3 octets placés peu après l'instruction STOSB. Le test repose sur le fait que ces octets doivent être placés de telle manière qu'ils se trouvent déjà à l'intérieur de la file d'attente sur un processeur avec file d'attente de 6 octets. Le processeur ne pourra donc prendre en compte cette modification du code programme. Avec un processeur à file d'attente de 4 octets, au contraire, ces instructions se trouveront encore en dehors de la file d'attente lors de l'exécution de l'instruction de chaîne et les instructions modifiées seront donc chargées sous leur version modifiée lors du prochain remplissage de la file d'attente. Le programme en tire parti en plaçant parmi les instructions modifiées une instruction INC DX, qui incrémente le contenu du registre DX, chargé dans la routine de recevoir le code du processeur identifié. Cette instruction ne sera donc exécutée que si le processeur dispose d'une file d'attente de 6

octets et si cette instruction figurait, de ce fait, déjà dans la file d'attente avant que le code programme n'ait été modifié.

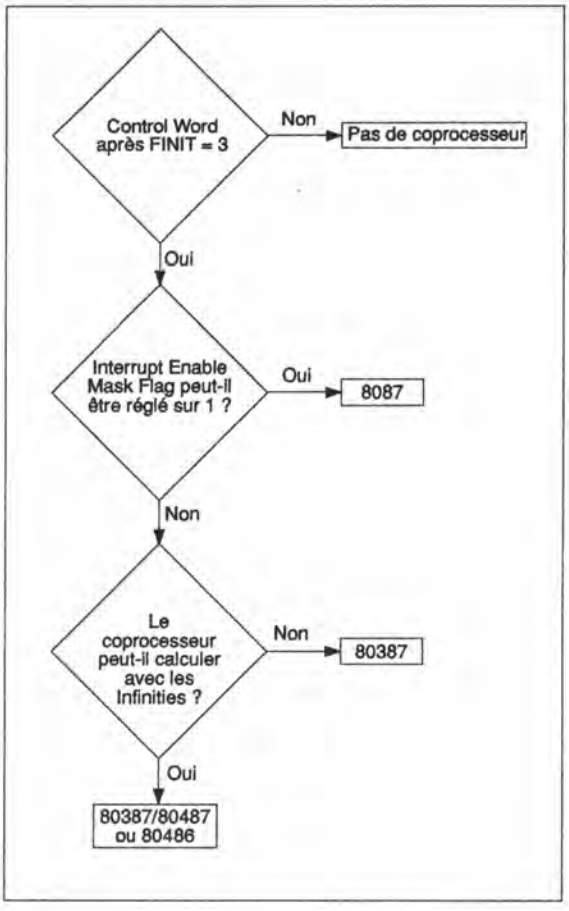
Sur un processeur à file d'attente de 4 octets, cette instruction sera remplacée par l'instruction STI, qui n'affecte pas le contenu du registre DX, ni donc le code du processeur, mais fixe simplement sur 1 le bit d'interruption du registre de flags. Le code du processeur sera donc toujours augmenté d'une unité sur les processeurs xx86. Nous avons naturellement choisi les codes des processeurs de telle manière que le code du processeur xx88 soit toujours suivi du code de son homologue xx86. Par conséquent, si le code de processeur 4, qui correspond au 80188, est chargé dans le registre DX avant le début de ce dernier test et s'il s'agit en fait d'un 80186, le code du processeur sera encore augmenté d'une unité au cours de ce dernier test. On obtiendra bien ainsi le code de processeur 5, que notre programme interprète comme le code du 80186.

14.2.2. Test du co-processeur

Outre le processeur, il est également intéressant de déterminer le co-processeur bien que moins de 3 % des ordinateurs soient équipés d'un tel outil de calcul.

L'instruction de langage machine FSTCW permet de déterminer très facilement la présence d'un co-processeur mathématique en prenant soin auparavant de réinitialiser le co-processeur avec l'instruction FINIT. FSTCW place le Control Word du co-processeur dans une variable spécifiée à l'avance et, après l'initialisation, l'octet de poids fort de ce registre contient toujours la valeur 3.

On s'aperçoit très vite de l'absence d'un co-processeur lorsque l'instruction FSTCW ne peut pas être exécutée et la cellule de mémoire spécifiée ne contient pas la valeur 3, à moins d'avoir déjà stocké cette valeur préalablement.



Test pour déterminer la présence du co-processeur

Si cette méthode a permis de signaler l'existence d'un co-processeur, il faut ensuite déterminer son type. Il s'agit en fait de faire la différence entre un 8087, 80287 et i387 car le co-processeur intégré dans le i486 est identique au i387. Il en est de même pour le i487 qui représente en fait un i486 dans lequel la partie i486 est simplement désactivée.

Différence entre le 8087 et 80287/i387

Par rapport au 80287 et au i387, le 8087 se différencie par un flag situé dans le Control Word appelé Interrupt Enable Mask. Dans le 8087, il décide si le co-processeur doit déclencher une interruption lorsqu'il reçoit une instruction de langage machine incor-

recte. Comme les 80287 et i387 utilisent désormais les exceptions au lieu des interruptions, ce flag est toujours fixé sur 0 et jamais sur 1 dans le cas des 80287/i387.

Cela permet de tester que ce flag est d'abord mis sur 0 lors du chargement du Control Word puis sur 1 par l'instruction FDISI. Cette instruction reste toutefois sans effet dans le cas des 80287/i387 où la détermination du Control Word et le test de ce flag sont exigés immédiatement.

Différence entre le 80287 et le i387

L'affaire se complique lorsqu'on ne travaille pas avec un 8087 car les flags ne servent plus à rien. La différence entre les deux processeurs repose sur leurs capacités à traiter les valeurs infinies désignées par "Infinities".

Une Infinity se produit à l'intérieur d'une routine de test lorsque un est divisé par zéro. Le résultat de cette opération est ensuite dupliqué, son signe inversé et les deux valeurs comparées. Sachant que l'inversion du signe des Infinities ne fonctionne qu'avec un i387, la diversité entre ces deux valeurs ne se rencontre que dans un i387. Bien entendu, cela peut être vérifié à l'aide des commandes appropriées et déterminer ainsi l'identité du co-processeur.

14.2.3. Programmes d'exemples

La théorie sur le test du processeur et son assistant sont conçus sous la forme de deux modules en Assembleur chargés d'assurer l'intégration dans les programmes Pascal et C. Ils portent les noms GETPROC et GETCO. Ils représentent des fonctions retournant des valeurs indiquant le type du processeur (GETPROC) ou le type du co-processeur (GETCO).

Les programmes en langage évolué PROCP.PAS et PROCC.C mettent cela en évidence en utilisant les valeurs de ces fonctions en tant qu'indices dans des tableaux String. Ces derniers contiennent les noms des divers processeurs et co-processeurs Intel.

Accès au registre 32 bits

L'examen des deux modules en Assembleur qui sont identiques jusqu'aux déclarations de segment vous permettra de constater les nombreuses instructions DB de la fonction GETPROC. Elles représentent les codes machine des des diverses commandes nécessaires pour effectuer la différence entre le i386 et le i486. Les instructions de langage machine du i386 et du i486 accédant aux registres 32 bits de ces processeurs (EAX, EBX, etc.) entrent en service dans ce contexte. Mais elles ne peuvent être utilisées que

lorsque la pseudo-commande .386 informe explicitement à l'Assembleur que ces dernières doivent être autorisées.

Mais la conséquence fâcheuse de ce phénomène est que l'Assembleur affecte l'attribut "32 bits" au segment de code et le conserve dans le fichier Objet. Le linker que le module Assembleur relie ultérieurement au module en langage évolué n'aime pas du tout cela parce que les différents segments deviennent alors incompatibles. Pour éviter ce désagrément, les instructions sont fournies directement en code machine si bien que l'Assembleur ne peut plus les considérer comme des instructions 386 étendues.

Listing : PROCP.PAS

```
{*********************************************************}
{*                      P R O C P                        *}
{*-------------------------------------------------------*}
{*   Fonction        : Détermine le type du processeur équipant  *}
{*                     un PC.                            *}
{*-------------------------------------------------------*}
{*   Auteur          : MICHAEL TISCHER                   *}
{*   Développé le    : 16.08.1988                        *}
{*   Dernière modif. : 15.02.1992                        *}
{*********************************************************}

program PROCP;

{-- Déclaration des routines Assembleur ---------------------------}

{$L procpa}                             { Intégrer le module Assembleur }

function GetProc : integer; external;
function GetCo   : integer; external;

{-- Types et variables globales -----------------------------------}

type  NomsProc = string[20];            { Tableau des noms de processeur }

const NomProcr : array [0..8] of NomsProc =
                    ( 'Intel 8088',                     { Code 0 }
                      'Intel 8086',                     { Code 1 }
                      'NEC V20',                        { Code 2 }
                      'NEC V30',                        { Code 3 }
                      'Intel 80188',                    { Code 4 }
                      'Intel 80186',                    { Code 5 }
                      'Intel 80286',                    { Code 6 }
                      '!386',                           { Code 7 }
                      '!486' );                         { Code 8 }

      CoName : array [0..3] of NomsProc =
                  ( 'Pas de coprocesseur',              { Code 0 }
                    '8087',                             { Code 1 }
                    '80287',                            { Code 2 }
                    '!387/!487' );                      { Code 3 }

{*********************************************************}
{**                  PROGRAMME PRINCIPAL                **}
{*********************************************************}

begin
  writeln(' PROCP (c) 1988, 92 by Michael Tischer ');
  writeln;
  writeln('Processeur   : ', NomProcr[ GetProc ] );
  writeln('Coprocesseur : ', CoName[ GetCo ] );
  writeln;
end.
```

Listing : PROCPA.ASM

```
;*********************************************************;
;*                    P R O C P A . A S M                *;
;*-------------------------------------------------------*;
;*   Fonction        : Fournit deux fonctions à intégrer dans un  *;
;*                     programmem Pascal et servant à déterminer  *;
;*                     le type du processeur et du                *;
;*                     coprocesseur.                              *;
;*-------------------------------------------------------*;
;*   Auteur          : MICHAEL TISCHER                   *;
;*   Développé le    : 15.08.1988                        *;
;*   Dernière modif. : 17.02.1992                        *;
;*-------------------------------------------------------*;
;*   Pour assembler  : MASM PROCPA;  ou TASM PROCPA      *;
;*                     ... puis intégrer dans un programme Pascal avec *;
;*                      la directive $L du compilateur   *;
;*********************************************************;

;-- Constantes ---------------------------------------------------

p_!486    equ  8                        ;Codes des divers types
p_!386    equ  7                        ;de processeurs
p_80286   equ  6
p_80186   equ  5
p_80188   equ  4
p_v30     equ  3
p_v20     equ  2
p_8086    equ  1
p_8088    equ  0

co_!387   equ  3                        ;Codes des coprocesseurs
co_80287  equ  2
co_8087   equ  1
co_aucun  equ  0

NOP_CODE  equ  90h      ;Code de l'instruction de langage machine NOP
DEC_DX_C  equ  4Ah                              ;Code de DEC DX

;-- Segment de données -------------------------------------------

DATA      segment word public

cpz       dw ?                          ;Pour le test du coprocesseur

DATA      ends

;-- Programm -----------------------------------------------------

CODE      segment byte public                   ;Segment de code

          assume cs:code, ds:data

public    getproc                       ;Fonction rendue accessible pour
public    getco                                 ;d'autres programmes

;-- GETPROC: Détermine le type du processeur équipant le PC ---
;-- Appel de Pascal: fonction getproc : integer;
```

```
;-- Sortie : Numéro du type du processeur (se reporter aux constantes)

getproc  proc near

         pushf                          ;Sauver le contenu du registre de flags

           ;== Tester s'il s'agit d'un antérieur ou postérieur à 80286

         xor  ax,ax                                    ;Régler AX sur 0
         push ax                                    ;et placer sur la pile
         popf                        ;Retirer de la pile comme registre de flags
         pushf                                      ;Replacer sur la pile
         pop  ax                                       ;et ramener dans AX
         and  ax,0f000h     ;Annuler tous les bits sauf les 4 supérieurs
         cmp  ax,0f000h              ;Les bits 12 à 15 valent-ils tous 1?
         je   inferieur_286                ;Oui, modèle antérieur à 80286

                  ;-- Tester s'il s'agit d'un 1486, 1386 ou 80286 ----

         mov  dl,p_80286                    ;Il s'agit dans tous les cas de
         mov  ax,07000h                        ;l'un des trois processeurs
         push ax                          ;Placer la valeur 07000h sur la pile
         popf                               ;Retirer comme registre de flags
         pushf                                     ;et replacer sur la pile
         pop  ax                                ;Ramener dans le registre AX
         and  ax,07000h     ;Masquer tous les bits sauf les bits 12 à 14
         je   pfin                   ;Les bits 12 à 14 valent-ils tous 0?
                                                 ;OUI --> Il s'agit d'un 80286

         inc  dl                                    ;Non, il s'agit d'un 1386
                                  ;ou 1486. Avant tout, prendre en compte 1386

                  ;-- Le test suivant entre 1386 et 1486 concerne ----
                    ;-- une extension du registre EFlags avec un 1486
                                                 ;-- à la position de bit 18.
           ;-- Ce flag n'existant pas dans un 1386, il est impossible
                                 ;-- de modifier son contenu par logiciel.

         cli                                ;Pas d'interruption pour l'instant

db 066h,088h,0ECh          ;mov   ebx,esp       Ranger SP actuel
db 066h,083h,0E4h,0FCh     ;and   esp,0FFFCh    Arrondir à DWORD
db 066h,09Ch               ;pushfd              Registre de flags sur la
db 066h,058h               ;pop   eax           pile après AX
db 066h,088h,0C8h          ;mov   ecx,eax       et ranger dans CX
db 066h,035h,000h,0h,4h,0h;xor   eax,1 shl 18   Transférer le bit d'ali-
 gnement
db 066h,050h               ;push  eax           et le placer dans Flag-
db 066h,09Dh               ;popfd               le registre de flags
db 066h,09Ch               ;pushfd              Remettre le flag sur la
 pile
db 066h,058h               ;pop   eax           puis replacer
db 066h,051h               ;push  ecx           l'ancien contenu du flag
db 066h,09Dh               ;popfd               après AX
db 066h,033h,0C1h          ;xor   eax,ecx       Tester bit AL
db 066h,0C1h,0E8h,012h     ;shr   eax,18        Bit AL après Bit 0
db 066h,083h,0E0h,001h     ;and   eax,1h        Remettre en place tous
db 066h,088h,0E3h          ;mov   esp,ebx       les SP restants.

         sti                                ;Réautoriser les interruptions
         add  dl,al                                    ;AL vaut 1 si 486
         jmp  pfin                                  ;Le test est terminé

           ;== Tester si 80186 ou 80188 ==========================

inferieur_286 label near

         mov  dl,p_80188                        ;Charger le code de 80188
         mov  al,0ffh          ;Régler tous les bits du registre AL sur 1
         mov  cl,021h            ;Nombre d'opérations de décalage dans CL
         shr  al,cl                    ;Décaler AL CL fois vers la droite
         jne  t88_86            ;AL ne vaut pas 0, il s'agit forcément du
                                                          ;80188 ou 80186

           ;== Tester si NEC V20 ou V30 ==========================

         mov  dl,p_v20                      ;Charger code pour NEC V20
         sti                      ;Les interruptions doivent être autorisées
         mov  si,0                  ;Commencer par le premier octet dans ES
         mov  cx,0ffffh                    ;Lire un segment complet
         rep  lods byte ptr es:[si]      ;REP avec un segment Override
                                  ;Fonctionne seulement avec NEC V20, V30
         or   cx,cx                ;Le segment a-t-il été entièrement lu?
         je   t88_86                                ;OUI --> c'est V20 ou V30

         mov  dl,p_8088                  ;NON --> c'est donc un 8088 ou 8086

           ;== Tester si ...88 ou ...86 ou. V20 ou V30 ==========
         ;-- Exécuter le test à l'aide de la queue (comme ci-dessus),
                                 ;--  mais utiliser ici une queue plus petite

t88_86   label near

         push cs                                       ;Placer CS sur la pile
         pop  es                                          ;et ramener comme ES
         std                        ;Vers le bas pour les instructions de chaîne
         mov  di,offset q2_end             ;Placer DI à la fin de la queue
         mov  al,0fbh                         ;Code d'instruction pour "STI"
         mov  cx,3                    ;Exécuter l'instruction chaîne 3 fois
         cli                                  ;Interdire les interruptions
         rep  stosb                         ;Réécrire l'instruction INC DX
         cld          ;A nouveau vers le haut pour les instructions chaîne
         nop          ;Instructions fictives pour remplir la file d'attente
         nop
         nop

         inc  dx                              ;Incrémenter le code du processeur
         nop
q2_end:  sti                                ;Autoriser à nouveau les interruptions

         ;-------------------------------------------------------------

pfin     label near                                 ;Les tests sont terminés

         popf           ;Retirer à nouveau le registre de flags de la pile
         xor  dh,dh          ;Octet de poids fort du code processeur sur 0
         mov  ax,dx       ;code processeur = valeur Return de la fonction

         ret                                       ;Retour au programme d'appel

getproc  endp                                          ;Fin de la procédure

;-- GETCO: Détermine le type du coprocesseur à condition qu'il existe --
;-- Appel de Pascal: function getco : integer;
;-- Sortie     : Numéro du type du coprocesseur (voir les constantes)

getco    proc near

         mov  dx,co_aucun             ;Commencer à partir du CP manquant

         mov  byte ptr cs:wait1,NOP_CODE ;Remplacer commande WAIT 8087
         mov  byte ptr cs:wait2,NOP_CODE                      ;par NOP

wait1:   finit                                        ;Initialiser Cop
         mov  byte ptr cpz+1,0               ;HI-Byte Control-Word sur 0
wait2:   fstcw cpz                            ;Sauvegarder Control-Word
         cmp  byte ptr cpz+1,3               ;HI-Byte Control-Word = 3?
         jne  gcfin                                 ;Non ---> Pas de coprocesseur

                  ;-- Il existe un coprocesseur. Tester 8087 --------

         inc  dx
         and  cpz,0FF7Fh              ;Masquer Interrupt-Enable-Mask-Flag
         fldcw cpz                          ;Charger dans le Control-Word
         fdisi                                        ;Placer flag IEM
         fstcw cpz                                ;Recharger Control-Word
         test  cpz,80h                            ;Le flag IEM est-il mis?
         jne  gcfin             ;OUI ---> c'est un 8087, Terminer le test

              ;-- Tester 80287/1387 ---------------------------------

         inc  dx
         finit                                         ;Initialiser Cop
         fld1                                     ;Nombre 1 sur la pile Cop
         fldz                                     ;Nombre 0 sur la pile Cop
         fdiv                                  ;Diviser 1 par 0, Résultat en ST
         fld  st                                  ;Déplacer ST sur la pile
         fchs                                        ;Permuter le signe dans ST
         fcompp                                ;Comparer ST et ST(1) et poppen
         fstsw cpz       ;Transférer le résultat du Status-Word dans ...
         mov  ah,byte ptr cpz+1  ;registre flags par la mémoire et ...
         sahf                                                ;le registre AX
         je   gcfin                                  ;Zero-Flag = 1 : 80287

         inc  dx  ;Pas de 80287, il s'agit forcément d'un 1387 ou d'un
                                                      ;coproc. intégré du 1486

gcfin:   mov  ax,dx                              ;Résultat de fonction en AX
         ret                                    ;Retour au programme d'appel

getco    endp

;== Fin ================================================================

CODE     ends                                      ;Fin du segment de code
         end                                              ;Fin du programme
```

Listing : PROCC.C

```
/*****************************************************************/
/*                          P R O C C                            */
/*---------------------------------------------------------------*/
/*   Fonction        : Détermine le type du processeur d'un PC   */
/*---------------------------------------------------------------*/
/*   Auteur          : MICHAEL TISCHER                           */
/*   Développé le    : 14.08.1988                                */
/*   Dernière modif. : 10.02.1992                                */
/*---------------------------------------------------------------*/
/*   (MICROSOFT C)                                               */
/*   Création      : CL /AS /c PROCC.C                           */
/*                   LINK PROCC PROCCA                           */
/*   Appel         : PROCC                                       */
/*---------------------------------------------------------------*/
/*   (BORLAND TURBO C)                                           */
/*   Création      : Avec un fichier Project de teneur suivante :*/
/*                   PROCC.C                                     */
/*                   PROCCA.ASM                                  */
/*****************************************************************/

#include <stdio.h>

extern int getproc( void );    /* Pour intégrer la routine Assembleur */
extern int getco( void );

/*****************************************************************/
/**                     PROGRAMME PRINCIPAL                     **/
/*****************************************************************/

void main()

{
static char * procname[] = {  /* Vecteur avec pointeurs sur les noms */
                              "Intel 8088",          /* Code 0 */
                              "Intel 8086",          /* Code 1 */
                              "NEC V20",             /* Code 2 */
                              "NEC V30",             /* Code 3 */
                              "Intel 80188",         /* Code 4 */
                              "Intel 80186",         /* Code 5 */
                              "Intel 80286",         /* Code 6 */
                              "i386",                /* Code 7 */
                              "i486"                 /* Code 8 */
                            };

static char * coproc[]   = {              /* noms des coprocesseurs */
                              "aucun",               /* Code 0 */
                              "8087",                /* Code 1 */
                              "80287",               /* Code 2 */
                              "i387/i487"            /* Code 3 */
                            };

printf(" PROCC (c) 1988, 92 by Michael Tischer \n\n");
printf("Votre PC est équipé d'un processeur %s.\n",
       procname[ getproc() ] );
printf("Coprocesseur : %s.\n\n", coproc[ getco() ] );
}
```

Listing : PROCCA.ASM

```
;*****************************************************************;
;*                          P R O C C A                          *;
;*---------------------------------------------------------------*;
;*   Fonction        : Fournit une fonction à intégrer dans un   *;
;*                     programme C et servant à déterminer le type*;
;*                     de processeur sur un PC                   *;
;*                                                               *;
;*---------------------------------------------------------------*;
;*   Auteur          : MICHAEL TISCHER                           *;
;*   Développé le    : 15.08.1988                                *;
;*   Dernière modif. : 10.02.1992                                *;
;*---------------------------------------------------------------*;
;*   Pour assembler  : MASM PROCCA;  ou TASM PROCCA              *;
;*                     ... puis linker avec un programme C       *;
;*****************************************************************;

IGROUP group _text              ;Regroupement des segments de programme
DGROUP group _bss, _data        ;Regroupement des segments de données
        assume CS:IGROUP, DS:DGROUP, ES:DGROUP, SS:DGROUP

_BSS    segment word public 'BSS' ;Ce segment reçoit toutes les variables
_BSS    ends                              ;statiques non initialisées

_DATA   segment word public 'DATA'      ;Toutes les variables globales et
                                  ;statiques initialisées sont placées
                                                      ;dans ce segment

_DATA   ends

;-- Constantes ---------------------------------------------------

p_i486    equ   8                           ;Codes pour les divers types
p_i386    equ   7                                 ;de processeur
p_80286   equ   6
p_80186   equ   5
p_80188   equ   4
p_v30     equ   3
p_v20     equ   2
p_8086    equ   1
p_8088    equ   0

co_i387   equ   3                           ;Codes pour les coprocesseurs
co_80287  equ   2
co_8087   equ   1
co_aucun  equ   0

NOP_CODE  equ   90h                                 ;Code machine pour NOP

DEC_DX_C  equ   4Ah                                 ;Code machine pour DEC DX

;-- Variables globales -------------------------------------------

_DATA   segment word public 'DATA'

cpz       dw 0                                  ;Pour test coprocesseur

_DATA   ends

;-- Programme ----------------------------------------------------

_TEXT   segment byte public 'CODE'              ;Le segment de programme

public  _getproc   ;Fonction rendue accessible pour d'autres programmes
public  _getco

;-- GETPROC: Détermine le type de processeur équipant un PC ---
;-- Appel en C : int getproc( void );
;-- Sortie     : Numéro du type de processeur (voir les constantes)

_getproc  proc near

          pushf               ;Sauvegarder le contenu du registre de flags
          push  dl

            ;-- Tester si i486/i386/80286 ------------------------

          xor   ax,ax                                   ;Régler AX sur 0
          push  ax                                  ;et placer sur la pile
          popf                ;Retirer de la pile comme registre de flags
          pushf                                  ;Replacer sur la pile
          pop   ax                                    ;et ramener dans AX
          and   ax,0f000h       ;Annuler tous bits sauf 4 bits supérieurs
          cmp   ax,0f000h           ;Les bits 12 à 15 valent-ils tous 1 ?
          je    inferieur_286                ;OUI, modèle antérieur à 80286

                ;-- Tester s'il s'agit d'un i486, i386 ou 80286 ----

          mov   dl,p_80286                    ;Il s'agit dans tous les cas
          mov   ax,07000h                      ;d'un des trois processeurs
          push  ax                    ;Placer la valeur 07000h sur la pile
          popf                           ;Retirer comme registre de flags
          pushf                                    ;et replacer sur la pile
          pop   ax                               ;Ramener dans le registre AX
          and   ax,07000h     ;Masquer tous les bits sauf les bits 12 à 14
          je    pfin                  ;Les bits 12 à 14 valent-ils tous 0?
```

```
                                        ;OUI --> Il s'agit d'un 80286
        inc  dl                              ;Non, il s'agit d'un 1386
                                    ;ou 1486. Commencer d'abord par 1386

    ;-- Comparer 1386 et 1486 avec un test de file d'attente -
    ;-- A cet effet, on place une instruction DX INC de sorte qu'elle
    ;-- apparaisse dans la queue dans le cas d'un 1486 et non d'un 1386.
    ;-- Dans le code programme, cette instruction est
    ;-- remplacée par une instruction STI. Si elle ne se trouve pas
    ;-- encore dans la queue (1386) il faut alors incrémenter DX.
    ;-- Si elle est déjà dans la queue (486), cette instruction ne peut
    ;-- toutefois être plus modifiée.

        clf

db 066h,088h,0DCh       ;mov   ebx,esp          garde SP
db 066h,083h,0E4h,0FCh  ;and   esp,0FFFCh       aligne sur un long
db 066h,09Ch            ;pushfd                 sauve reg. flag
db 066h,058h            ;pop   eax              récup dans AX
db 066h,088h,0C8h       ;mov   ecx,eax          et CX
db 066h,035h,000h,0h,4h,0h ;xor  eax,1 shl 18   bit align XOR
db 066h,050h            ;push  eax              sur la pile
db 066h,09Dh            ;popfd                  pour reg de flag
db 066h,09Ch            ;pushfd                 réenpile
db 066h,058h            ;pop   eax              récup dans AX
db 066h,051h            ;push  ecx              ancien flag
db 066h,09Dh            ;popfd                  récupérés
db 066h,033h,0C1h       ;xor   eax,ecx          test bit AL
db 066h,0C1h,0E8h,012h  ;shr   eax,18           décal sur 0
db 066h,083h,0E0h,001h  ;and   eax,1h           masque les autres
db 066h,088h,0E3h       ;mov   esp,ebx          restaure sp


    sti                                         ;interrupt valides
    add  dl,al                                    ; si al=1 -> 486
    jmp  pfin                                     ; fin du test

          ;-- Tester si 80186 ou 80188 ----------

inferieur_286 label near

        mov  dl,p_80188                  ;Charger le code du 80188
        mov  al,0ffh          ;Régler tous les bits du registre AL sur 1
        mov  cl,021h            ;Nombre d'opérations de décalage dans CL
        shr  al,cl                    ;Décaler AL CL fois vers la droite
        jne  t88_86           ;AL ne vaut pas 0, il s'agit forcément du
                                                        ;80188 ou 80186

          ;-- Tester si NEC V20 ou V30 ----------

        mov  dl,p_v20                    ;Charger le code pour NEC V20
        sti                  ;Les interruptions doivent être autorisées
        push si                        ;Ranger le contenu du registre SI
        mov  si,0             ;Commencer par le premier octet dans ES
        mov  cx,0ffffh                  ;Lire un segment complet
        rep  lods byte ptr es:[si]      ;REP avec un Segment Override
                                ;fonctionne seulement avec NEC V20, V30
        pop  si                      ;Retirer à nouveau SI de la pile
        or   cx,cx                ;Le segment a-t-il été entièrement lu ?
        je   t88_86                           ;OUI --> c'est V20 ou V30

        mov  dl,p_8088             ;NON --> c'est donc un 8088 ou 8086

          ;-- Tester si ...88 ou ...86 ou V20 ou V30 ----------

    ;-- Exécuter le test à l'aide de la queue (comme ci-dessus), mais
                                    ;-- utiliser ici une queue plus petite

t88_86  label near

        push cs                                 ;Ranger CS sur la pile
        pop  es                                   ;et ramener dans ES
        std      ;Pour les instructions de chaîne, compter vers le bas
        mov  di,offset q2_end          ;Placer DI à la fin de la queue
        mov  al,0fbh                    ;Code d'instruction pour "STI"
        mov  cx,3                  ;Exécuter 3 fois l'instruction chaîne
        clf                                 ;Interdire les interruptions
        rep  stosb                        ;Réécrire l'instruction DX INC
        cld    ;A nouveau vers le haut pour les instructions de chaîne
        nop                ;Instructions fictives pour compléter la queue
        nop
        nop

        inc  dx                          ;Incrémenter le code processeur
        nop
q2_end: sti                         ;Autoriser à nouveau les interruptions

        ;-------------------------------------------------------------

pfin    label near                              ;Les tests sont terminés

        pop  dl
        popf                        ;Retirer le registre de flags de la pile
        xor  dh,dh          ;Octet de poids fort du code processeur sur 0
        mov  ax,dx            ;Le code processeur =valeur de la fonction
        ret                                 ;Retour au programme d'appel

_getproc endp                                       ;Fin de la procédure

;-- GETCO : détermine la présence d'un coprocesseur ---------------------
;-- Appel en C : int getco( void ); -------------------------------------
;-- sortie : nº dans la liste des coprocesseurs -------------------------

_getco  proc near

        mov  dx,co_aucun                        ;pas de coprocesseur

        mov  byte ptr cs:wait1,NOP_CODE              ;WAIT (80x87)
        mov  byte ptr cs:wait2,NOP_CODE            ;remplacé par NOP

wait1:  finit                                     ;initialise coproc
        mov  byte ptr cpz+1,0                    ;msb mot de control
wait2:  fstcw cpz                               ;récup mot de control
        cmp  byte ptr cpz+1,3                ;coprocesseur présent ?
        jne  gfin                                     ;pas de coproc

;-- Coprocesseur détecté. Test 8087 -------------------------------------

        inc  dx
        and  cpz,0FF7Fh              ;masque d'interruption validées
        fldcw cpz                          ;dans le mot de control
        fdisi                                           ;flag IEM
        fstcw cpz                          ;dans le mot de control
        test cpz,080h                        ;Flag IEM présent ?
        jne  gfin                                    ; oui = 8087

;-- Test pour 80287/1387/1487 -------------------------------------------

        inc  dx
        finit                                     ;initialise 80x87
        fld1                                           ;empile 1
        fldz                                           ;empile 0
        fdiv                                  ;1 / 0, erreur dans ST
        fld  st                                        ;ST empilé
        fchs                                           ;ST=-ST
        fcompp                          ;compare et dépile ST et ST(1)
        fstsw cpz                          ;sauve résultat dans cpz
        mov  ah,byte ptr cpz+1
        sahf                             ;et dans le registre de flag
        je   gfin                             ;flag de zéro =1 : 80287

        inc  dx                          ;sinon 1387, 1486DX ou 1487SX
gfin:   mov  ax,dx                                        ;résultat
        ret

_getco  endp

;-- Fin -----------------------------------------------------------------

_text   ends                                ;Fin du segment de programme
        end                                 ;Fin du source en Assembleur
```

15. L'histoire du DOS en bref

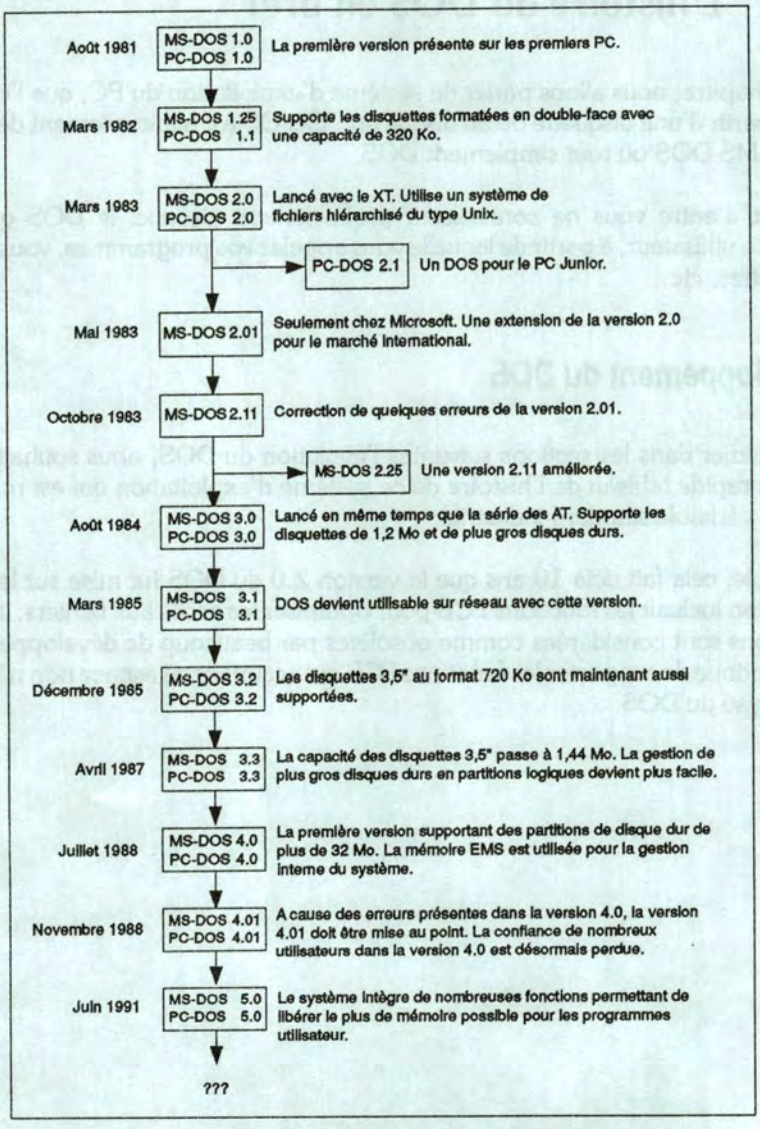
Dans ce chapitre, nous allons parler du système d'exploitation du PC, que l'ordinateur charge à partir d'une disquette ou du disque dur. Cet OS est habituellement désigné par PC-DOS, MS-DOS ou tout simplement DOS.

La plupart d'entre vous ne connaissent probablement jusqu'ici le DOS qu'en tant qu'interface utilisateur, à partir de laquelle vous appelez vos programmes, vous formatez des disquettes, etc...

Le développement du DOS

Avant d'étudier dans les sections suivantes l'évolution du DOS, nous souhaitons vous brosser un rapide tableau de l'histoire de ce système d'exploitation qui est maintenant devenu un véritable standard industriel.

Par exemple, cela fait déjà 10 ans que la version 2.0 du DOS fut mise sur le marché. Cette version incluait les fonctions FCB pour optimiser les accès aux fichiers. Toutefois, ces fonctions sont considérées comme obsolètes par beaucoup de développeurs, mais le DOS continue de supporter les fonctions FCB qui occupent un espace non négligeable dans le noyau du DOS.



L'évolution du DOS

L'histoire du DOS commence dans les années 70 lorsque Intel élabora le microprocesseur 8086, qui est la première génération des microprocesseurs 16 bit. En 1980, la plupart des micro-ordinateurs étaient des systèmes 8 bit et les processeurs Intel 8080 et Zilog Z-80 équipaient la majorité de ces micros. Le PC n'était qu'une vision dans l'esprit des développeurs et le seul OS disponible était le CP/M 80 de Digital Research.

Alors que la société Digital Research avait annoncé une version pour le processeur 8086 du système d'exploitation CP/M 80 (qui devait maintenant s'appeler CP/M 86), cette version n'était toujours pas disponible en avril 1980. C'est alors qu'un programmeur, Jim Paterson, se lança dans le développement d'un nouveau OS appelé QDOS (Quick and Dirty Operating System) et nommé plus tard 86-DOS, ce qui devait devenir l'ancêtre de l'actuel MS-DOS ou PC-DOS.

Il existait cependant à l'époque un grand nombre de logiciels sous CP/M 80. La nécessité de reprendre totalement le développement de ces logiciels aurait donc entraîné des coûts et des problèmes considérables. Le but premier pour Paterson était donc de parvenir à un système qui permette une conversion très aisée des logiciels déjà existants sous CP/M 80. C'est pourquoi il essaya de reproduire le fonctionnement et les principales structures de données de CP/M. Il ne se contenta toutefois pas de copier servilement tout le système CP/M 80 mais essaya, au contraire, de corriger les points faibles de ce système. Le résultat auquel il parvint fut un système d'exploitation occupant seulement 6 Ko et sous lequel on pouvait faire tourner sans trop de travail d'adaptation les programmes développés pour CP/M 80. Ce système fut baptisé 86-DOS.

Or, à la même époque, IBM avait pour projet de construire un micro-ordinateur 16 bits pour lequel Microsoft proposait de développer un système d'exploitation. A cet effet, Microsoft reçut un prototype du nouvel ordinateur d'IBM, acheta le système d'exploitation développé par Paterson et en poursuivit le développement. Paterson restait associé au projet mais les mesures de sécurité extrêmement strictes de la société IBM lui interdisaient ne serait-ce que de voir la machine pour laquelle il était tout de même censé développer un système d'exploitation. Le travail de développement put cependant être achevé pour août 1981 et le nouveau système d'exploitation fut présenté au public, sous le nom de MS-DOS, en même temps que le nouvel ordinateur personnel d'IBM.

La version 1.0

La version 1.0 constituait essentiellement un compromis pour Microsoft car on avait dû coller de très près à CP/M 80 pour permettre un transfert facile des programmes. C'est ce qui explique par exemple que la structure des noms de fichiers (8 caractères pour le nom de fichier et 3 caractères d'extension) soit restée identique à celle de CP/M 80. Mais le parallèle avec le système 8 bits si répandu apparaissait aussi dans la désignation des lecteurs de disquette ainsi que dans l'ensemble de la structure interne.

Il était évident à l'époque que l'électronique des PC était appelée à subir encore de nombreuses améliorations et extensions (des mémoires RAM plus grandes, des lecteurs de disquette plus rapides). C'est pourquoi on décida chez Microsoft que la prochaine étape devait consister à rendre le DOS plus indépendant de la machine. C'est ainsi qu'on fut, par exemple, amené à séparer la longueur de données physique de la longueur de données logique.

Sous CP/M 80, chaque disquette était subdivisée en unités de 128 octets et on ne pouvait accéder à chacune de ces unités qu'en bloc. Il n'était de ce fait pas toujours possible d'accéder à des octets déterminés de la disquette, ce qui entraînait un surcroît de travail inutile. Le DOS put donc éliminer ce défaut en séparant entre les longueurs de données logique et physique. On mit ensuite au point quelques fonctions qui permettaient d'écrire ou de lire simultanément plusieurs enregistrements d'un fichier sur disquette alors qu'on ne pouvait jusqu'ici en traiter qu'un à la fois. L'indépendance par rapport à l'électronique fut également obtenue en traitant les périphériques d'entrée ou de sortie comme des fichiers. Chacun reçut à cet effet un nom particulier.

```
CON   (Clavier et écran)
PRN   (Imprimante)
AUX   (Interface série)
```

Lorsqu'on indiquait un de ces trois noms, au lieu d'un nom de fichier normal, pour accéder à un fichier à travers une routine du DOS, ce n'était plus le lecteur de disquette mais le périphérique correspondant qui était appelé. Ainsi était également rendu possible un détournement du clavier ou de l'écran vers un fichier ou un autre périphérique.

Le DOS ne soutenait auparavant que les fichiers de programme qui avaient été chargés dans un emplacement bien précis de la mémoire centrale pour y être exécutés. Ce vestige de l'époque de CP/M s'avéra peu pratique et on introduisit donc un nouveau type de fichiers de programme. Contrairement à leurs prédécesseurs, les fichiers de ce type ne recevaient pas l'extension de nom de fichier .COM mais étaient désignés par l'extension .EXE et pouvaient être chargés dans (presque) n'importe quel emplacement de la mémoire de l'ordinateur.

En ce qui concerne l'interpréteur de commandes, cette partie du système d'exploitation qui reçoit les instructions de l'utilisateur et qui commande l'exécution de ces instructions, deux innovations permirent d'assouplir ce système. L'interpréteur de commandes fut tout d'abord relégué dans un fichier séparé appelé COMMAND.COM, ce qui permettait à l'utilisateur de développer un interpréteur de commandes personnel, par exemple géré par menu, pour l'intégrer dans le système.

La seconde innovation consistait à scinder l'interpréteur de commandes en une partie résidente et une partie transitoire. C'est qu'en effet la mémoire RAM du PC était encore assez réduite à l'époque et il s'agissait donc de comprimer au maximum la partie qui devait rester disponible en permanence sur le PC. On logea donc sur disquette le code de certaines instructions du système d'exploitation telles que DIR et COPY pour ne le charger dans la mémoire RAM que lorsque cela était nécessaire.

Une autre innovation essentielle, qui constituait en même temps une étape supplémentaire dans la distanciation par rapport aux structures de CP/M 80, consista, en ce qui concerne la gestion de disquettes, à mettre en place une table d'allocation de fichier appelée FAT (File Allocation Table). Chaque entrée de cette table correspond à une zone de données de 512 octets sur la disquette, appelée secteur, et elle indique si le secteur correspondant est déjà attribué à un fichier ou s'il est encore libre. La FAT joue

un très grand rôle à l'égard de l'entrée du répertoire mise en place pour chaque type de fichier. Cette entrée du répertoire indique en effet, outre le nom de fichier et quelques autres informations, le numéro de l'entrée de la FAT qui correspond au premier secteur du fichier sur la disquette.

Cette entrée de la FAT désigne à son tour, comme les suivantes, une autre entrée de la FAT, qui correspond chaque fois au prochain secteur affecté au fichier considéré.

Indiquons encore, pour terminer, deux évolutions qui facilitent le travail de l'utilisateur sur un PC : le traitement batch et le stockage de la date de modification d'un fichier.

Le traitement batch a été introduit à la demande pressante d'IBM. Il permet à l'utilisateur de regrouper plusieurs instructions DOS dans un fichier. Lorsque ce fichier est appelé, le DOS lit automatiquement les différentes instructions de ce fichier, pour les exécuter dans l'ordre, comme si elles avaient été entrées au fur et à mesure au clavier.

Il est souvent extrêmement intéressant pour l'utilisateur de savoir quand un fichier a été modifié pour la dernière fois. C'est pourquoi le DOS inscrit les date et heure actuelles, lors de chaque modification d'un fichier, dans l'entrée de ce fichier dans le répertoire. Ces date et heure sont affichées sur l'écran chaque fois que le répertoire est appelé. Lorsqu'IBM sortit en 1982 un nouveau PC qui utilisait les deux faces de la disquette pour la sauvegarde des données, Microsoft sortit la version PC-DOS 1.1.

Elle comportait les modifications que nous venons de décrire par rapport à la version 1.0 et elle soutenait en outre les lecteurs de disquette double face. Il devenait ainsi possible de stocker maintenant 320 Ko sur une disquette, au lieu de 160 Ko auparavant.

La version 2.0

IBM annonça en mars 1983 un nouvel ordinateur personnel, appelé PC-XT, qui devait disposer non seulement d'un lecteur de disquette comme ses prédécesseurs, mais aussi d'un disque dur. L'énorme capacité de ce disque dur (10 Mo) présentait pour l'utilisateur l'avantage de permettre de loger plusieurs centaines de fichiers sur une unité de mémoire. Cela n'allait pas cependant sans poser quelques problèmes du point de vue du système d'exploitation. Le principal problème était que le DOS ne prévoyait qu'un seul répertoire par support de données. Il semblait pourtant peu souhaitable, pour l'utilisateur, que les innombrables fichiers d'un disque dur fussent réunis dans un même répertoire. Deux options s'offraient à Microsoft pour résoudre ce problème, la première empruntée à CP/M 80, la seconde au système d'exploitation UNIX.

CP/M traite le disque dur comme plusieurs lecteurs de disquette distincts, qui se partagent la mémoire totale disponible sur le disque dur, mais dont chacun ne dispose que d'un seul répertoire.

UNIX utilise par contre un système de fichiers hiérarchisé dans lequel chaque support de données dispose d'un répertoire racine qui peut comprendre non seulement des fichiers mais aussi des sous-répertoires. Chacun de ces sous-répertoires peut à son tour comporter des sous-répertoires et on obtient ainsi une sorte d'arbre de répertoires, dont le tronc est constitué par la racine et dont les branches représentent les différents sous-répertoires.

Tous les utilisateurs du PC savent que c'est le système de fichiers hiérarchisé qui a été choisi et qu'il est maintenant devenu un élément à part entière du DOS. C'était encore une étape supplémentaire vers un éloignement par rapport à CP/M 80 et vers un système d'exploitation 16 bits puissant.

L'introduction du système de fichiers hiérarchisé a fait subir au DOS quelques modifications d'envergure sur le plan de la gestion de fichiers.

L'accès à un fichier se faisait auparavant à travers un bloc de contrôle de fichier appelé FCB (File Control Block). Ce bloc de contrôle avait été mis en place par souci de compatibilité avec CP/M 80 et il contenait quelques informations importantes sur la taille et l'emplacement d'un fichier sur disquette, ainsi que le nom du fichier. Il n'était cependant pas possible d'appeler un fichier d'un autre répertoire (qui n'existait de toute façon pas sous CP/M).

C'est pour cette raison que les développeurs du DOS décidèrent de réorganiser l'accès aux fichiers à travers la fonction du DOS. L'accès à un fichier devait désormais se faire uniquement à travers ce qu'on appelait des file handles, c'est-à-dire des numéros transmis au programme dès qu'un fichier était ouvert à l'aide d'une fonction DOS. Les FCB n'étaient toutefois pas supprimés mais le programmeur n'avait plus à s'en préoccuper car le DOS se chargeait automatiquement du maniement de ces blocs de contrôle.

Une autre innovation importante consistait à introduire la possibilité d'installer des drivers de périphériques. Ces drivers de périphérique permettent à l'utilisateur d'intégrer aisément dans le DOS certains appareils tels qu'un disque dur exotique, une souris ou un streamer. Dans le domaine de la sortie sur écran, la version 2.0 du DOS introduisit le driver de périphérique ANSI.SYS qui permet par exemple au programmeur de positionner le curseur ou de sélectionner les couleurs à travers des fonctions du DOS.

Un premier pas timide vers un travail multi-tâches consista à mettre en place le procédé appelé background processing. C'est une méthode qui consiste à faire tourner à l'arrière-plan un programme qui est appelé chaque fois que le programme de premier plan n'a rien à faire sur le moment. C'est sur ce procédé que repose par exemple le programme PRINT.COM du DOS qui permet d'imprimer un fichier sur l'imprimante pendant qu'un autre programme tourne au premier plan.

Pour augmenter la capacité des disquettes DOS, les différentes pistes d'une disquette ne furent plus formatées avec huit mais avec neuf secteurs à partir de la version 2.0. La

capacité d'une disquette formatée sur une face passait ainsi de 160 à 180 Ko alors que celle d'une disquette formatée double face passait de 320 à 360 Ko.

Extensions de la version 2.0

De même que la version 1.25 corrigeait les bugs de la version 1.0, trois sous-versions de Microsoft et une version 2.0 commercialisée d'IBM apparurent sur le marché au cours de la même année.

La version 2.01 supportait les jeux de caractères internationaux, incluant l'alphabet Kanji. Très peu après la sortie de la version 2.01, la 2.11 apparut pour corriger d'autres petites erreurs. Cette version finale fut le standard DOS jusqu'à la sortie de la version 3.0

IBM demanda un DOS adapté spécifiquement à l'IBM portable. Cet ordinateur était l'un des moyens d'IBM pour dominer le marché des ordinateurs domestiques qui était occupé par Commodore et Atari dans le milieu des années 80. La version 2.1 du DOS ainsi que le portable d'IBM disparurent rapidement du marché. C'était la dernière fois que Microsoft et IBM ne disposaient pas de versions identiques. A part quelques différences mineures entre des constructeurs tiers (par exemple la présence ou non de EXE2BIN.EXE), les numéros de version des deux sociétés étaient équivalents.

En 1985, après la disponibilité de la version 3.0, la version 2 fut mise à jour une dernière fois. La version 2.25 fut mise au point spécialement pour les utilisateurs d'Extrême-Orient, elle supportait les jeux de caractères internationaux, comme par exemple kanji.

La version 3.0 et ses descendantes

Comme pour la version 2.0, c'est l'apparition d'un modèle de PC plus puissant, le PC-AT en l'occurrence, qui entraîna l'apparition de la version 3.0.

Cette version fut présentée au public en août 1984. Elle soutient le disque dur 20 Mo de l'AT mais aussi le lecteur de disquette de grande capacité qui permet de loger jusqu'à 1,2 Mo sur une disquette.

Beaucoup des modifications apportées par rapport à la version 2.0 concernent les routines internes du DOS. Ces modifications ont permis l'exécution plus rapide de certaines opérations bien que ces changements restent invisibles pour le programmeur.

Six mois après, la version 3.1 fut annoncée. C'était la première fois que le support du réseau fut disponible. Quelques nouvelles fonctions furent ajoutées pour implémenter la gestion des réseaux.

La version 3.2 fut lancée en 1985. Cette version supportait les disquettes 3,5" de 720 Ko. Cette version représenta le standard pour des implémentations du DOS jusqu'à la sortie de la version 3.3 en avril 1987.

En plus de vouloir représenter le DOS le plus ouvert, la version 3.3 supportait les périphériques inclus dans les systèmes IBM PS/2 et les disquettes 3,5" haute densité de 1,44 Mo. Cette version fut aussi l'occasion d'améliorer la gestion des langues étrangères à travers les pages de code.

La version 3.3 permit d'améliorer la gestion des disques durs par l'utilisation des partitions. L'utilisateur pouvait partager un disque dur en partitions principale et secondaire, juste comme les lecteurs de disquettes pouvaient être divisés en unités physique et logique. il n'y avait plus besoin de logiciel externe pour gérer le partionnement.

La plupart des changements entre les versions 3.0 et 2.0 étaient internes. Ils permettaient une exécution plus rapide des applications même si cela n'était pas très visible pour l'utilisateur.

La version 4.0

Quatre ans exactement après l'apparition de la version 3.0 du DOS fut présentée, en août 1988, la nouvelle version 4.0 du DOS. Depuis que Microsoft avait annoncé, à l'automne 1987, un nouveau système d'exploitation multitâche, sous le nom d'OS/2 et depuis que sa diffusion avait débuté au printemps 1988, il semblait acquis que les versions futures du DOS ne seraient pas dotées de possibilités de multitâche.

Malgré cela, on peut dire que beaucoup de choses ont néammoins changé avec la version 4.0, notamment du point de vue de l'utilisateur. L'interpréteur de commande orienté ligne a maintenant été remplacé par un environnement utilisateur graphique qui rappelle Windows et Presentation Manager, l'environnement d'OS/2. Cet environnement permet à l'utilisateur de réaliser des menus personnels pour appeler aisément les applications et utilitaires souvent utilisés et de sélectionner des fichiers ou des répertoires sur l'écran, à l'aide de la souris ou du clavier.

L'utilisateur ne voit pas cependant les modifications intervenues dans l'intérieur même du DOS et qui avaient essentiellement pour but d'adapter ce système d'exploitation aux nouveaux standards électroniques. Avec la puissance toujours croissante des PC, les applications traitées deviennent aussi toujours plus complexes, ce qui entraîne également un accroissement constant de la taille des programmes ainsi que des masses de données gérées. Or les versions antérieures fixaient une limite à cette croissance car elles limitaient la capacité maximale des disques durs à 32 Mo "seulement" et la taille de la mémoire centrale à 640 Ko.

Le DOS 4.0 fait sauter ces deux verrous en soutenant les disques durs d'une taille de jusqu'à 2 gigaoctets (2048 mégaoctets) et en soutenant la mémoire étendue (EMS) d'après le standard LIM, qui permet d'installer jusqu'à 8 Mo en plus de la mémoire RAM normale, cette mémoire supplémentaire pouvant être utilisée par les programmes d'application.

Malheureusement, le DOS 4.0 fut lancé avant d'être entièrement testé. De nombreux utilisateurs connurent des pannes système non provoquées et des pertes des données. Microsoft sortit une version déboguée (version 4.01) en novembre 1988, mais beaucoup d'utilisateurs revinrent tout simplement à la version 3.3 et attendirent une vraie version entièrement testée.

La version 5.0

Microsoft mit en place une vaste série de tests avant la sortie de la version 5.0 du DOS. Plus de 7000 utilisateurs et développeurs de logiciels répartis dans le monde entier installèrent et testèrent les beta pour garantir une version 5.0 sans aucun bogue. La version finale fut disponible en juin 1991.

La version 5.0 inclut une utilisation efficace de la RAM, qui fournit plus de RAM aux applications et aux programmes résidents. Comme les drivers de périphériques et les programmes résidents pouvaient être chargés au-delà de la barrière des 640 Ko, cela libérait d'autant plus de RAM utilisateur.

Dix ans après : Un regard en arrière

En résumé, la version 2.0 du DOS fut la base pour toutes les releases suivantes du DOS. Toutefois, les améliorations les plus révolutionnaires sont à venir, et ceci par rapport aux interfaces graphiques comme Windows et OS/2.

DR DOS

Depuis 1981, Digital Research a lancé des systèmes d'exploitation et des interfaces utilisateur graphiques. En particulier, Digital avait lancé l'interface graphique GEM qui fut rapidement supplantée par Microsoft Windows.

Avec DR DOS, Digital Research a accaparé une partie du marché du PC. La plus grande caractéristique de DR DOS est sa compatibilité avec les produits Microsoft. Tout ce que nous disons à propos du DOS dans cet ouvrage s'applique à MS-/PC-DOS et à DR DOS.

16. La structure interne du DOS

Ce chapitre traite de la structure interne du DOS et des procédures qui ont lieu lors du démarrage du système. Ces deux thèmes ne concerne pas la programmation DOS de tous les jours ; en effet, toutes les procédures que nous allons étudier se déroulent "en coulisses". Mais ce sont très souvent les choses cachées qui nous attirent le plus. Nous en savons tous quelque chose. Alors, levons le rideau...

16.1. La structure interne du DOS

Le DOS se compose de plusieurs éléments affectés respectivement à un domaine de fonctions du système. Les trois éléments principaux en sont le DOS-BIOS (à ne pas confondre avec le BIOS), le noyau du DOS et l'interpréteur de commandes. Ces trois éléments sont physiquement séparés les uns des autres par le fait, notamment, qu'ils sont rangés dans des fichiers distincts.

Le DOS-BIOS

Le DOS-BIOS se trouve dans un fichier qui, dans le cas du DOS IBM, porte le nom de IBMBIO.SYS, et dans le cas de MSDOS le nom de IO.SYS. Ce fichier constitue obligatoirement sur chaque PC le premier fichier du répertoire racine du disque d'amorçage et est affecté des attributs de fichier HIDDEN et SYS afin qu'il n'apparaisse pas à l'écran lors de l'exécution de la commande DIR. Il contient les drivers des périphériques suivants :

Nom	Pilote
CON	Clavier et écran
PRN	Imprimante
AUX	Interface série
NUL	Objectif nul !
$CLOCK	Horloge
DISK	Disquette et disque dur, à condition qu'ils ne requièrent pas de drivers particuliers

Lorsque le DOS veut communiquer avec l'un des périphériques, il appelle le driver du périphérique concerné contenu dans ce module, lequel fera à son tour appel aux routines du BIOS. Sachant que seul le DOS-BIOS a accès à la machine (en passant par le BIOS), il est de fait, parallèlement aux pilotes de périphériques décrits dans le fichier de configuration CONFIG.SYS, l'élément du système d'exploitation le plus intimement lié

à l'électronique. Il y a quelques années de cela, alors que les différences entre les PC existants étaient beaucoup plus importantes, les fabricants étaient contraints d'adapter cet élément du DOS à leur matériel.

Le noyau du DOS

Le noyau du DOS est rangé dans le fichier IBMDOS.SYS (pour le DOS IBM) ou dans le fichier MSDOS.SYS (dans le cas de MS-DOS). Il est rangé immédiatement après le fichier IBMBIO.SYS/IO.SYS du répertoire racine du disque d'amorçage. Ce fichier, à l'instar de celui qui le précède, est affecté des attributs HIDDEN et SYS et est donc, en principe, invisible pour l'utilisateur. De la même manière, il ne peut être supprimé car il est également affecté de l'attribut READ ONLY.

Ce fichier renferme les innombrables fonctions du DOS-API auxquelles on accède par l'interruption 21h. Mais c'est également l'ensemble de la gestion interne qui se déroule ici ; ni les utilisateurs, ni les programmeurs n'y ont accès. Toutes les routines sont formulées de manière indépendante de la machine et elles passent par les drivers de périphériques du DOS-BIOS pour accéder aux périphériques tels que le clavier, l'écran et le lecteur de disquettes. Il n'est donc pas nécessaire d'adapter ce module à l'électronique des différents ordinateurs.

L'interpréteur de commandes

A l'opposé des modules présentés jusqu'à présent, l'interpréteur de commandes du DOS, baptisé SHELL, est rangé dans un fichier visible appelé COMMAND.COM. Ce programme est automatiquement lancé lors du chargement du système. C'est lui qui affiche à l'écran le message bien connu A> ou C>, qui reçoit les commandes saisies par l'utilisateur et se charge de les faire exécuter. Etant le seul élément visible du DOS, il est souvent considéré, à tort, par de nombreux utilisateurs comme le système d'exploitation "en personne". Il s'agit en réalité d'un programme comme les autres qui s'exécute sous le contrôle du DOS.

L'interpréteur de commandes n'est toutefois pas un bloc monolithique ; il est en effet organisé en trois modules : une partie résidente, une partie transitoire et la routine d'installation.

La partie résidente, c'est-à-dire la partie qui se trouve en permanence dans la mémoire de l'ordinateur, contient les différents contrôleurs d'interruption du DOS. Cette partie doit donc impérativement résider en mémoire ; dans le cas contraire, l'ordinateur "plantera" au plus tard lors de l'appel de la prochaine interruption machine dont le contrôleur est censé se trouver dans cette partie.

La partie transitoire contient quant à elle le code de sortie du prompt (A>, C>, etc) permettant également la lecture des saisies clavier de l'utilisateur et leur exécution. Ce module doit son nom au fait qu'il peut être remplacé et écrasé par des programmes utilisateur, la mémoire RAM dans laquelle il est stocké n'étant pas protégée. Ceci est sans importance car, le programme utilisateur une fois terminé, le contrôle est alors rendu à la partie résidente de l'interpréteur de commandes. Celle-ci vérifie alors si la partie transitoire a été effacée et, le cas échéant, la charge à nouveau à partir du disque d'amorçage.

Le module d'initialisation est chargé lors du lancement du système et se charge d'initialiser le système. Nous étudierons en détails dans le chapitre suivant cette partie de l'interpréteur de commandes. Cette phase d'exécution une fois achevée, le module n'est plus nécessaire et l'espace qu'il occupait en mémoire vive peut alors être affecté à une autre application.

Exploitation des saisies utilisateur

Les instructions reçues par l'interpréteur de commandes se répartissent en trois groupes : les commandes internes, les commandes externes et les fichiers batch.

Dans la première catégorie, on trouve les instructions dont le code appartient à la partie transitoire de l'interpréteur de commandes. On y trouve notamment les commandes COPY, REN et DIR. Lors du lancement de ces commandes, l'interpréteur de commandes n'a pas besoin de charger un programme ; il exécute ces commandes en tant que sous-programmes.

Contrairement à ce type d'instructions, les commandes externes ne font pas partie de l'interpréteur de commandes et sont stockées sur disquette ou disque dur. Elles doivent donc être préalablement chargées en mémoire afin de pouvoir être exécutées. Citons à titre d'exemple les commandes FORMAT et CHKDSK.

Ces commandes une fois exécutées, la mémoire qu'elles occupaient précédemment est alors rendue disponible pour l'exécution d'autres programmes ; son contenu peut donc être écrasé.

Un fichier batch est un fichier de texte qui contient ligne après ligne les noms des programmes ou des commandes DOS qui doivent être exécutés. Lorsque l'utilisateur indique le nom d'un fichier de ce type, celui-ci est chargé, puis traité dans la partie transitoire de l'interpréteur de commande par un interpréteur spécialement affecté à l'exécution des fichiers batch. Chaque ligne du fichier est alors traitée de la même manière que si l'utilisateur l'avait saisie directement depuis le clavier à partir de la ligne de commande du DOS.

Le fichier batch le plus connu est certainement le fichier AUTOEXEC.BAT qui est chargé et exécuté par le DOS immédiatement après le lancement du système. Les différentes

instructions d'un fichier batch (il en va de même pour les instructions saisies au clavier) sont tout d'abord examinées afin de déterminer s'il s'agit d'une commande DOS interne ou externe ou s'il s'agit d'une des instructions batch qui ne peuvent être utilisées qu'à l'intérieur d'un fichier batch.

S'il s'agit d'une commande interne du DOS, celle-ci pourra être immédiatement exécutée vu que son code se trouve déjà en mémoire, c'est-à-dire dans la partie transitoire de l'interpréteur de commandes. En revanche, s'il s'agit d'une commande externe ou d'un fichier batch, l'interpréteur recherchera tout d'abord dans le répertoire courant un fichier portant le nom indiqué. Si aucun fichier correspondant ne s'y trouve, l'ensemble des répertoires indiqués dans la commande PATH comme répertoires par défaut supplémentaires sera alors examiné dans l'ordre correspondant. La recherche ne portera toutefois que sur les fichiers ayant l'extension .COM, .EXE ou .BAT.

L'interpréteur de commandes ne pouvant examiner simultanément ces trois types de fichiers, il commencera sa recherche par les fichiers .COM, puis .EXE, et finalement .BAT. Si la recherche est infructueuse, un message d'erreur apparaît alors à l'écran et de nouvelles saisies sont alors attendues.

16.2. Lancement du DOS

Le fonctionnement du DOS-BIOS, le noyau, et de l'interpréteur de commandes, tel que nous l'avons vu au chapitre précédent, signifie bien évidemment que tous deux doivent être chargés en mémoire et y être initialisés. C'est ce qui se produit au cours de la procédure d'amorçage que nous allons aborder plus en détails au cours de cette section.

Recherche des fichiers d'amorçage

Au début de la procédure d'amorçage du DOS, le BIOS tente de localiser ce que l'on appelle un secteur d'amorçage. C'est en effet le BIOS qui s'exécute dès que l'ordinateur est mis sous tension, procédant alors à une auto-initialisation, avant de charger le système d'exploitation proprement dit.

S'il détecte un secteur d'amorçage sur une disquette placée dans un lecteur ou sur le disque dur, celui-ci est chargé et la routine de lancement qu'il contient est alors exécutée. Celle-ci vérifie tout d'abord que les deux premiers fichiers inscrits sur le disque d'amorçage sont effectivement les fichiers IBMBIO.SYS et IBMDOS.COM ou IO.SYS et MSDOS.SYS. Si tel n'est pas le cas, le DOS ne peut être chargé et l'utilisateur en est alors averti par un message affiché à l'écran. Si les deux fichiers sont présents, ils sont alors chargés en mémoire et l'exécution du programme peut se poursuivre à l'aide du premier d'entre eux.

Lancement de l'initialisation

La routine d'initialisation est alors exécutée. Cette routine commence par se copier elle-même à la fin de la mémoire RAM d'où elle va gérer la suite des opérations d'initialisation du système. L'étape suivante consiste à transférer à son emplacement définitif dans la mémoire le noyau du DOS qui a été chargé avec le fichier IBMDOS.SYS ou MSDOS.SYS.

La routine d'initialisation appelle alors une routine à l'intérieur du noyau du DOS qui se charge de l'initialisation de ce module. Cette routine met en place certaines tables et zones de données essentielles et appelle les routines d'initialisation des différents pilotes de périphérique qui ont été chargés avec le fichier IBMBIO.SYS ou IO.SYS.

Lecture du fichier de configuration

L'initialisation du noyau du DOS est alors terminée, toutes les fonctions du DOS-API sont maintenant disponibles et la recherche du fichier de configuration CONFIG.SYS peut alors commencer. Lorsque ce fichier est trouvé, il est chargé puis exploité. Le système peut alors être configuré et recevoir de nouveaux pilotes de périphériques, lesquels viennent s'ajouter aux pilotes de périphériques internes du DOS.

La dernière étape consiste finalement à charger l'interpréteur de commandes et à lui transmettre le contrôle des procédures qui suivront. La procédure d'amorçage est ensuite terminée et la routine d'initialisation restera en mémoire, en tant que données usagées, jusqu'à ce qu'elle soit remplacée et écrasée par un autre programme.

17. Programmes COM et EXE

Le DOS reconnaît deux types de fichiers programmes, en dehors des fichiers batch, les programmes COM et EXE. Ces extensions indiquent que ces fichiers disposent de propriétés différentes et qu'ils sont exécutables.

Les différences entre les types de programmes ne sont pas importantes pour un programmeur travaillant avec un langage évolué. Le programmeur désire seulement que son programme fonctionne, peu importe son format. D'ailleurs, les packs de développement comme Turbo Pascal ou QuickBASIC créent uniquement des fichiers EXE. Certains compilateurs C peuvent produire des programmes COM en utilisant le modèle TINY, dans lequel tout le code de programme, le segment de données et la pile tiennent dans 64 Ko de RAM ou moins.

Le seul avantage des programmes COM par rapport aux EXE est la taille du programme qui est plus petite, la différence étant souvent de quelques centaines d'octets.

Le programmeur travaillant avec un langage évolué n'a pas à s'inquiéter à propos des formats et des nuances entre les programmes COM et EXE car le compilateur les gère directement. Toutefois, cette information est importante pour le programmeur qui développe des logiciels en langage assembleur.

Dans ce chapitre, nous allons décrire la structure et les fonctions de ces deux types de programmes.

17.1. Différences entre programmes COM et EXE

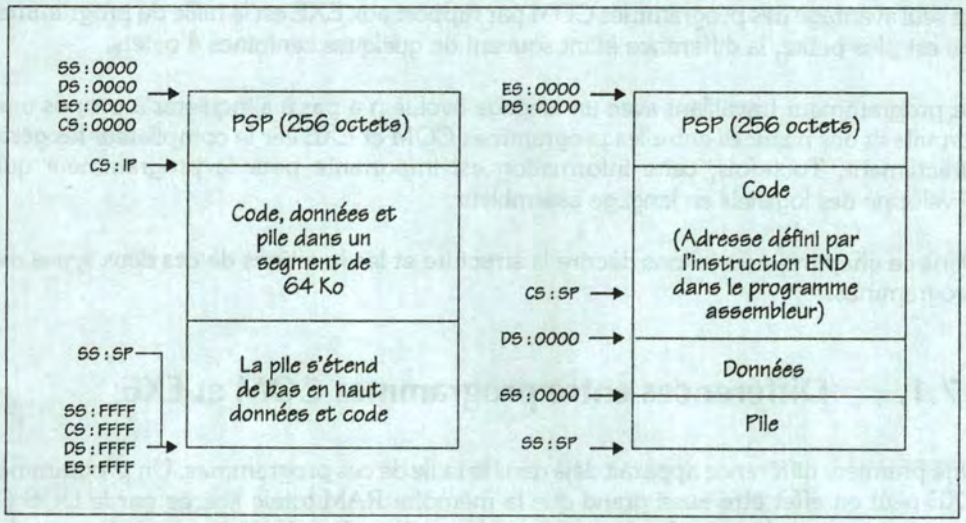
Une première différence apparaît déjà dans la taille de ces programmes. Un programme EXE peut en effet être aussi grand que la mémoire RAM totale libérée par le DOS (il peut même être plus grand mais ne pourrait pas être chargé dans ce cas) alors que la taille d'un programme COM ne doit pas excéder 64 Ko. Le programme COM est une relique de l'époque CP/M où la RAM était minimale et les programmes ne pouvaient pas dépasser 64 Ko.

Il faut loger dans ces 64 Ko aussi bien le code de programme que les données et la pile. La conséquence en est naturellement que tous les registres de segment reçoivent lors du lancement et pendant l'exécution du programme la même valeur et qu'ils désignent le début du segment de mémoire de 64 Ko. Seul le contenu du registre ES peut être modifié car il ne revêt pas de signification directe pour l'exécution du programme.

Pour les fichiers EXE, l'organisation des segments n'est pas prescrite de façon aussi stricte. Le code, les données et la pile sont logés dans des segments différents, chacun de ces éléments pouvant même (suivant sa taille) être réparti sur plusieurs segments.

C'est pourquoi les registres de segment peuvent avoir différentes valeurs au cours de l'exécution d'un programme EXE.

Les deux types de programmes peuvent être chargés et lancés à l'aide de la fonction (DOS) EXEC. Cette fonction, accessible à l'utilisateur mais aussi à l'interpréteur de commandes, sert à l'exécution des instructions externes. Avant que la fonction EXEC ne charge dans la mémoire le programme à appeler, elle réserve à travers d'autres fonctions DOS la mémoire RAM que le programme à appeler devra occuper. La fonction EXEC place au début de cette mémoire une structure de données appelée PSP (program segment prefix). Le programme est chargé à la fin de cette structure de données, les registres de segment et la pile sont initialisés et le programme est finalement lancé. Une fois l'exécution du programme achevée, la mémoire occupée par le programme et le PSP est à nouveau libérée (à moins que cela ne soit empêché par un appel de fonction du programme).



Comparaison entre les programmes COM et EXE en mémoire

17.2. Les programmes COM

Les fichiers COM sont stockés sur la disquette sous une forme reflétant exactement le contenu de la mémoire RAM lorsqu'ils sont chargés. Dans de cas, aucune information supplémentaire n'est nécessaire et les programmes COM peuvent donc être chargés et lancés plus rapidement que les programmes EXE.

Microsoft et les programmes COM

Microsoft supporte de moins en moins les programmes COM. Microsoft Assembler (connu sous le nom de MASM) assemble les fichiers en tant que programmes EXE. Pour les convertir en COM, vous pouvez utiliser EXE2BIN qui est un utilitaire important fourni avec les anciennes versions du DOS. MS-DOS 4.0 ne contient plus EXE2BIN. Le système de contrôle de version interdit justement d'utiliser EXE2BIN des anciennes versions avec les versions plus récentes de MS-DOS.

Aussi, les programmeurs qui désiraient continuer à utiliser EXE2BIN étaient obligés de modifier le système de contrôle de version pour rendre EXE2BIN compatible avec MS-DOS.

Contrairement au programme LINK fourni avec Microsoft Assembler MASM, Turbo Assembler de Borland International peut créer aussi bien des EXE que des COM.

Microsoft fournit à nouveau EXE2BIN avec la version 5.0 du DOS.

Registres au début du chargement du programme

Un programme COM charge immédiatement après le PSP. L'exécution commence dès lors à la première zone mémoire suivant le PSP à l'offset 100H. Pour cette raison, un programme COM doit commencer par une instruction exécutable, même s'il s'agit d'un jump au début réel du programme.

Limites mémoire des COM

Comme nous l'avons déjà expliqué, un programme COM ne peut dépasser une taille de 64 Ko (65 536 octets), y compris la longueur du PSP (256 octets) et au moins 1 mot (2 octets) pour la pile. La longueur d'un programme COM ne peut de ce fait jamais dépasser 64 Ko et pourtant le DOS réserve toujours la totalité de la mémoire RAM disponible pour un programme de ce type. Il ne reste donc jamais de mémoire disponible (du point de vue du DOS) et le programme COM ne peut appeler un autre programme à l'aide de la fonction EXEC.

Ce problème peut toutefois trouver une solution si le programme COM libère à l'aide d'une fonction DOS la mémoire dont il n'a pas besoin pour la rendre disponible pour une autre application.

Lorsque le contrôle est transmis au programme COM, tous les registres de segment sont dirigés sur le début du PSP. Le début du programme COM (par rapport au début du PSP) se situe donc toujours à l'adresse 100h. Le pointeur de pile reçoit la valeur FFFEh et il est ainsi dirigé sur la fin du segment de mémoire de 64 Ko occupé par le

programme COM. Lors de chaque appel de sous-programme à l'intérieur du programme COM, la pile se rapproche de 2 octets vers la fin du programme. Le programmeur (et lui seul) doit donc veiller à ce que la pile ne puisse grossir jusqu'à entrer en collision avec la fin du programme, ce qui entraînerait probablement un plantage du système.

Quitter un programme COM

Il existe plusieurs possibilités pour terminer un programme COM et pour rendre le contrôle au DOS ou au programme d'appel :

Si le programme tourne sous la version 1.0 du DOS, il peut être terminé en appelant la fonction 00h de l'interruption 21h ou bien en appelant l'interruption 20h. Une autre possibilité, qui peut sembler curieuse au premier abord, consiste à terminer le programme par un "Near Return" (Instruction assembleur RET). Lorsque cette instruction est exécutée, le programme se poursuit à l'adresse figurant au sommet de la pile. Or la fonction EXEC a placé en cet endroit la valeur 0 avant de passer la main au programme COM. L'exécution du programme se poursuivra donc de façon fort logique à l'adresse CS:0000. Comme cette adresse représente le début du PSP et qu'elle contient un appel de l'interruption 20h, le programme sera finalement terminé de cette façon.

Sous toutes les versions postérieures, c'est habituellement la fonction 4Ch de l'interruption (DOS) 21h qu'on utilisera. Celle-ci permet au programme qui vient d'être terminé d'envoyer au programme d'appel un message sous forme d'une valeur numérique. Le DOS n'impose ici l'emploi d'aucune valeur déterminée mais la transmission d'une valeur n'a naturellement d'intérêt que si le programme d'appel et le programme appelé lui attribuent la même signification. Une possibilité, souvent utilisée, consistera par exemple à indiquer à l'appelant, à l'aide de la valeur 0, que le travail du programme appelé s'est achevé sans problème, alors que toute autre valeur signalera l'apparition d'une erreur lors de l'exécution du programme.

Développer des programmes COM en assembleur

Un programmeur en langage évolué n'a pas lieu de se préoccuper du développement d'un programme COM car soit le compilateur se chargera pour lui de tout le travail, soit il travaillera sous un interpréteur qui contrôlera l'exécution du programme. Le programmeur en assembleur doit par contre prendre en compte un certain nombre de problèmes lorsqu'il veut développer un programme COM.

Les programmes COM sont stockés sur disquette ou disque dur sous une forme reflétant exactement le contenu de la mémoire lorsque le programme est chargé. Or ces programmes ne sont pas chargés par le DOS à une adresse définie d'avance. Ils peuvent en effet être chargés à n'importe quelle adresse (qui soit un multiple entier de 16).

Les programmes COM ne doivent donc pas comporter de call FAR ni d'adresses de segment explicites. Seuls les call NEAR, qui contiennent des adresses d'offset mais pas d'adresses de sgement, sont autorisés.

Si en effet les adresses d'offset se réfèrent toujours au segment actuel et indiquent donc l'adresse qui convient à l'intérieur d'un segment quelconque, qui peut commencer à n'importe quelle cellule de mémoire, il n'en va pas de même pour les adresses de segments car ces dernières définissent des segments constants alors même que les programmes COM ne peuvent justement être systématiquement chargés à des adresses de segments constantes.

C'est d'ailleurs précisément pour cette raison que la taille d'un programme COM est limitée à 64 Ko et qu'il est interdit au programmeur d'utiliser des instructions FAR dans son programme. Les instructions FAR sont toutes les instructions qui permettent de poursuivre l'exécution du programme dans un autre endroit du programme, en définissant non seulement l'adresse d'offset mais aussi l'adresse de segment du nouvel emplacement. Le programmeur doit donc se limiter aux instructions NEAR, celles qui n'indiquent qu'une adresse d'offset.

Les programmes COM ne peuvent pas contenir des instructions assembleur comme LDS ou LES. L'assembleur et le linker accepteront ces instructions mais EXE2BIN refusera la conversion. Cela s'applique aussi au Turbo Linker TLINK.

Des valeurs constantes peuvent cependant être chargées dans un registre de segment, comme l'adresse de segment de la RAM vidéo par exemple, mais il n'est pas possible de charger dans un registre de segment l'adresse du segment de code ou de données du programme COM. Ces adresses peuvent en effet varier lors de chaque appel du programme. Il découle du même principe qu'un programme COM ne peut non plus comporter plusieurs segments et que les variables doivent appartenir au segment unique, c'est-à-dire au segment de code.

Les instructions suivantes sont par exemple interdites car l'instruction MOV lit l'adresse de segment d'un programme :

```
MOV  AX,SEG PROGRAM
MOV  DS,AX
```

Les instructions suivantes sont acceptées car une adresse de segment constant est utilisée en référence :

```
MOV  AX,0B000h
MOVE DS,AX
```

Nous avons déjà indiqué également qu'un programme COM n'a pas l'embarras du choix pour ce qui est de la définition de la pile : celle-ci est en effet automatiquement placée à la fin du segment COM de 64 Ko. Avant d'appeler un programme COM, le DOS réserve pour ce programme toute la mémoire disponible. Cela ne pose d'ailleurs aucun

problème dans la majorité des cas car ce programme disparaît de la mémoire après avoir été exécuté et la mémoire qu'il occupait est à nouveau libérée.

Si toutefois le programme COM se transforme en programme résident, c'est-à-dire s'il reste dans la mémoire après son exécution, un certain nombre de problèmes peuvent en résulter puisqu'il n'y a (du point de vue du DOS) plus de mémoire disponible et qu'il n'est donc plus possible de charger et d'exécuter d'autres programmes.

Un autre problème apparaît chaque fois qu'un programme COM en cours d'exécution veut charger ou exécuter un autre programme à l'aide de la fonction EXEC. Cela est également impossible puisque le DOS considère qu'il n'y a plus de mémoire disponible pour recevoir le programme appelé. Ces deux problèmes peuvent toutefois être résolus en libérant la place mémoire dont le programme n'a pas besoin.

On dispose à cet effet de deux possibilités : soit on libère uniquement la mémoire en dehors du segment COM de 64 Ko, soit on libère la totalité de la mémoire dont le programme n'a pas besoin, y compris la mémoire inutilisée à l'intérieur du segment COM. Cette seconde solution permet bien sûr de libérer encore plus de place pour d'autres programmes mais la pile se retrouve alors en dehors du segment de mémoire COM et elle peut de ce fait être effacée par d'autres programmes. C'est pourquoi il est nécessaire, dans ce cas, de déplacer la pile vers la fin du code de programme et de ne libérer que la mémoire placée après la fin de la pile. Il est bien sûr indispensable pour cela d'attribuer à la pile une taille déterminée. Dans la plupart des cas, 512 octets seront très largement suffisants.

L'exemple de programme suivant pourra vous servir de modèle pour tous les programmes COM que vous développerez. Une petite routine déplace la pile, immédiatement après le lancement du programme, vers la fin du segment de code et libère toute la mémoire restante. Même si votre programme ne charge pas d'autre programme et même s'il ne reste pas résident après exécution, cette routine peut se révéler utile et nous vous conseillons de l'intégrer dans tous vos programmes COM. Si l'exécution de plusieurs programmes à la fois est permise dans des versions ultérieures du DOS, cela ne sera en effet possible que si chaque programme dispose de suffisamment de mémoire. Or cela ne sera le cas que si la mémoire inutilisée est libérée.

Listing : COM_RAM.ASM

```
;*****************************************************************;
;*                      COM_RAM.ASM                              *;
;*---------------------------------------------------------------*;
;*  Fonction       : Outil de développement de programmes COM en *;
;*                   Assembleur avec le Turbo Assembleur TASM ou *;
;*                   le Macro Assembler MASM de Microsoft        *;
;*---------------------------------------------------------------*;
;*  Auteur         : MICHAEL TISCHER                             *;
;*  Développé le   : 07.06.1987                                  *;
;*  Dernière modif. : 10.10.1991                                 *;
;*---------------------------------------------------------------*;
;*  Développement  : MASM:  masm com_ram;                        *;
;*                          link com_ram;                        *;
;*                          exe2bin com_ram.exe com_ram.com      *;
;*                  TASM:  tasm com_ram                          *;
;*                          tlink /t com_ram                     *;
;*****************************************************************;

.com    segment para 'CODE'             ;Définition d'un segment COM
                                        ;(le nom peut être choisi librement)

        org 100h                        ;Le code commence à l'adresse 100h
                                        ;obligatoirement après le PSP

        assume cs:com, ds:com, es:com, ss:com

                                        ;Pendant l'exécution du programme,
                                        ;tous les registres de segment désignent
                                        ;le segment COM
```

```
;
start:   jmp  init              ;Ici commence l'exécution du programme
                                ;Saut vers la routine d'initialisation
;
;-- Données ---------------------------------------------------------
;
         ;-- Ici, vous pouvez mettre en place toutes les données, le --
                                        ;-- le buffer et les variables
;
                                                        ;...
                                                        ;...
                                                        ;...
;
;-- Programme principal ---------------------------------------------
;
prog     proc near              ;Cette procédure représente le programme
                        ;principal à proprement parler. Elle est exécutée
                                        ;après le lancement du programme.
;
               ;-- Ici, vous pouvez inclure le code programme du------
                                        ;-- programme principal
;
                                                        ;...
                                                        ;...
                                                        ;...
;
    ;--- Terminer le programme à l'aide de la fonction DOS 4Ch ---------
;
         mov  ax,4C00h  ;Charger le numéro de fonction, code d'erreur 0
         int  21h                                       ;Appels DOS
;
      ;--- Ici l'exécution du programme n'a plus aucun sens parce que --
                        ;--- l'appel DOS précédent a mis fin au programme
;
prog     endp                           ;Fin du programme principal
;
;-- Sous-routines ---------------------------------------------------
;-- Cette place est réservée aux sous-routines du programme
;
a_proc   proc near
;
                                                        ;...
                                                        ;...
                                                        ;...
;
         ret
;
a_proc   endp
;
b_proc   proc near
;
                                                        ;...
                                                        ;...
                                                        ;...
;
         ret
;
b_proc   endp
;
;-- Initialisation --------------------------------------------------
;-- Ici, toute la mémoire est libérée après le label FIN
;-- et la pile réorganisée.
;
init:    mov  ah,4Ah          ;N° fonc. pour "Changer la taille mémoire"
         mov  bx,offset fin     ;Arrondir la longueur du programme aux
         add  bx,15              ;paragraphes suivants dans la mémoire
         mov  cl,4                       ;Calculer l'adresse d'offset
         shr  bx,cl                                    ;en paragraphes
         inc  bx
         int  21h                ;Appeler la fonction par l'inter. DOS
;
         mov  sp,offset fin                      ;Réorganiser la pile
         jmp  prog                ;Sauter vers le programme principal
;
init_fin label near
;
;-- Pile ------------------------------------------------------------
;
         dw (256-((init_fin-init) shr 1)) dup (?)
;
                                 ;La pile comprend 256 mots et termine
                               ;le code de la routine INIT qui devient
                                                    ;inutile après son
                                                          ;exécution
;
fin      equ this byte          ;Fin de la mémoire occupée, aucun code
                                                    ;ne doit plus faire suite
;
;-- Fin -------------------------------------------------------------
;
com      ends                                      ;Fin du segment COM
         end  start             ;Fin du programme Assembleur. Commencer
                                  ;l'exécution avec l'instruction START
;
```

Lorsque vous voulez faire exécuter un programme COM, vous devez naturellement l'assembler à l'aide du macro-assembleur, comme un programme normal. Vous devez ensuite le linker avec le programme LINK. Ce programme sortira alors le message :

```
Warning: no stack segment
```

Vous ne devez pas vous laisser impressionner par ce message. Si votre programme ne comporte pas d'erreur, le programme LINK produira tout d'abord un fichier EXE. Mais comme nous voulons développer un programme COM, et non un programme EXE, la dernière étape consistera à appeler ensuite le programme EXE2BIN. Ce programme convertira le programme EXE en un programme COM. Voici les différentes étapes à suivre pour obtenir un programme assembleur intitulé TEST.ASM

```
masm test;
link test;
exe2bin test.exe test.com
```

TASM et TLINK de Borland effectuent les mêmes tâches que MASM, LINK et EXE2BIN mais de manière quelque peu différente. Assemblez le fichier TEST.ASM avec TASM comme suit :

```
TASM TEST
```

Utilisez la syntaxe suivante pour créer directement le code COM :

```
TLINK /t TEST
```

17.3. Les programmes EXE

Par rapport aux programmes COM, les programmes EXE présentent l'avantage de ne pas être limités à une longueur maximale de 64 Ko pour le code, les données et la pile. Le prix à payer en échange de cet avantage est représenté par la plus grande complexité de ces fichiers qui est due à la présence dans un fichier EXE de toute une série d'informations autres que le programme lui-même. Pour les versions ultérieures du DOS, il peut cependant tout à fait en résulter un avantage supplémentaire, à savoir que ces programmes seront sans doute plus commodes à adapter à différentes innovations telles que l'introduction du travail multi-tâches par exemple.

EXE contre COM

Les programmes EXE comportent des segments distincts pour le code, les données et la pile. Ces segments peuvent se présenter dans n'importe quel ordre. Contrairement aux programmes COM, les programmes EXE ne sont pas chargés de la disquette ou du disque dur dans la mémoire pour être exécutés directement. Avant d'être exécutés, ils subissent en effet une préparation effectuée par une routine de la fonction EXEC. Cette préparation est nécessaire pour résoudre les problèmes déjà évoqués lors de la description des programmes COM.

Les programmes EXE ne sont pas non plus chargés dans un emplacement de la mémoire défini d'avance mais en n'importe quelle adresse (multiple entier de 16). Comme un programme EXE peut comporter plusieurs segments, l'utilisation d'instructions (langage machine) FAR est indispensable si l'on veut par exemple appeler à partir d'un segment un sous-programme situé dans un autre segment. Or une instruction FAR doit comporter non seulement l'indication de l'adresse d'offset mais encore celle de l'adresse de segment. D'où un premier problème puisque cette adresse de segment peut varier lors de chaque exécution du programme.

Dans les fichiers COM, ce problème est résolu de façon simple mais sévère puisque la taille des programmes est limitée à 64 Ko, ce qui rend l'emploi d'instructions FAR superflu. Les programmes EXE résolvent ce problème de façon plus complexe mais aussi plus constructive. Le programme LINK met en effet en place au début de chaque fichier EXE une structure de données qui contient entre autres les adresses de toutes les références de segment, c'est-à-dire l'adresse de toutes les cellules de mémoire dans lesquelles devra figurer l'adresse de segment d'un segment quelconque lors de l'exécution du programme.

Lorsque le programme EXE est chargé par la fonction EXEC, celle-ci connaît les adresses auxquelles sont chargés les différents segments du programme EXE. Elle peut ainsi inscrire les valeurs appropriées dans les cellules de mémoire enregistrées dans la tête du fichier EXE. Cette opération entraîne un délai plus long entre l'appel et le lancement du programme que pour un programme COM et la taille d'un fichier EXE sera toujours supérieure à celle d'un fichier COM équivalent mais ces inconvénients sont négligeables au regard de l'avantage qu'il y a à pouvoir développer des programmes d'une taille supérieure à 64 Ko. Le tableau suivant vous présente la structure du header d'un fichier EXE.

Structure du header d'un fichier

Adresse	Contenu	Type
00h	Marque d'un programme EXE (5A4Dh)	1 WORD
02h	Longueur du fichier MOD 512	1 WORD
04h	Longueur du fichier DIV 512	1 WORD
06h	Nombre d'adresses de segment à adapter	1 WORD
08h	Taille du header en paragraphes	1 WORD
0Ah	Nombre minimal de paragraphes nécessaires	1 WORD
0Eh	Nombre maximal de paragraphes nécessaires	1 WORD
10h	Contenu du registre SP au lancement du programme	1 WORD
12h	Checksum sur l'en-tête du fichier EXE	1 WORD
14h	Contenu du registre IP au lancement du programme	1 WORD
16h	Début du segment de code dans le fichier EXE	1 WORD
18h	Adresse de la table de relogement dans le fichier EXE	1 WORD
1Ah	Numéro d'overlay	1 WORD
1Ch	Mémoire-tampon	variable
??	Adresses des adresses de segment à adapter	variable
??	Code programme, segments de données et de pile	variable

Après que les références de segment à l'intérieur du programme EXE aient été adaptées aux adresses effectives, la fonction EXEC fixe les registres de segment DS et ES sur le début du PSP, qui précède également dans la mémoire tout programme EXE, de même que tout programme COM. Le programme EXE peut donc accéder très aisément aux informations contenues dans le PSP telles que l'adresse du bloc d'environnement et les paramètres transmis dans la ligne d'instruction. L'adresse de la pile et le contenu du pointeur de pile sont stockés dans la tête du fichier EXE, d'où ils sont repris. Cela vaut également pour l'adresse du segment de code dans lequel figure la routine de lancement

du programme, ainsi que pour le compteur de programme. Après qu'ils aient reçu les valeurs définies, l'exécution du programme commence.

Pour garantir la compatibilité avec les versions ultérieures du DOS, il convient de terminer un programme EXE en appelant la fonction 4Ch de l'interruption (DOS) 21h.

Affectation de la RAM

Il convient naturellement aussi de mettre de la mémoire à la disposition de tout programme EXE chargé. Cela ne se fera cependant pas tout à fait comme pour les programmes COM. Le chargeur EXE peut obtenir du fichier EXE la taille des différents segments du programme EXE et donc du même coup la taille globale du programme dans la mémoire. Il peut ensuite réclamer la mémoire nécessaire à l'aide d'une autre fonction. La mémoire réservée dépasse naturellement la mémoire de programme proprement dite. La taille de cette mémoire supplémentaire est fixée d'après le contenu de deux champs du header du fichier EXE qui indiquent les dimensions minimale et maximale de cette zone en paragraphes.

Le chargeur EXE tente dans un premier temps de réserver le nombre maximum de paragraphes. Si cela n'est pas possible, il se contente de la mémoire restante à condition toutefois qu'elle ne soit pas en-deçà du nombre minimum de paragraphes. Ces deux champs ne sont pas fixés par le linker mais par le compilateur utilisé ou en l'occurrence par l'assembleur. Il indique systématiquement FFFFh comme maximum. Cette dernière valeur est d'ailleurs parfaitement illusoire (elle correspondrait à 1 Mo) mais elle a pour effet de faire réserver toute la mémoire disponible pour le programme EXE.

Et voilà que nous retrouvons les mêmes problèmes que pour les programmes COM. Les programmes EXE ont cependant une structure qui ne convient pas très bien pour les installer comme programmes résidents mais il se peut très bien que tel ou tel programme ait besoin d'appeler un autre programme pendant son exécution. Or cela n'est possible, comme pour les programmes COM, que si la mémoire réservée en trop est libérée. C'est pourquoi nous vous présentons à nouveau un exemple de programme qui contient une routine pour réduire au minimum la mémoire réservée.

Sa structure comporte un segment de code ainsi qu'un segment de données et un segment de pile séparés. Tant que vos segments ne dépassent pas une taille de 64 Ko, c'est-à-dire tant que vous n'êtes pas contraint de les répartir sur plusieurs segments physiques, vous pouvez donc utiliser ce programme comme base de tous vos programmes EXE.

Listing : EXE_RAM.ASM

```
;*****************************************************************;
;*                      E X E _ R A M . A S M                    *;
;*---------------------------------------------------------------*;
;*   Fonction        : Outil de développement de programmes EXE en *;
;*                     Assembleur avec le Turbo Assembleur TASM ou *;
;*                     le Macro Assembler MASM de Microsoft        *;
;*---------------------------------------------------------------*;
;*   Auteur          : MICHAEL TISCHER                           *;
;*   Développé le    : 7.06.1987                                 *;
;*   Dernière modif. : 10.10.1991                                *;
;*---------------------------------------------------------------*;
;*   Développement   : MASM: masm exe_ram;                       *;
;*                           link exe_ram;                       *;
;*                                                               *;
;*                     TASM: tasm exe_ram                        *;
;*                           tlink exe_ram                       *;
;*****************************************************************;

;-- Pile ------------------------------------------------------------

stackseg  segment para STACK 'STACK'      ;Définition du segment de pile

          dw 256 dup (?)                  ;La pile comprend 256 mots

stackseg  ends                            ;Fin du segment de pile

;-- Données ---------------------------------------------------------

data      segment para 'DATA'             ;Définition du segment de données

               ;-- Ici vous pouvez mettre en place toutes les données,
                                          ;-- le buffer et les variables
                                                                  ;...
                                                                  ;...
                                                                  ;...

data      ends                            ;Fin du segment de données

;-- Code ------------------------------------------------------------

code      segment para 'CODE'             ;Définition du segment de code

          assume cs:code, ds:data, ss:stackseg

                         ;CS concerne le segment de code, DS le segment
                              ;de données et SS le segment de pile.
                                        ;ES peut être utilisé
                                                  ;librement.
prog      proc far              ;Cette procédure représente le programme
                         ;principal à proprement parler. Elle est exécutée
                                  ;après le lancement du programme.

                  ;-- A ce niveau, CS et SS sont déjà initialisés. ---
          ;-- Il ne reste plus qu'à initialiser DS à la main, parce qu'il
                                        ;-- désigne le PSP tout comme ES.
          mov  ax,data        ;Charger l'adresse de segment du segment de
          mov  ds,ax                    ;données dans le registre DS

          call setfree                ;Libérer la mémoire non nécessaire

          ;-- Ici vous pouvezclure le code programme du programme-----
                                                          ;-- principal
                                                                  ;...

                                                                  ;...
                                                                  ;...

       ;--- Terminer le programme à l'aide de la fonction 4Ch ---------

          mov  ax,4C00h  ;Charger le numéro de fonction, code d'erreur 0
          int  21h                                        ;Appels DOS

       ;--- Ici l'exécution du programme n'a plus aucun sens parce que --
                              ;--- l'appel DOS précédent a mis fin au programme

prog      endp                                ;Fin de la procédure PROG

;-- Sous-routines ---------------------------------------------------
;-- Cette place est réservée aux sous-routines du programme

a_proc    proc near
                                                                  ;...
                                                                  ;...
                                                                  ;...
          ret

a_proc    endp

b_proc    proc near
                                                                  ;...
                                                                  ;...
                                                                  ;...
          ret

b_proc    endp

;-- SETFREE: Libérer la place mémoire non occupée ----------------
;-- Entrée  : ES = Adresse du PSP
;-- Sortie  : Aucune
;-- Register : AX, BX, CL et les FLAGS sont modifiés
;-- Infos   : Comme le segment de pile est toujours le dernier segment
;             d'un  fichier EXE, ES:0000 désigne le début du programme
;             dans la mémoire et SS:SP la fin. Cela permet donc de
;             calculer la longueur du programme
setfree   proc near

          mov  bx,ss      ;Calculer l'intervalle entre le PSP et la pile
          mov  ax,es                                  ;en paragraphes
          sub  bx,ax

          mov  ax,sp                    ;Calculer l'état de la pile en
          add  ax,15                                  ;en paragraphes
          mov  cl,4
          shr  ax,cl                    ;Contenu, longueur de la pile

          add  bx,ax                        ;Ajouter les deux valeurs

          mov  ah,4ah                   ;Seule cette mémoire doit rester
          int  21h                                        ;réservée

          ret                               ;Retour au programme d'appel

setfree   endp

;-- Fin -------------------------------------------------------------

code      ends                                    ;Fin du segment de code
          end  prog                         ;Commencer l'exécution avec la
                                                    ;procédure PROG
```

Lorsque vous développez un programme EXE, vous devez naturellement le faire assembler comme un programme normal, à l'aide du macro-assembleur. Vous devez ensuite le linker à l'aide du programme LINK. Si votre programme ne comporte aucune erreur, le programme LINK produira alors un fichier EXE.

Voici la marche à suivre pour un programme assembleur appelé TEST.ASM :

```
masm test;
link test;
```

Si vous utilisez Turbo Assembler TASM, lancez les commandes suivantes :

```
tasm test
tlink test
```

Si toutes ces opérations se sont effectuées sans difficulté, vous pouvez lancer le programme TEST.EXE à partir du niveau du DOS en entrant simplement le nom TEST.

17.4. Le PSP

Nous allons terminer ce chapitre en parlant du Program Segment Prefix (PSP) que DOS place devant chaque programme EXE ou COM en mémoire.

C'est un vestige des premières versions du DOS et il présente de ce fait de grandes similitudes avec les structures de données que CP/M place en mémoire devant les programmes à exécuter.

Le tableau suivant décrit la structure et les champs du PSP, plusieurs d'entre-eux restent non documentés (ou réservés) par Microsoft. La plupart de ces champs ont été décodés, même s'ils ne sont pas très utiles en programmation.

Structure du PSP		
Adresse	**Contenu**	**Type**
00H	Appel de l'interruption 20H	2 BYTE
02H	Adresse de segment de mémoire allouée pour le programme	1 WORD
04H	Réservé	1 BYTE
05H	Appel de l'interruption 21H	5 BYTE
0AH	Copie du vecteur d'interruption 23H	2 WORD
0EH	Copie du vecteur d'interruption 23H	2 WORD
12H	Copie du vecteur d'interruption 24H	2 WORD
16H	Réservé	22 BYTE
2CH	Adresse de segment du bloc d'environnement	1 WORD
2EH	Réservé	46 BYTE
5CH	FCB 1	16 BYTE
6CH	FCB 2	16 BYTE
80H	Nombre de caractères dans la ligne de commande	1 BYTE
81H	Ligne de commande (CR-LF)	127 BYTE

Le PSP lui-même a toujours une longueur de 256 octets et il contient des informations importantes aussi bien pour le DOS que pour le programme à exécuter. A la cellule de mémoire 00h figure un appel de la fonction du DOS qui sert à terminer un programme. Elle sert à libérer la mémoire de programme et à rendre le contrôle à l'interpréteur de commandes ou du moins au programme d'appel quel qu'il soit. Un autre appel de fonction de l'interruption 21h figure à la cellule de mémoire 05h du PSP. Il ne s'agit cependant pas, dans ce cas, de l'instruction INT 21h mais d'une instruction FAR CALL, qui saute directement au code de programme résident du DOS. Seules les fonctions jusqu'à 24h peuvent cependant être appelées de cette façon et le numéro de fonction n'est pas transmis dans le registre AL, comme il est habituel avec INT 21h, mais dans le registre CL. Ces deux appels de fonction ne sont toutefois plus utilisés aujourd'hui car ils ne permettent pas d'appeler toutes les fonctions DOS et parce que la définition des registres lors de l'appel des fonctions DOS suivant cette méthode diffère des conventions habituelles.

A la cellule de mémoire 02h du PSP figure un mot (16 bits) qui indique la fin de la RAM réservée pour le programme. Si un programme a besoin de mémoire supplémentaire, il peut déterminer à l'aide de cette valeur s'il y a encore de la place entre sa propre fin et la fin de la mémoire RAM réservée. Si c'est le cas, il peut utiliser cette mémoire pour ses problèmes propres, sinon il doit réclamer de la mémoire RAM à travers une fonction du DOS. Les cellules de mémoire 0Ah, 0Eh et 12h stockent le contenu de trois vecteurs d'interruption. Il s'agit des vecteurs d'interruption pour terminer un programme, pour réagir lorsque la touche Control-C ou Control-Break a été actionnée et pour réagir à certaines erreurs. Si le programme modifie l'un de ces vecteurs au cours de son exécution, le DOS pourra inscrire à nouveau l'adresse originelle du vecteur dans la cellule de mémoire qui convient, en lisant les anciennes valeurs contenues dans le PSP.

L'adresse de segment du bloc d'environnement est stockée sous forme de mot dans la cellule de mémoire 2Ch. L'adresse d'offset du bloc d'environnement est toujours de 0 par rapport à l'adresse de segment. Le bloc d'environnement est une succession de chaînes ASCII dont chacune est terminée par une marque de fin (code ASCII. Ainsi sont délimitées différentes chaînes de caractères contenant différentes informations telles que par exemple le chemin de recherche actuel et le répertoire dans lequel figure l'interpréteur de commandes, dans le fichier COMMAND.COM.

Par souci de compatibilité avec CP/M, le PSP contient deux blocs de contrôle de fichier (file control block) respectivement à partir des cellules de mémoire 5Ch et 6Ch. Ces blocs contiennent les deux premiers paramètres qui ont été indiqués en plus du nom du programme lors de l'appel de ce dernier. Il est systématiquement supposé qu'il s'agit d'un nom de fichier. Même lorsque cela est vérifié, seul peut toutefois être stocké le nom de fichier ainsi que le lecteur (A, B etc...) mais pas le chemin car les FCB ne soutiennent pas le système de fichiers hiérarchisé. Ces deux FCB sont d'ailleurs en fait sans importance pour la plupart des programmes car la compatibilité avec CP/M ne joue plus aucun rôle pour le développement des programmes alors que, d'autre part, un programme peut difficilement se dispenser de soutenir le système de fichiers hiérarchisé.

Tous les paramètres qui ont été indiqués lors de l'appel du programme à la suite du nom du programme (y compris donc ceux qui sont inscrits par le DOS dans les deux FCB) sont stockés dans le PSP à partir de la cellule de mémoire 81h. Les paramètres qui servent au détournement de l'entrée et de la sortie ne sont cependant pas pris en compte à cet égard car le détournement de l'entrée/sortie est de la responsabilité du système d'exploitation et il reste invisible pour le programme. Les paramètres ne remplissent toutefois pas systématiquement la totalité de la mémoire restant disponible entre l'adresse 81h et la fin du PSP à l'adresse 0FFh. C'est pourquoi le nombre de caractères dans le buffer de paramètres est inscrit dans la cellule de mémoire 80h, sans prendre en compte cependant le retour de chariot (code ASCII 13).

Cette zone (le buffer de paramètres) est cependant également utilisée par le DOS comme zone de transfert de données disque (en anglais DTA pour Disk Transfer Area). Tant que le programme ne reloge pas la DTA dans une autre zone de mémoire à l'aide d'une fonction DOS, le DOS utilisera cette zone comme buffer de données lors de tout accès de fichier soutenu par FCB.

18. Entrée et sortie de caractères

C'est en appelant les fonctions DOS pour l'entrée et la sortie de caractères que la plupart des programmeurs font leurs premières expériences dans l'emploi des différentes fonctions du DOS. C'est pourquoi nous avons souhaité aborder la description de cette fonction avant celle des autres fonctions telles que la manipulation des fichiers, de la disquette ou du disque dur.

Les différentes fonctions du DOS pour l'entrée et la sortie de caractères permettent de commander des périphériques tels que le clavier, l'écran, l'imprimante et l'interface série. Ces fonctions se divisent en deux groupes, le premier groupe comprenant les fonctions équivalant au système d'exploitation CP/M et le seconde groupe comprenant les fonctions compa-tibles avec le système d'exploitation UNIX. Les premières constituent donc ce qu'on appellera les fonctions traditionnelles d'entrée et de sortie de caractères. Elles comprennent cependant quelques fonctions supplémentaires en dehors des fonctions dont dispose déjà CP/M. Elles facilitent en tout cas la transposition de programmes CP/M dans l'environnement DOS.

Les fonctions compatibles avec UNIX sont appelées fonctions Handle car un code numérique (appelé "handle" en anglais) doit être transmis lorsqu'elles sont appelées. Le DOS associe à chaque Handle un périphérique déterminé qu'il appellera lors de l'entrée/sortie.

Tout comme dans d'autres domaines tels que la gestion de fichiers, les fonctions CP/M jouent le rôle d'un rescapé d'une époque révolue et continuent de survivre pour assurer la compatibilité avec les anciennes versions de DOS. Ces fonctions ne servent pratiquement à rien jusqu'à la fonction 09h qui est parfois utilisée dans les utilitaires en Assembleur pour afficher les messages à l'écran.

De même, les fonctions Handle interviennent très peu dans les programmes professionnels parce que la majorité des applications DOS fonctionnent aujourd'hui en mode interactif, c'est-à-dire avec des fenêtres, des menus et des boîtes de dialogue. Les fonctions DOS ne sont toutefois pas capables de gérer l'écran de cette manière car elles le considèrent comme un simple terminal. Par conséquent, elles ne permettent pas de positionner le curseur ou définir une couleur de sortie.

En dehors de ces arguments qui ne s'appliquent qu'aux diverses fonctions liées à la sortie sur l'écran, il existe d'autres raisons valables allant à l'encontre des fonctions DOS effectuant des sorties sur l'imprimante ou l'interface série. En effet, ces deux périphériques peuvent parfaitement être contrôlés grâce à des fonctions BIOS prévues à cet effet, à condition qu'il soit possible de programmer directement l'électronique concernée.

La seule et unique possibilité de détourner l'entrée et la sortie des programmes lors de leur appel parle encore en faveur des fonctions DOS car les fonctions BIOS ou la programmation directe de l'électronique ne sont pas encore en mesure d'assurer

parfaitement cette routine. Les filtres, décrits à la fin de ce chapitre, constituent un débouché important pour les fonctions DOS chargées d'entrer et sortir les caractères.

18.1. Fonctions Handle

Comme nous le verrons dans un des chapitres suivants, les fonctions Handle ne servent pas seulement à l'entrée et à la sortie de caractères sur un périphérique mais aussi à l'accès aux fichiers. Le DOS reconnaît la différence en examinant le nom attribué à un Handle. S'il s'agit d'un nom de périphérique, il appellera le périphérique correspondant, sinon il considérera que l'accès doit se faire sur un fichier. Les noms de périphériques définis sont :

```
CON             Clavier et écran
AUX             Interface série
COM1, COM2      Interfaces série
PRN             Imprimante
LPT1,LPT2,LPT3  Interfaces imprimante (parallèles)
NUL             Périphérique fictif, rien ne se passe
                lorsqu'il est appelé
```

Avec les périphériques AUX, COM1 et COM2, l'entrée comme la sortie sont envoyées sur l'interface série correspondante. Avec CON, c'est naturellement la sortie qui se fera sur l'écran alors que les entrées seront reçues du clavier.

Sachant que sur les interfaces parallèles, il est uniquement possible d'effectuer une sortie, les périphériques PRN, LPT1, LPT2 et LPT3 assurent exclusivement la sortie. Le même phénomène s'applique au périphérique NUL qui est une sortie de trou noir dans lequel les sorties disparaissent à jamais.

Les Handles standard

Lorsque le DOS donne le contrôle à un programme, il définit d'avance cinq handles à travers lesquels il sera possible d'accéder aux différents périphériques. Ces handles portent les numéros 0 à 4 et ils correspondent aux périphériques suivants :

0	Périphérique d'entrée standard (CON)
1	Périphérique de sortie standard (CON)
2	Périphérique standard de sortie des messages d'erreur (CON)
3	Interface série (AUX)
4	Imprimante standard (PRN)

Voici maintenant un petit exemple pour illustrer la signification de cette table :

Si un programme veut recevoir des entrées effectuées par l'utilisateur, il indiquera le handle 0 pour appeler les fonctions Handle car c'est à travers ce handle que le périphérique d'entrée standard peut être appelé. Il s'agira normalement du clavier et le programme pourra ainsi recevoir les entrées effectuées au clavier par l'utilisateur. Mais un programme ne peut jamais être entièrement sûr de cette action puisque l'utilisateur peut rediriger l'entrée et la sortie à l'aide des symboles et .

Par exemple, les fonctions BIOS ne se laissent nullement influencer par ce détournement de l'accès à l'écran et au clavier même s'il est effectué au niveau du DOS à travers le détournement des handles standard. Ainsi, les données lues sur le périphérique d'entrée standard lors de l'accès peuvent provenir du clavier ou d'un fichier. Ce détournement demeure cependant imperciptible pour le programme. Il en est de même pour le périphérique de sortie standard avec le handle 1. Toutefois, l'utilisateur n'a pas le droit de rediriger les autres handles standard.

Avant d'en venir maintenant à une description des différents périphériques, nous voudrions vous présenter plusieurs fonctions qui permettent d'accéder à tous les périphériques.

Fonction	Tâche
3Dh	Ouverture d'un handle
3Eh	Fermeture d'un handle
3Fh	Lecture
40h	Ecriture

Sortie de caractères

Pour envoyer des données sur un périphérique, on utilise la fonction 40h de l'interruption 21h. Elle attend le numéro de fonction (40h) dans le registre AH et le handle dans le registre BX. Pour sortir par exemple un message d'erreur, on indiquerait la valeur 2 comme handle, pour appeler le périphérique chargé de la sortie des messages d'erreur (notez d'ailleurs que ce périphérique ne peut être détourné, le handle 2 appelle donc toujours l'écran). Avant d'appeler la fonction, il faut charger dans le registre CX le nombre de caractères à transmettre. Ces caractères doivent figurer de façon séquentielle dans un buffer dont l'adresse de segment est inscrite dans la paire de registres DS:DX sous forme d'un pointeur FAR.

Après appel de la fonction, le flag Carry indique si les données ont pu être transmises sans problème. Si le flag Carry est annulé, c'est que la transmission a été réussie et le nombre de caractères transmis est inscrit dans le registre AX. Si la valeur de ce registre ne correspond pas au nombre d'octets à transmettre, cela indique au programmeur que

les données étaient censées être transférées dans un fichier mais qu'il n'y avait plus de place disponible sur cette mémoire de masse (disquette ou disque dur). Si le flag Carry est mis après appel de la fonction, cela signifie par contre que les données n'ont pu être transmises et un code d'erreur figure alors dans le registre AX. Un code d'erreur de 5 dans ce registre signale que l'accès au périphérique a été refusé alors que la valeur 6 indique que le handle spécifié n'était pas ouvert. Reportez-vous à la description de la fonction DOS 59h en annexe pour obtenir les autres codes d'erreur.

Entrée de caractères

La fonction 3Fh, symétrique à la fonction 40h, permet de lire des données sur un périphérique. Le rôle des registres est le même que pour la fonction 40h : le registre AH doit recevoir le numéro de fonction, le registre BX le handle, le registre CX le nombre de caractères à lire et enfin la paire de registres DS:DX l'adresse du buffer dans lequel les caractères lus doivent être transférés.

Après appel de cette fonction, le flag Carry signale également si une erreur est apparue, le registre AX recevant toujours le code d'erreur s'il y a lieu. Les codes d'erreur possibles, 5 et 6, ont la même signification que les codes d'erreur pour la fonction 40h. Si cependant le flag Carry est annulé, cela signale que la fonction a pu être exécutée correctement. Le registre AX contient alors le nombre de caractères lus figurant dans le buffer. Un 0 dans ce registre indique que les données à lire devaient venir d'un fichier mais que toutes les données de ce fichier avaient déjà été lues.

Ouverture de fichiers et périphériques

Comme nous l'avons déjà expliqué, lors de l'accès aux différents périphériques, il est possible que l'utilisateur ait opéré une détournement de l'entrée ou de la sortie, auquel cas les entrées que le programme attend du clavier viendraient en réalité d'un fichier. Si l'on veut éviter le risque d'un tel détournement, on peut ouvrir un nouveau handle pour s'assurer que la communication se fera bien avec le périphérique avec lequel on veut réellement échanger des données.

On appelle à cet effet la fonction 3Dh de l'interruption 21h. Elle attend le numéro de fonction (3Dh) dans le registre AH et le mode d'accès dans le registre AL. Le mode d'accès définit si des données doivent être envoyées au périphérique ou bien au contraire reçues de ce périphérique. 0 signifiera ici lecture, 1 écriture et la valeur 2 permettra la lecture et l'écriture sur le périphérique. Le nom du périphérique appelé est placé dans un buffer dont l'adresse de segment doit être inscrite dans la paire de registres DS:DX sous forme d'un pointeur FAR. Pour que le DOS puisse identifier correctement le nom du périphérique (CON, PRN etc...), il doit être entièrement écrit en majuscules et le dernier caractère du nom de périphérique doit être suivi dans la mémoire d'une marque de fin (le code ASCII 0).

Le flag Carry indique après appel de la fonction si l'ouverture du handle a été réussie. Si ce flag est mis, ce n'est pas le cas et le registre AX contient un code d'erreur. Si le flag Carry est par contre annulé, le registre AX contient le nouveau handle, à travers lequel on pourra accéder au périphérique lors de toutes les opérations suivantes. Reportez-vous à la description de la fonction DOS 59h en annexe pour obtenir les autres codes d'erreur.

Fermeture de fichiers et périphériques

Si plus aucun accès à ce périphérique n'est effectué à travers ce handle (ce qui arrive au plus tard à la fin du programme), il convient de refermer le handle à l'aide de la fonction 3Eh de l'interruption 21h. Si le programmeur oublie de refermer les handles qu'il a ouvert, il peut arriver qu'il y ait tellement de handles ouverts et non refermés sur le PC qu'il n'y ait plus d'autres handles disponibles. C'est pourquoi tout handle ouvert doit être refermé dès qu'il n'est plus utilisé. Il faut charger à cet effet la valeur 3Eh (le numéro de fonction) dans le registre AH et le handle à refermer dans le registre BX.

Si le handle a pu être refermé sans problème, le flag Carry sera annulé après appel de l'interruption. Si ce flag est mis, c'est que le handle n'aura pu être refermé soit parce qu'il avait déjà été refermé auparavant, soit parce qu'il n'existait pas.

Cette fonction permet naturellement également de refermer les handles prédéfinis 0 à 4. Il convient cependant d'être particulièrement prudent à cet égard. Si vous fermez en effet le handle 0 (périphérique d'entrée/ sortie standard), il ne sera par exemple plus possible de recevoir les caractères entrés au clavier.

Examinons les particularités des différents périphériques :

Le périphérique d'entrée standard

Aucune opération d'écriture n'est naturellement possible sur le clavier qui admet seulement les opérations de lecture. Le résultat de la fonction de lecture dépend étroitement du mode sous lequel ce périphérique est appelé. Le DOS distingue ici les modes raw et cooked. En mode cooked, tout caractère envoyé à un périphérique ou reçu d'un périphérique est examiné pour voir s'il s'agit d'un caractère de commande spécial. Si c'est le cas, une opération déterminée sera exécutée. En mode Raw au contraire, les différents caractères ne sont pas examinés et ne subissent donc aucune manipulation au départ. Le DOS exploite a priori tous les périphériques d'entrée et sortie de caractères en mode cooked mais il est possible de passer en mode Raw. Nous vous indiquerons plus loin comme procéder pour ce faire.

L'exemple du clavier permet d'illustrer très clairement la différence entre les modes cooked et raw :

Nous allons donc supposer que nous voulions lire 30 caractères sur le clavier en mode cooked. Pour l'entrée des caractères, le DOS autorise l'édition de l'entrée à l'aide de quelques touches de commande, comme Backspace par exemple, et il stocke les entrées tout d'abord dans un buffer interne. On tiendra également compte pendant la saisier de l'entrée éventuelle de <CTRL>-<C> ou <CTRL>-<Break>, auquel cas la saisie sera interrompue.

Dans ce mode, les combinaisons de touches <CTRL>-<P> et <CTRL>-<S> acquièrent également une signification particulière. Lorsque <CTRL>-<S> est actionnée, le programme est suspendu jusqu'à ce qu'une nouvelle touche soit actionnée. Après entrée de <CTRL>-<P>, la sortie est dérivée de l'écran vers l'imprimante (PRN). Ce détournement n'est annulé que si la touche <CTRL>-<P> est à nouveau actionnée. Lorsque la touche <Return> est actionnée, les 30 premiers caractères ou au moins tous les caractères entrés (s'il y en a eu moins de 30) sont copiés du buffer DOS dans le buffer d'entrée du programme. Les caractères de commande utilisés pour l'édition ne sont cependant pas transmis dans ce cas.

En mode Raw au contraire, tous les caractères entrés, y compris les caractères de commande sont transmis au programme appelant la fonction d'entrée. La touche <Return> ne sera pas non plus traitée différemment et le contrôle sera de toute façon rendu au programme d'appel après l'entrée de 30 caractères même si la touche <Return> a été actionnée dès le second caractère.

Ecran

Pour sortir des caractères sur l'écran, on appelle généralement le handle 1 pour le périphérique de sortie standard. Comme ce périphérique peut cependant être redirigé, les sorties effectuées à travers ce handle risquent de ne pas apparaître obligatoirement sur l'écran. Le périphérique d'erreur standard (handle 2) ne peut toutefois, quant à lui, pas être détourné et les sorties à travers ce handle apparaîtront donc toujours sur l'écran. C'est pour cette raison que ce handle sera à recommander chaque fois qu'il s'agira de sortir des caractères sur l'écran et uniquement sur l'écran.

L'écran est lui aussi exploité en principe en mode Cooked. Lorsqu'un caractère est sorti sur l'écran, les touches <Control>-<C> ou <Control>-<Break> sont donc testées. Ce simple test ralentit cependant tellement la sortie sur l'écran qu'il est parfois préférable de passer en mode raw. Nous vous présentons à la fin de ce chapitre un programme permettant de le faire.

Interface parallèle

Contrairement au clavier et à l'écran, la sortie sur l'imprimante ne peut être détournée (du moins pas à partir du niveau utilisateur). La seule exception réside dans la possibilité

de détourner la sortie d'une imprimante parallèle vers une imprimante série avec la commande DOS MODE. Il est toutefois possible de faire entreposer les caractères à sortir dans un fichier, avant qu'ils ne soient envoyés à l'imprimante ou de les faire retransmettre à une autre imprimante à l'intérieur d'un réseau. Il s'agit cependant dans ce cas d'un travail qui concerne le DOS et qui n'est pas perçu par le programme.

Pour la sortie sur imprimante, le programmeur dispose du handle 4 qui lui permet d'appeler l'imprimante standard. S'il s'agit cependant d'appeler une imprimante bien précise parmi les trois qui peuvent être connectées sur le PC, il est nécessaire d'ouvrir un handle pour accéder à cette imprimante. L'imprimante portant le numéro 1 s'appelle LPT1, la numéro 2 LPT2 et la numéro 3 LPT3. Le DOS considère que l'imprimante LPT1 et le périphérique DOS ne font qu'un. Lorsque la sortie est envoyée sur PRN, elle s'effectue sur l'imprimante LPT1. Lorsqu'on ouvre le handle, on indique comme nom de périphérique LPT1, LPT2 ou LPT3.

Interface série

Bon nombre des observations faites sur l'imprimante s'appliquent également à l'interface série. C'est ainsi, par exemple, que l'entrée et sortie sur l'interface série ne peuvent être détournées en aucun cas, même pas d'une imprimante série vers une imprimante parallèle. Le programmeur peut utiliser pour accéder à l'interface série le handle 3 qui permet d'appeler l'interface série standard (AUX).

Le PC peut comporter plusieurs interfaces série. Seules les deux premières seront cependant reconnues par le DOS. Elles portent les noms COM1 et COM2. Comme on ne sait jamais très précisément laquelle des deux interfaces est appelée lorsqu'on accède au périphérique AUX, il est conseillé d'ouvrir un nouveau handle pour accéder à une interface particulière.

La littérature technique consacrée au système DOS relève une erreur lors de l'exécution des opérations en lecture sur l'interface série en mode Cooked. Le nombre de caractères lus renvoyé dans le registre AX après une opération en lecture diffère en effet du nombre de caractères effectivement lus. C'est pourquoi il est conseillé d'exploiter l'interface série en mode raw, même si ce mode ne permet pas de détecter les différents caractères de commande tels que <Control>-<C> et la fin de fichier.

18.2. Fonctions traditionnelles

Les fonctions traditionnelles d'entrée et de sortie de caractères reposent sur un principe radicalement différent de celui des fonctions handle. Lorsque ces fonctions sont appelées, aucun handle n'est transmis car chaque fonction concerne un périphérique bien précis. Il n'existe donc pas de fonctions de lecture et écriture de base permettant d'accéder à tous les périphériques mais plutôt des fonctions diverses pour un périphé-

rique donné. A partir de la version 2.0, le détournement de l'entrée et de la sortie peut également être réalisé à travers les fonctions de caractères traditionnelles mais elle reste également dissimulée au programmeur.

Nous allons donc maintenant vous présenter les fonctions traditionnelles de communication avec les différents périphériques d'entrée et de sortie, en décrivant le mode d'appel et les particularités de ces fonctions.

Fonction	Tâche
01h	Entrée de caractères sur l'écran avec Echo
02h	Sortie de caractères sans Echo
03h	Lecture de caractères sur interface série
04h	Sortie de caractères sur interface série
05h	Sortie de caractères sur imprimante
06h	Entrée/Sortie de caractères directe
07h	Entrée directe de caractères sans Echo
08h	Entrée de caractères sans Echo
09h	Sortie d'une chaîne de caractères
0Ah	Entrée d'une chaîne de caractères
0Bh	Lecture de l'état d'entrée
0Ch	Suppression du buffer d'entrée et appel de la fonction d'entrée/sortie

Clavier

Les sept fonctions qu'offre le DOS pour communiquer avec le clavier se distinguent par la façon dont elles réagissent lorsque les touches <Control>-<C> ou <Control>-<Break> sont actionnées ainsi que par le fait qu'elles sortent ou non sur l'écran les caractères entrés.

Lorsqu'est actionnée une touche représentée par un code clavier étendu, les fonctions clavier renvoient dans un premier temps le code 0 pour signaler à l'appelant qu'il s'agit d'un code étendu. C'est le prochain appel d'une fonction clavier qui fournira alors l'extension du code clavier elle-même.

Les codes clavier étendus sont par exemple représentés par les touches du curseur, mais aussi <Insert> et <Del> parce que ces touches ne sont plus utilisées dans le jeu de caractères ASCII du PC.

Pour lire les caractères un à un sur le clavier, le programmeur dispose des fonctions 01h, 06h, 07h et 08h de l'interruption 21h. Toutes les fonctions attendent dans le registre AH le numéro de la fonction et le caractère figure dans le registre AL après appel de la fonction. Contrairement à la fonction 06h, les fonctions 01h, 07h et 08h attendent qu'un caractère soit entré si aucun caractère ne figure encore dans le buffer clavier. La fonction 01h sort le caractère entré sur l'écran (ou plus précisément sur le périphérique de sortie standard) contrairement aux fonctions 07h et 08h.

La fonction 06h se distingue des fonctions présentées précédemment par le fait qu'elle sert également à la sortie de caractères. Pour lui indiquer qu'il s'agit de lire un caractère, il faut donc lui transmettre la valeur 255 dans le registre DL, en plus du numéro de fonction. Elle n'attend pas qu'un caractère soit entré mais rend au contraire immédiatement le contrôle au programme d'appel. Lorsqu'elle renvoie un caractère (qu'elle ne sort pas sur l'écran), elle le signale en annulant le flag zéro. Si ce flag est par contre mis après appel de la fonction, c'est qu'aucun caractère n'a pu être lu.

En appelant la fonction 0Bh, un programme peut déterminer si un ou plusieurs caractères figurent dans le buffer clavier avant même d'appeler une autre fonction de lecture de caractères. Après l'appel de fonction, le contenu du registre AL indique si des caractères sont disponibles. Si ce registre contient la valeur 0, aucun caractère n'est disponible, alors que la valeur 255 signale qu'un ou plusieurs caractères sont en attente dans le buffer clavier.

S'il est nécessaire de vider le buffer clavier avant d'appeler une fonction de lecture de caractères, on peut se servir de la fonction 0Ch. Celle-ci vide le buffer clavier et appelle ensuite une des autres fonctions de lecture de caractères dont le numéro de fonction doit être transmis à travers le registre AL lors de l'appel de la fonction 0Ch.

En dehors des fonctions que nous vous avons présentées jusqu'ici, et qui servent uniquement à l'entrée d'un caractère, le DOS offre également une fonction qui permet de lire une série de caractères. Elle porte le numéro de fonction 0Ah. Elle attend dans le registre AH le numéro de fonction ainsi que l'adresse d'un buffer dans la paire de registres DS:DX. Ce buffer recevra les caractères entrés mais il contiendra également dans ses deux premiers octets des informations sur le buffer lui-même. Le premier octet du buffer indique le nombre maximum de caractères que le DOS devra lire et qui devront être stockés à partir de la cellule de mémoire numéro deux dans le buffer. Cet octet doit bien sûr être fixé par le programmeur avant appel de la fonction.

Le DOS lit alors au maximum le nombre de caractères indiqué mais termine l'entrée auparavant si la touche <Return> a été actionnée. L'entrée peut être éditée avec les touches d'édition habituelles. Le nombre de caractères effectivement lus (y compris le caractère Return) est alors inscrit par le DOS dans la cellule de mémoire 1 du buffer. Les codes clavier étendus occupent 2 octets dans le buffer, le premier octet (toujours le code ASCII 0) indiquant qu'un code clavier étendu suit. Cela signifie par conséquent que si vous voulez lire 10 caractères, vous ne pourrez faire entrer plus de 5 codes étendus. C'est pourquoi il convient de prévoir une taille de buffer suffisamment grande pour que le nombre voulu de caractères puissent être lus même s'il s'agit de caractères étendus.

Structure du buffer lors de l'utilisation de la fonction 0Ah

Cela contraint par exemple le buffer à prévoir 20 octets lors de l'entrée de 10 caractères parce que, dans le pire des cas, il s'agit de codes clavier étendus occupant deux octets. Si vous ne respectez pas cette règle, il se peut que DOS écrive à l'extérieur du buffer en considérant que sa taille est suffisante pour recevoir également les codes clavier étendus. Cela porte préjudice à la bonne exécution du programme puisque les variables stockées en mémoire immédiatement après le buffer sont écrasées.

La table suivante vous indique quelles sont, parmi les sept fonctions présentées, celles qui réagissent lorsque les touches <Control>-<C> ou <Control>-<Break> sont actionnées. Cette table vous fournit un rapide aperçu des différentes fonctions d'entrée de caractères.

Numéro	Fonction	Control	Echo
Numéro	Fonction	Control-C	Echo
01h	Entrée de caractères	oui	oui
06h	Entrée de caractères directe	non	non
07h	Entrée de caractères	non	non
08h	Entrée de caractères	oui	non
0Ah	Entrée de chaînes de caractères	oui	non
0Bh	Lire état d'entrée	oui	non
0Ch	Reset buffer d'entrée puis entrée	variable	variable

L'écran

Le DOS offre trois fonctions de sortie de caractères ou de textes sur l'écran ou sur le périphérique de sortie standard. Deux servent à la sortie d'un caractère alors que la troisième permet de sortir une chaîne de caractères. Elle porte le numéro de fonction 09h et contrairement aux fonctions du BIOS pour la sortie de chaînes de caractères, ce n'est pas le caractère de fin (code ASCII 0) mais le caractère dollar (code ASCII 36) qu'elle attend comme marque de fin à la fin de la chaîne de caractères à sortir. Le numéro de fonction doit naturellement, ici aussi, être transmis dans le registre AH. L'adresse de la chaîne de caractères à sortir doit cependant être placée dans la paire de registres DS:DX sous forme de pointeur FAR sur le buffer correspondant.

La table suivante indique la signification des différents codes de commande qui peuvent être contenus dans la chaîne de caractères :

7	"Bell", déclenche un bip
8	"Backspace", efface le caractère précédent et ramène le curseur un caractère en arrière
10	"Line Feed", (LF) déplace le curseur une ligne vers le bas
13	"Carriage Return", (CR) amène le curseur sur le début de la ligne actuelle

Comme pour la fonction 02h, les touches <Control>-<C> ou <Control>-<Break> sont aussi testées pendant la sortie des différents caractères.

La fonction 02h est le pendant de la fonction 09h, si ce n'est toutefois qu'elle ne peut recevoir qu'un seul caractère à sortir (dans le registre DL) lors de chaque appel.

La seconde fonction de sortie d'un caractère est la fonction 06h que nous avons déjà rencontrée à propos de l'entrée de caractères. Elle attend également dans le registre DL le caractère à sortir. Le caractère de code ASCII 255 ne peut toutefois pas être sorti car la fonction 06h interprète la valeur 255, dans le registre DL, comme une demande d'entrée d'un caractère. Elle travaille cependant pour la sortie de caractères un peu plus rapidement que sa collègue, la fonction 02h, car elle ne teste pas après chaque caractère sorti si les touches <Control>-<C> ou <Control>-<Break> ont été actionnées.

L'imprimante

Les fonctions traditionnelles de sortie de caractères sur l'imprimante se limitent à la fonction 05h qui permet d'envoyer un caractère à la fois à l'imprimante ou l'interface parallèle. On ne sait malheureusement pas très précisément, lorsqu'on appelle cette fonction, à quelle imprimante le caractère à sortir sera communiqué car le caractère sera envoyé de façon très générale à l'imprimante standard (correspondant au périphérique PRN ou LPT1), qui peut être n'importe laquelle des trois imprimantes parallèles possibles ou même l'interface série.

Si l'imprimante standard est occupée lorsque cette fonction est appelée, la fonction attendra pour sortir le caractère que l'imprimante soit à nouveau disponible. Ce n'est qu'ensuite que le contrôle sera rendu au programme d'appel. Pendant ce laps de temps, la fonction réagira si les touches <Control>-<C> ou <Control>-<Break> sont actionnées.

Si vous voulez recourir à cette fonction malgré les difficultés évoquées, vous devez charger le numéro de fonction dans le registre AH, comme d'habitude, le code ASCII du caractère à sortir dans le registre DL et appeler alors l'interruption 21h. La fonction ne renvoie malheureusement pas comme résultat l'état de l'imprimante. Il n'y a donc aucun moyen d'obtenir cet état sous le DOS.

C'est pourquoi il est conseillé, à titre exceptionnel, de préférer une fonction du BIOS à la fonction du DOS, pour sortir des caractères sur l'imprimante. Les fonctions du BIOS

permettent en effet de définir précisément à quelle imprimante le caractère doit être envoyé mais aussi de connaître l'état de l'imprimante appelée après chaque sortie sur l'imprimante.

Interface série

Pour communiquer avec l'interface série, le DOS offre deux fonctions de base de lecture et écriture. Comme pour l'imprimante, le problème est toutefois ici également que les données sont systématiquement envoyées à l'interface standard et il n'est pas possible de savoir exactement s'il s'agit de COM1 ou de COM2. Les deux fonctions réagissent si les touches <Control>-<C> ou <Control>-<Break> sont actionnées. Elles ne fournissent malheureusement pas non plus l'état de l'interface série après avoir été appelées. Il n'existe donc aucun moyen sous le DOS de tester cet état pour détecter d'éventuelles erreurs de transmission.

Pour recevoir un caractère de l'interface série, on utilise la fonction 03h qui renvoie le caractère reçu dans le registre AL. Le DOS ne place malheureusement pas dans un buffer les données reçues de l'interface série. Il peut donc arriver que l'interface série reçoive des données plus vite qu'elles ne peuvent être lues avec la fonction 03h, ce qui entraîne bien sûr une perte de données. C'est pourquoi, utilisez de préférence les diverses fonctions du BIOS pour l'accès à l'interface série.

Pour envoyer des données à l'interface série, on peut avoir recours à la fonction 04h qui envoie à l'interface série le caractère dont le code ASCII lui est transmis dans le registre DL lorsqu'elle est appelée. Si l'interface série est occupée, la fonction attendra jusqu'à ce qu'elle soit de nouveau disponible et transférera alors seulement le caractère à sortir.

18.3. Basculer du mode Raw vers le mode Cooked

En décrivant les fonctions d'entrée au clavier et de sortie sur l'écran, nous avons évoqué la possibilité de passer du mode cooked au mode raw sur ces périphériques. Les programmes que voici maintenant se chargent de cette commutation. Ils sont formulés en Pascal et C. Ils utilisent la fonction appelée IOCTL qui permet d'accéder au drivers de périphériques du DOS. Il s'agit de routines qui servent d'interface entre les fonctions DOS d'entrée et de sortie et l'électronique. Les différents programmes indiquent au driver de périphérique chargé du clavier et de l'écran (il s'appelle CON), à travers cette fonction IOCTL, s'il doit travailler en mode Cooked ou en mode Raw.

Pour vous présenter les différences dans l'effet produit par plusieurs caractères sous les deux modes, le processeur CON est tout d'abord fixé par les programmes sur le mode Raw. Une chaîne de test est ensuite sortie un nombre de fois déterminé sur ce processeur. Contrairement à ce qui se passe habituellement (lorsque le processeur est

en mode Cooked), il n'est pas possible sous ce mode de suspendre la sortie en actionnant simultanément <CTRL> et <S>.

Une fois terminée la sortie de la chaîne de test, le processeur passe en mode Cooked et la chaîne de test est à nouveau sortie. Si vous actionnez alors <CTRL> et <S>, la sortie sur l'écran sera suspendue comme à l'habitude.

La commutation entre les modes Raw et Cooked ne s'effectue cependant pas directement à travers une fonction. Il faut en effet déterminer tout d'abord ce qu'on appelle l'attribut de périphérique du processeur. Cet attribut contient différentes informations qui identifient le processeur et décrivent son mode de fonctionnement. Un bit de ce mot indique si le processeur est exploité en mode Raw ou Cooked. Ce bit sera donc annulé ou mis par nos programmes suivant le mode sous lequel le processeur sera censé travailler.

Listing : RAWCOOKP.PAS

```
{*********************************************************************}
{*                        R A W _ C O O K P                          *}
{*-------------------------------------------------------------------*}
{*  Fonction       : Offre deux fonctions permettant de faire        *}
{*                   passer un driver de périphérique de caractère   *}
{*                   en mode RAW ou en mode COOKED.                  *}
{*-------------------------------------------------------------------*}
{*  Auteur         : MICHAEL TISCHER                                 *}
{*  Développé le   : 16/08/1987                                      *}
{*  Dernière modif.: 28/03/1992                                      *}
{*********************************************************************}

program RAWCOOKP;

Uses Crt, Dos;                              { Intégrer unités CRT et DOS }

const ENTSTAND = 0;                      { handle 0 : périph. d'entrée std }
      SORSTAND = 1;                      { handle 1 : périph. de sortie std }

var Touche : char;     { Sert uniquement au programme de démonstration }

{*********************************************************************}
{* GetMode: lit l'attribut d'un driver de périphérique               *}
{* Entrée : le handle transmis doit être relié au périphérique à     *}
{*          appeler                                                  *}
{* Sortie : L'attribut de périphérique                               *}
{*********************************************************************}

function GetMode(Handle : integer) : integer;

var Regs : Registers;                          { Reg. d'appel d'interruption }

begin
  Regs.ah := $44;                { Numéro de fonction pour IOCTL: Get Mode }
  Regs.bx := Handle;
  MsDos( Regs );                          { Appeler interruption du DOS $21 }
  GetMode := Regs.dx;                { Transmettre attribut de périphérique }
end;

{*********************************************************************}
{* SetRaw: fait passer un driver de périphérique en mode RAW         *}
{* Entrée : le handle transmis doit être relié au périphérique à     *}
{*          appeler.                                                 *}
{* Sortie : Aucune                                                   *}
{*********************************************************************}

procedure SetRaw(Handle : integer);

var Regs : Registers;                          { Reg. d'appel d'interruption }

begin
  Regs.ax := $4401;              { Numéro de fonction pour IOCTL: Set Mode }
  Regs.bx := Handle;
  Regs.dx := GetMode(Handle) and 255 or 32;      { Nvl attr. de périph. }
  MsDos( Regs );                          { Appeler interruption du DOS $21 }
end;

{*********************************************************************}
{* SetCooked: fait passer un driver de caractère en mode COOKED      *}
{* Entrée : le handle transmis doit être relié au périphérique à     *}
{*          appeler.                                                 *}
{* Sortie : Aucune                                                   *}
{*********************************************************************}

procedure SetCooked(Handle : integer);

var Regs : Registers;                          { Reg. d'appel d'interruption }

begin
  Regs.ax := $4401;              { Numéro de fonction pour IOCTL: Set Mode }
  Regs.bx := Handle;
  Regs.dx := GetMode(Handle) and 223;      { Nouvel attribut de périph. }
  MsDos( Regs );                          { Appeler interruption du DOS $21 }
end;

{*********************************************************************}
{* SortieTest: sort une chaîne de test 1000 fois sur le périphérique *}
{*             de sortie standard                                    *}
{* Entrée : Aucune                                                   *}
{* Sortie : Aucune                                                   *}
{*********************************************************************}

procedure SortieTest;

var Regs     : Registers;                  { Reg. d'appel d'interruption }
    Compteur : integer;                          { Variable de boucle }
    Test     : string[9];                  { La chaîne de test à sortir }

begin
  Test    := 'Test.... ';
  Regs.bx := SORSTAND;  { Sortir sur le périphérique de sortie standard }
  Regs.cx := 9;                                  { Nombre de caractères }
  Regs.ds := Seg(Test);                    { Adresse de segment du texte }
  Regs.dx := Ofs(Test)+1;                   { Adresse d'offset du texte }
  for Compteur := 1 to 1000 do              { Sortir la chaîne 1000 fois }
    begin
      Regs.ah := $40;            { Numéro de fonction pour écrire handle }
      MsDos( Regs );                  { Appeler interruption du DOS $21 }
    end;
  writeln;
end;

{*********************************************************************}
{*                        PROGRAMME PRINCIPAL                        *}
{*********************************************************************}

begin
  ClrScr;                                                { Vider écran }
  writeln('RAWCOOKP (c) 1987, 92 by Michael Tischer'@13@10);
  writeln('Le driver de console est maintenant en mode RAW.',
```

```
        ' Les touches de commande,');
  writeln('comme <CTRL-S> par exemple ne seront pas identifiées lors',
          ' de la sortie.'#13#10'Faites-en donc l''essai pendant les',
          ' sorties suivantes !');
  writeln('Veuillez frapper une touche pour commencer...');
  Touche := ReadKey;                                { Attendre une touche }

  SetRaw(ENTSTAND);        { Faire passer driver de console en mode RAW }
  SortieTest;                       { Sortir chaine de test 1000 fois }

  ClrScr;                                           { Vider l'écran }
    while KeyPressed do                        { Vider le buffer clavier }
    Touche := ReadKey;                                  { Lire touche }
  writeln('Le driver de console se trouve maintenant en mode COOKED.');
  writeln('Les touches de commande comme <CTRL-S> par exemple sont');
  writeln('identifiées lors de la sortie et traitées en conséquence.');
  writeln('Veuillez frapper une touche pour commencer...');

  Touche := ReadKey;                                { Attendre une touche }
  SetCooked(ENTSTAND);             { Faire passer driver en mode COOKED }
  SortieTest;                       { Sortir la chaine de test 1000 fois }
end.
```

Listing : RAWCOOKC.C

```
/*****************************************************************
*                          R A W _ C O O K C                     *
*----------------------------------------------------------------*
*   Fonction      : offre deux fonctions permettant de faire passer *
*                   un driver de caractère en mode RAW ou en mode  *
*                   COOKED.                                        *
*----------------------------------------------------------------*
*   Auteur        : MICHAEL TISCHER                                *
*   Développé le  : 16/08/1987                                     *
*   Dernière modif.: 29/02/1989                                    *
*----------------------------------------------------------------*
*   (MICROSOFT C)                                                  *
*   Création      : CL /AS RAW_COOK.C                              *
*   Appel         : RAW_COOK                                       *
*----------------------------------------------------------------*
*   (BORLAND TURBO C)                                              *
*   Création      : Avec instruction RUN dans ligne d'instruction  *
*                   (sans fichier Project)                         *
*****************************************************************/

/*-- Intégrer fichiers Include ---------------------------------*/
#include <dos.h>
#include <stdio.h>
#include <conio.h>

/*-- Constantes ------------------------------------------------*/
#define ENTSTAND 0              /* handle 0 = périph. d'entrée std */
#define SORSTAND 1              /* handle 1 = périph. de sortie std */

/*****************************************************************
* GETMODE: lit l'attribut d'un driver de périphérique            *
* Entrée : le handle transmis doit être relié au périphérique à  *
*          appeler.                                              *
* Sortie : L'attribut de périphérique                            *
*****************************************************************/

int GetMode( int Handle )
{
  union REGS Register;                  /* Reg. d'appel d'interruption */

  Register.x.ax = 0x4400;   /* Numéro de fonction pour IOCTL: Get Mode */
  Register.x.bx = Handle;
  intdos(&Register, &Register);   /* Appeler interruption du DOS 21(h) */
  return(Register.x.dx);       /* Transmettre attribut de périphérique */
}

/*****************************************************************
* SETRAW: fait passer un driver de caractère en mode RAW         *
* Entrée : le handle transmis doit être relié au périphérique à  *
*          appeler                                               *
* Sortie : Aucune                                                *
*****************************************************************/

void SetRaw( int Handle )
{
  union REGS Register;                  /* Reg. d'appel d'interruption */

  Register.x.ax = 0x4401;               /* N° fonction IOCTL : Set Mode */
  Register.x.bx = Handle;
  Register.x.dx = (GetMode(Handle) & 255) | 32;  /* Nvl attrb. périph. */
  intdos(&Register, &Register);   /* Appeler interruption du DOS 21(h) */
}

/*****************************************************************
* SETCOOKED: fait passer un driver de caractère en mode COOKED   *
* Entrée : le handle transmis doit être relié au périphérique à  *
*          appeler.                                              *
* Sortie : Aucune                                                *
*****************************************************************/

void SetCooked( int Handle )
{
  union REGS Register;                  /* Reg. d'appel d'interruption */

  Register.x.ax = 0x4401;               /* N° fonction IOCTL : Set Mode */
  Register.x.bx = Handle;
  Register.x.dx = GetMode(Handle) & 223; /* Nouvel attribut de périph. */
  intdos(&Register, &Register);   /* Appeler interruption du DOS 21(h) */
}

/*****************************************************************
* SORTIETEST: sort une chaine de test 1000 fois sur le périphérique *
*             de sortie standard.                                *
* Entrée : Aucune                                                *
* Sortie : Aucune                                                *
*****************************************************************/

void SortieTest( void )
{
  int           i;                      /* Variable de boucle */
  static char  Test[] = "Test.... ";    /* Le texte à sortir */

  printf("\n");
  for( i = 0; i < 1000; i++ )                   /* Sortir 1000 fois */
    fputs( Test, stdout );  /* Sortir chaine sur périph. de sortie std */
  printf("\n");
}

/*****************************************************************/
/**                       PROGRAMME PRINCIPAL                   **/
/*****************************************************************/

void main()

{
  printf("\nRAWCOOKC (c) 1987, 92 by Michael TISCHER\n\n");
  printf("Le driver de console (Clavier, écran) se trouve maintenant\n");
  printf("en mode RAW. C'est pourquoi, lors des sorties suivantes,\n");
  printf("les caractères de commande comme <CTRL-S> par exemple ne\n");
  printf("seront pas identifiés.\n");
  printf("Essayez et vous verrez.\n\n");
  printf("Frappez une touche pour commencer...");
  getch();                                      /* Attendre une touche */
  SetRaw(ENTSTAND);       /* Faire passer driver de console en mode RAW */
  SortieTest();
  while( kbhit())           /* Eliminer du buffer clavier les touches */
                                        /* entrées entre-temps. */
  printf("Le driver de console se trouve maintenant en mode COOKED.\n");
  printf("Les touches de commande comme <CTRL-S> par exemple, sont \n");
  printf("identifiées lors de la sortie et traitées en conséquence !\n");
  printf("Veuillez frapper une touche pour commencer...");
  getch();                                      /* Attendre une touche */
  SetCooked(ENTSTAND);    /* Faire passer driver console en mode COOKED */
  SortieTest();
}
```

18.4. Filtres du DOS

Un filtre est, de façon très générale, un instrument dans lequel on introduit quelque chose pour en faire ressortir autre chose. Cette définition s'applique parfaitement à la notion de filtre sous le système d'exploitation : on entre en effet des caractères dans ce filtre pour les faire manipuler d'une certaine façon avant de les faire ressortir. Cette manipulation peut revêtir différentes formes. Il peut par exemple s'agir de trier l'entrée, de remplacer certaines entrées par d'autres, de coder ou de décoder l'entrée, etc...

Le DOS fournit trois filtres standard :

- ✓ FIND pour rechercher une séquence de caractères déterminées
- ✓ SORT pour trier des textes ou des données
- ✓ MORE pour l'affichage page par page d'un texte

Ces filtres reposent, comme tous les autres filtres concevables, sur la possibilité de rediriger les entrée et sortie standard. C'est en effet ce qui permet de lire des caractères sur le périphérique d'entrée standard et de les manipuler pour les ressortir ensuite sur le périphérique de sortie standard. Sous le DOS, le périphérique d'entrée standard est le clavier et le périphérique de sortie standard l'écran mais depuis la version 2.0 du DOS, il est possible, au niveau de l'utilisateur, de détourner vers des fichiers les entrée et sortie standard. Un filtre qui lit des caractères sur le périphérique d'entrée standard ne remarquera pas, de ce fait, que ces caractères ne viennent pas du clavier mais qu'ils sont tirés d'un fichier. Cela est rendu possible par le fait que le filtre de lecture et écriture a recours à l'une des fonctions handle du DOS. A cet effet, cinq handles sont réservés en permanence par le DOS :

0	Entrée standard	CON (Clavier)
1	Sortie standard	CON (Ecran)
2	Sortie d'erreur	CON (Ecran)
3	Interface série standard	AUX
4	Imprimante standard	PRN

Lorsque l'utilisateur appelle un programme (dans notre exemple le filtre SORT) à partir du niveau utilisateur, il peut rediriger l'entrée avec le caractère "<" et la sortie à l'aide du caractère ">". L'exemple suivant illustre comment détourner l'entrée sur le fichier ENTREE.TXT et la sortie sur le fichier SORTIE.TXT :

```
sort <entree.txt >sortie.txt
```

Une fois que cette ligne aura été entrée au clavier par l'utilisateur et transmise à l'interpréteur de commandes en actionnant la touche RETURN, ce dernier la passera au crible. Il notera tout d'abord qu'il s'agit d'appeler un programme appelé SORT. Il

rencontrera ensuite l'expression "<entree.txt" qui lui enjoint de rediriger l'entrée standard. Le processeur d'instructions s'exécutera donc, en associant au fichier ENTREE.TXT le handle 0 (entrée standard) qui était jusqu'ici associé au clavier. Il en ira de même pour l'expression ">sortie.txt", si ce n'est naturellement que c'est le handle 1 et non le handle 0 qui sera redéfini. Concrètement cette opération consiste tout d'abord à fermer le handle concerné.

La fonction Ouvrir handle est ensuite appelée avec le nom du fichier voulu. Comme le DOS attribue systématiquement le premier handle libre, c'est automatiquement le handle qui vient juste d'être refermé qui est immédiatement réouvert.

L'examen de la ligne d'entrée est alors terminé et le processeur d'instructions appelle maintenant le programme SORT à l'aide de la fonction EXEC (fonction 4Bh du DOS). Comme le programme appelé à l'aide de la fonction EXEC dispose de tous les handles du programme d'appel, le programme SORT peut maintenant lire et sortir des caractères respectivement à travers les handles 0 et 1. Peu importe pour lui d'où viennent effectivement ces caractères et d'ailleurs il n'a pas la possibilité de le savoir.

Une fois que le programme SORT a terminé son travail, il rend le contrôle à l'interpréteur de commandes. Ce dernier annule la redirection des entrée et sortie avant d'attendre de nouvelles entrées de l'utilisateur.

Ce principe des filtres tel qu'il est soutenu par le DOS n'acquiert cepen-dant toute sa puissance qu'à partir du moment où il est combiné avec la méthode du DOS, Pipe. Cette expression vient du terme pipe-line, qui signifie oléoduc. La fonction des pipe du DOS rappelle en effet ces gigantesques canalisations puisqu'ils servent à détourner des caractères d'un programme vers un autre, ce qui permet de relier différents programmes entre eux.

Lorsqu'on fait appel à cette possibilité, les caractères sortis par un programme sur le périphérique de sortie standard sont lus par l'autre programme lorsqu'il accède au périphérique d'entrée standard. Comme pour le détournement des entrée et sortie, les pipe sont mis en place à l'insu des deux programmes. La différence entre les deux méthodes réside dans le fait que la redirection des entrée et sortie standard permet seulement de détourner les données vers un périphérique ou vers un fichier alors que les pipe retransmettent les données à un autre programme.

La technique du pipe-lining présente un intérêt tout particulier dans le domaine des filtres car elle permet d'activer plusieurs filtres successifs et de les combiner entre eux. On utilise à cet effet le symbole Pipe "|" qui doit être placé entre les programmes à relier. Essayons d'illustrer ce principe par un exemple concret :

Il s'agira de trier puis de sortir page par page sur l'écran un fichier de texte, que nous appellerons DEMO.TXT. C'est une tâche qui peut sembler assez compliquée et pourtant elle est très facile à réaliser à l'aide de deux filtres du DOS : SORT triera notre fichier et MORE l'affichera ensuite page par page sur l'écran. Mais comment envoyer nos instructions à l'interpréteur de commandes ? Il convient naturellement tout d'abord

d'appeler SORT en indiquant à ce filtre qu'il doit trier le fichier DEMO.TXT. Nous pouvons utiliser à cet effet la redirection de l'entrée standard, que nous avons déjà expliquée au début de ce chapitre :

```
SORT <DEMO.TXT
```

Si vous entriez cette instruction, SORT sortirait maintenant le fichier DEMO.TXT, trié, sur l'écran. Pour que nous puissions examiner le résultat, il serait cependant préférable que la sortie se fasse page écran par page écran. A cet effet, nous pourrions rediriger la sortie de SORT vers un fichier (TEMP.TXT par exemple) puis faire afficher ce fichier avec MORE (pour la sortie page par page). Nous devrions donc entrer les instructions suivantes :

```
SORT <DEMO.TXT >TEMP.TXT
MORE <TEMP.TXT
```

La fonction de pipe-lining nous permet justement de simplifier ces deux entrées en combinant directement le filtre SORT avec le filtre MORE :

```
SORT <DEMO.TXT | MORE
```

Cette entrée a pour effet d'envoyer directement à MORE la sortie de SORT, de sorte que le fichier trié est immédiatement sorti page par page. Il est ainsi possible d'accrocher à la suite les uns des autres autant de filtres qu'on le souhaite, pourvu que la logique des différents programmes soit respectée. Le DOS procède toujours de gauche à droite, c'est-à-dire qu'il renvoie la sortie du premier programme à l'entrée du second programme dont la sortie est envoyée en entrée au troisième programme, et ainsi de suite. Rien n'empêche enfin, pour le dernier programme, de rediriger la sortie finale à l'aide du caractère ">" de façon à ce que le résultat de toute cette chaîne de programmes ou de filtres soit envoyé dans un fichier ou sur un autre périphérique au lieu d'être affiché sur l'écran.

Il convient toutefois de noter que le DOS n'est pas réellement capable de retransmettre directement des données d'un filtre à l'autre. Cela exigerait en effet que deux filtres fussent exécutés simultanément alors que le DOS ne permet pas encore le travail multitâche dans sa version actuelle. C'est pourquoi il a en fait recours à la méthode que nous indiquions ci-dessus. Il appelle donc d'abord le premier fichier et envoie sa sortie dans ce qu'on appelle un fichier pipe. Une fois terminé le travail du premier filtre, le second est appelé et son entrée est redirigée sur le fichier pipe. Les sorties du premier filtre deviennent donc les entrées du second. Il est ensuite procédé de même pour tous les filtres appelés. Une fois terminé le travail du fichier filtre, le fichier pipe est supprimé de sorte qu'il n'est jamais visible pour l'utilisateur.

18.5. Un exemple de filtre

Dans le jargon des informaticiens, on désigne sous le terme de dumping l'opération consistant à sortir le contenu d'un fichier en faisant afficher les différents caractères à la fois sous forme de caractères ASCII et sous forme de codes hexadécimaux. C'est cette fonction qui est réalisée par les exemples de programmes dont nous vous présentons maintenant le listing et qui s'appellent tous DUMP. Il s'agit du même programme écrit en Assembleur, Pascal et C.

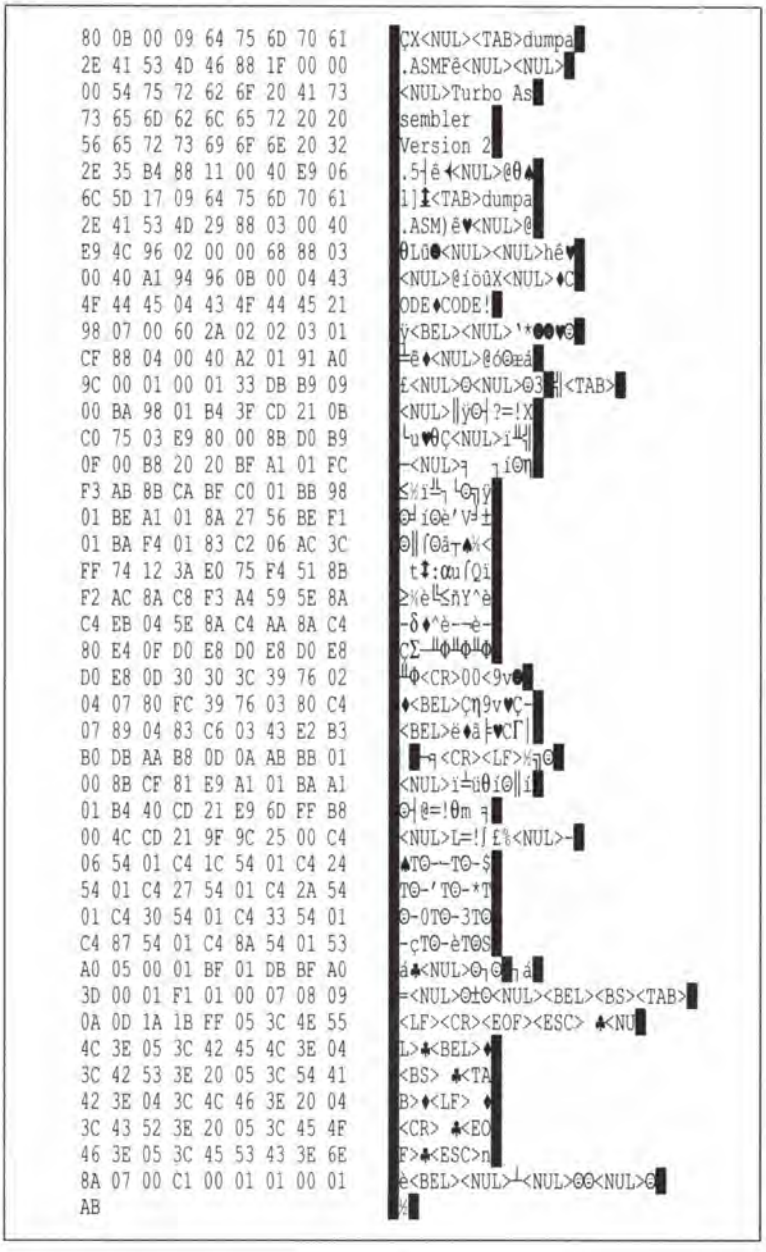
La tâche de ce filtre consiste à lire des caractères sur le périphérique d'entrée standard pour les ressortir comme DUMP sur le périphérique de sortie standard aussi bien comme caractères ASCII qu'en code hexa. Pour sortir les caractères ASCII, DUMP distingue les caractères ASCII normaux (lettres, nombres, etc...) et les caractères de commande comme Carriage Return, Line Feed, etc... Ces derniers ne sont pas sortis en effet sous forme de codes ASCII mais comme des textes tels que '<CR>' pour Carriage Return ou '<LF>' pour Line Feed. La fonction du filtre Dump est donc assez élémentaire mais il peut rendre de grands services dans le travail avec le système d'exploitation, par exemple lorsqu'il s'agit de consulter rapidement le contenu d'un fichier.

Le mode de fonctionnement des programmes Dump est caractéristique des filtres. Comme ils sortent en effet sur une ligne 9 caractères au maximum sous forme de caractères ASCII et de codes hexa, ils réclament tout d'abord 9 caractères au périphérique d'entrée standard à travers la fonction handle de lecture. S'il ne reste plus autant de caractères disponibles, les programmes se contentent d'un nombre inférieur. Les caractères entrés sont ensuite convertis en caractères ASCII et en codes hexa à l'intérieur d'un buffer interne. Ce buffer reçoit chaque fois une ligne entière (78 caractères).

Une fois terminé le traitement du buffer, le buffer est sorti sur le périphérique de sortie standard à l'aide de la fonction handle d'écriture. Cette opération se répète jusqu'à ce qu'aucun caractère ne puisse plus être lu sur le périphérique d'entrée standard.

Par exemple, voici l'aspect du DUMP du fichier DUMPA.OBJ une fois que le module DUMPA.ASM a été assemblé à l'aide du Turbo Assembler :

Exemple :



```
80 0B 00 09 64 75 6D 70 61   ÇX<NUL><TAB>dumpa
2E 41 53 4D 46 88 1F 00 00   .ASMFê<NUL><NUL>
00 54 75 72 62 6F 20 41 73   <NUL>Turbo As
73 65 6D 62 6C 65 72 20 20   sembler
56 65 72 73 69 6F 6E 20 32   Version 2
2E 35 B4 88 11 00 40 E9 06   .5┤ê┤<NUL>@Θ♠
6C 5D 17 09 64 75 6D 70 61   l]↨<TAB>dumpa
2E 41 53 4D 29 88 03 00 40   .ASM)ê♥<NUL>@
E9 4C 96 02 00 00 68 88 03   ΘLûΘ<NUL><NUL>hê♥
00 40 A1 94 96 0B 00 04 43   <NUL>@íöûX<NUL>♦C
4F 44 45 04 43 4F 44 45 21   ODE♦CODE!
98 07 00 60 2A 02 02 03 01   ÿ<BEL><NUL>`'*☻☻♥☺
CF 88 04 00 40 A2 01 91 AO   ╧ê♦<NUL>@óΘæá
9C 00 01 00 01 33 DB B9 09   £<NUL>Θ<NUL>Θ3╫╣<TAB>
00 BA 98 01 B4 3F CD 21 0B   <NUL>║ÿΘ┤?=!X
C0 75 03 E9 80 00 8B D0 B9   └u♥ΘÇ<NUL>ï╨╣
0F 00 B8 20 20 BF A1 01 FC   ─<NUL>╕  ┐íΘn
F3 AB 8B CA BF C0 01 BB 98   ≤½ï╩┐└Θη╗ÿ
01 BE A1 01 8A 27 56 BE F1   Θ╛íΘè'V╛±
01 BA F4 01 83 C2 06 AC 3C   Θ║⌠Θâ┬♠¼<
FF 74 12 3A E0 75 F4 51 8B    t↕:αu⌠Qï
F2 AC 8A C8 F3 A4 59 5E 8A   ≥¼è╚≤ñY^è
C4 EB 04 5E 8A C4 AA 8A C4   -δ♦^è-¬è-
80 E4 0F D0 E8 D0 E8 D0 E8   ÇΣ╨φ╨φ╨φ
D0 E8 0D 30 30 3C 39 76 02   ╨φ<CR>00<9v☻
04 07 80 FC 39 76 03 80 C4   ♦<BEL>Çη9v♥Ç-
07 89 04 83 C6 03 43 E2 B3   <BEL>ë♦â╞♥ΘΓ│
B0 DB AA B8 0D 0A AB BB 01   ▒╬¬╕<CR><LF>½╗Θ☺
00 8B CF 81 E9 A1 01 BA A1   <NUL>ï╧üθíΘ║í
01 B4 40 CD 21 E9 6D FF B8   Θ┤@=!θm ╕
00 4C CD 21 9F 9C 25 00 C4   <NUL>L=!ƒ£%<NUL>-
06 54 01 C4 1C 54 01 C4 24   ♠TΘ-∟TΘ-$
54 01 C4 27 54 01 C4 2A 54   TΘ-'TΘ-*T
01 C4 30 54 01 C4 33 54 01   Θ-0TΘ-3TΘ
C4 87 54 01 C4 8A 54 01 53   -çTΘ-èTΘS
A0 05 00 01 BF 01 DB BF A0   á♣<NUL>Θ┐Θ▒┐á
3D 00 01 F1 01 00 07 08 09   =<NUL>Θ±Θ<NUL><BEL><BS><TAB>
0A 0D 1A 1B FF 05 3C 4E 55   <LF><CR><EOF><ESC> ♣<NU
4C 3E 05 3C 42 45 4C 3E 04   L>♣<BEL>♦
3C 42 53 3E 20 05 3C 54 41   <BS> ♣<TA
42 3E 04 3C 4C 46 3E 20 04   B>♦<LF> ♦
3C 43 52 3E 20 05 3C 45 4F   <CR> ♣<EO
46 3E 05 3C 45 53 43 3E 6E   F>♣<ESC>n
8A 07 00 C1 00 01 01 00 01   è<BEL><NUL>┴<NUL>ΘΘ<NUL>Θ
AB                           ½
```

Listing : DUMPP.PAS

```
{*****************************************************************}
{*                          D U M P P                            *}
{*---------------------------------------------------------------*}
{*   Fonction       : un filtre qui lit des caractères sur l'entrée *}
{*                    standard pour les ressortir sur la sortie    *}
{*                    standard sous forme de dump hexa et ASCII    *}
{*---------------------------------------------------------------*}
{*   Auteur         : MICHAEL TISCHER                            *}
{*   Développé le   :  8/08/1987                                 *}
{*   Dernière modif.: 28/02/1992                                 *}
{*---------------------------------------------------------------*}
{*   Infos          : Ce programme ne peut être appelé qu'à partir *}
{*                    du niveau DOS après avoir été compilé par    *}
{*                    TURBO en fichier EXE                         *}
{*****************************************************************}

program DUMPP;

Uses Dos;                                          { Intégrer unité DOS }

{$V-}                                  { Pas de test de la longueur de chaîne }

const NUL = 0;                                     { Code ASCII caractère NUL }
      BEL = 7;                                      { Code ASCII sonnerie }
      BS  = 8;                                      { Code ASCII Backspace }
      TAB = 9;                                      { Code ASCII Tabulateur }
      LF  = 10;                                     { Code ASCII Line Feed }
      CR  = 13;                                     { Code ASCII Carriage Return }
      EOF = 26;                                     { Code ASCII Fin de fichier }
      ESC = 27;                                     { Code ASCII Escape }

type CCText = string[3]; { sert à transmettre le nom d'un car. spécial }
     DumpBf = array[1..80] of char;           { Reçoit le dump à sortir }

{*****************************************************************}
{* CC: Ecrit le nom d'un caractère de commande dans un buffer    *}
{* Entrée : voir plus bas                                        *}
{* Sortie  : Aucune                                              *}
{* Infos   : Après appel de cette procédure, le pointeur transmis *}
{*           désigne le dernier caractère du nom de caractère de *}
{*           commande dans le buffer Dump                        *}
{*****************************************************************}

procedure CC(var Buffer  : DumpBf;   { C'est ici qu'est entré le texte }
                 Text    : CCText;                  { Le texte à entrer }
             var Pointer : integer);      { Adr. départ txt dans buffer }

var Compteur : integer;                            { Compteur de boucle }

begin
 Buffer[Pointer] := '<';                  { Précède caractère de commande }

 for Compteur := 1 to length(Text) do  { Transférer texte dans buffer }
  Buffer[Pointer + Compteur] := Text[Compteur];

 Buffer[Pointer + Compteur + 1] := '>'; { Fin du caractère de commande }
 Pointer := Pointer + Compteur + 2;          { Ptr sur caractère suivant }
end;

{*****************************************************************}
{* DoDump: lit les caractères et les sort sous forme de Dump     *}
{* Entrée  : Aucune                                              *}
{* Sortie  : Aucune                                              *}
{*****************************************************************}

procedure DoDump;

var Regs : Registers;      { Variable de registre pour appeler interr. }
    NeufOctets : array[1..9] of char;          {Reçoit caractères lus }
    DumpBuf  : DumpBf;                    { Reçoit une ligne du DUMP }
    ChHexa,
    Compteur,                                      { Compteur de boucle }
    NextA    : integer;         { Pointeur dans buffer pour code ASCII }
    Fin      : boolean;                        { Pas lu d'autre octet ? }

begin
 Fin := false;                                        { Pas encore Fin }

 repeat
   Regs.ah := $3F;             { Numéro de fonction pour Lecture Handle }
   Regs.bx := 0;                    { Périph. d'entrée std : Handle 0 }
   Regs.cx := 9;                                    { Lire 9 caractères }
   Regs.ds := seg(NeufOctets);          { Adresse de segment du buffer }
   Regs.dx := ofs(NeufOctets);           { Adresse d'offset du buffer }
   MsDos( Regs );                      { Appeler interruption DOS $21 }

   if( Regs.ax = 0 ) then
     Fin := true                                  { Aucun caractère lu ? }
   else
     begin                                                      { NON }
       for Compteur := 1 to 30 do          { Remplir buffer d'espaces }
         DumpBuf [Compteur] := ' ';
       DumpBuf[31] := #219;          { Fixer car. séparation hexa/ASCII }
       NextA := 32;  { Caractères ASCII après caractère de séparation }

       for Compteur := 1 to Regs.ax do   { Traiter les caractères lus }
       begin
         ChHexa := ord(NeufOctets[Compteur]) shr 4 + 48;   { octet en }
         if( ChHexa > 57) then
           ChHexa := ChHexa + 7;                      { 1er chiffre hexa }
         DumpBuf[Compteur * 3 - 2] := chr(ChHexa);  { Sauve ds buffer }

         ChHexa := ord(NeufOctets[Compteur]) and 15 + 48;  { octet en }
         if( ChHexa > 57) then
           ChHexa := ChHexa + 7;                      { 2nd chiffre hexa }
         DumpBuf[Compteur * 3 - 1] := chr(ChHexa);  { Sauve ds buffer }

         case ord(NeufOctets[Compteur]) of        { Tester code ASCII }
           NUL : CC(DumpBuf, 'NUL', NextA);            { Caractère NUL }
           BEL : CC(DumpBuf, 'BEL', NextA);                 { Sonnerie }
           BS  : CC(DumpBuf, 'BS' , NextA);                 { Backspace }
           TAB : CC(DumpBuf, 'TAB', NextA);                 { Tabulateur }
           LF  : CC(DumpBuf, 'LF' , NextA);                 { Line Feed }
           CR  : CC(DumpBuf, 'CR' , NextA);            { Carriage Return }
           EOF : CC(DumpBuf, 'EOF', NextA);            { Fin de fichier }
           ESC : CC(DumpBuf, 'ESC', NextA);                    { Escape }
           else
             begin                                    { Caractère normal }
               DumpBuf[NextA] := NeufOctets[Compteur];  { Sauve ASCII }
               NextA := succ(NextA)          { Fixe ptr sur car. suivant }
             end
         end;
       end;

       DumpBuf[NextA] := #219;                  { Fixer caractère de fin }
       DumpBuf[NextA+1] := chr(CR);                              { CR/LF}
       DumpBuf[NextA+2] := chr(LF);                 { à la fin du buffer }
       Regs.ah := $40;       { Numéro de fonction pour Ecriture Handle }
       Regs.bx := 1;                    { périph. sortie std : Handle 1 }
       Regs.cx := NextA+2;                          { Nombre de caractères }
       Regs.ds := seg(DumpBuf);           { Adresse de segment du buffer }
       Regs.dx := ofs(DumpBuf);            { Adresse d'offset du buffer }
       MsDos( Regs );                      { Appeler interruption DOS $21 }
     end;
 until Fin;      { Répéter jusqu'à ce que plus de caractère disponible }
end;

{*****************************************************************}
{*                       PROGRAMME PRINCIPAL                     *}
{*****************************************************************}

begin
 DoDump;                                                { Sortir Dump }
end.
```

Listing : DUMPC.C

```
/*****************************************************************
*                          D U M P C                             *
*****************************************************************
*   Fonction      : un filtre qui lit des caractères sur l'entrée *
*                   standard pour les ressortir sur la sortie     *
*                   standard sous forme de dump hexa et ASCII     *
*****************************************************************
*   Auteur        : MICHAEL TISCHER                               *
*   Développé le  : 14/08/1987                                    *
*   Dernière modif.: 29/02/1992                                   *
*****************************************************************
*   (MICROSOFT C)                                                 *
*   Création      : CL /AS DUMPC.C                                *
*   Appel         : DUMPC [<Entrée] [>Sortie]                     *
*****************************************************************
*   (BORLAND TURBO C)                                             *
*   Création      : Créer avec l'instruction COMPILE - MAKE       *
*                   (sans fichier Project) et faire exécuter      *
*                   sous l'environnement DOS                      *
*****************************************************************/

/*-- fichiers Include ---------------------------------------------*/
#include <stdio.h>                          /* Intégrer fichiers header */
#include <dos.h>

/*-- Typedefs -----------------------------------------------------*/
typedef unsigned char byte;     /* Voilà comment se bricoler un OCTET */

/*-- Constantes ---------------------------------------------------*/
#define NUL     0                           /* Code du caractère NUL */
#define BEL     7                           /* Code de la sonnerie */
#define BS      8                      /* Code de la touche Backspace */
#define TAB     9                      /* Code de la touche tabulateur */
#define LF      10                          /* Code de Line Feed */
#define CR      13                     /* Code de la touche Return */
#define ESC     27                     /* Code de la touche Escape */

/*-- Macros -------------------------------------------------------*/
#define tohex(c) ( ((c)<10) ? ((c) | 48) : ((c) + 'A' - 10) )

/*****************************************************************
* GETSTDIN: lit un nombre déterminé de caractères sur le périphérique *
*           d'entrée standard et les place dans un buffer       *
* Entrée  : voir plus bas                                       *
* Sortie  : Nombre de caractères lus                            *
*****************************************************************/

unsigned int GetStdIn( char *Buffer, unsigned NombreMax )
{
  union REGS    Register;           /* Registre d'appel d'interruption */
  struct SREGS  Segments;           /* Reçoit les registres de segment */

  segread( &Segments );     /* Charger contenu des registres de segment */
  Register.h.ah = 0x3F;                     /* Numéro de fonction pour */
  Register.x.bx = 0;                  /* Périph. d'entrée std : handle 0 */
  Register.x.cx = NombreMax;                /* Nombre d'octets à lire */
  Register.x.dx = (unsigned int) Buffer; /* Adresse d'offset du buffer */
  intdosx(&Register, &Register, &Segments);         /* int. DOS 21(h) */
  return(Register.x.ax);            /* Nombre d'octets lus à l'appelant */
}

/*****************************************************************
* STRAP: Ajouter caractère à une chaine                         *
* Entrée  : voir plus bas                                       *
* Sortie  : Pointeur à la suite du dernier caractère ajouté     *
*****************************************************************/

char *Strap(char *String, char *Ajouter)
{
  while( *Ajouter )          /* Répéter jusqu'à ce que '\0' rencontré */
    *String++ = *Ajouter++;                  /* Transférer caractère */
  return(String);           /* Transmettre pointeur à fonction appelée */
}

/*****************************************************************
* DODUMP: charge les caractères et les sort comme Dump          *
* Entrée  : Aucune                                              *
* Sortie  : Aucune                                              *
*****************************************************************/

void DoDump( void )

{
  char NeufOctets[9],                   /* Reçoit les caractères entrés */
       DumpBuf[80],                        /* Reçoit une ligne du DUMP */
       *NextAscii;  /* Désigne prochain caractère ASCII dans le buffer */
  byte i,                                       /* Compteur de boucle */
       Nombre;                             /* Nombre d'octets entrés */

  DumpBuf[30]  = 219;           /* Caractère de séparation hexa/ASCII */
  while( (Nombre = GetStdIn(NeufOctets, 9)) != 0 )      /* Caractère ? */
  {                                                          /* Oui */
    for( i = 0; i < 30; DumpBuf[i++] = ' ')         /* Remplir d'espaces */
      ;
    NextAscii = &DumpBuf[31];                      /* Caractères ASCII */
    for( i = 0; i < Nombre; i++ )       /* Traiter ts caractères entrés */
    {
      DumpBuf[i*3]   = tohex((byte) NeufOctets[i] >> 4); /* Conv. code */
      DumpBuf[i*3+1] = tohex((byte) NeufOctets[i] & 15);    /* en hexa */
      switch (NeufOctets[i])                    /* Evaluer code ASCII */
      {
        case NUL : NextAscii = Strap(NextAscii, "<NUL>");
                   break;
        case BEL : NextAscii = Strap(NextAscii, "<BEL>");
                   break;
        case BS  : NextAscii = Strap(NextAscii, "<BS>");
                   break;
        case TAB : NextAscii = Strap(NextAscii, "<TAB>");
                   break;
        case LF  : NextAscii = Strap(NextAscii, "<LF>");
                   break;
        case CR  : NextAscii = Strap(NextAscii, "<CR>");
                   break;
        case ESC : NextAscii = Strap(NextAscii, "<ESC>");
                   break;
        case EOF : NextAscii = Strap(NextAscii, "<EOF>");
                   break;
        default  : *NextAscii++ = NeufOctets[i];
      }
    }
    *NextAscii     = 219;     /* Caractère de fin pour affichage ASCII */
    *(NextAscii+1) = '\r';       /* Carriage Return à la fin du buffer */
    *(NextAscii+2) = '\0';        /* NUL converti en LF pendant sortie */
    puts(DumpBuf);          /* Ecrire chaine sur périph. de sortie std */
  }
}

/*****************************************************************/
/**                       PROGRAMME PRINCIPAL                   **/
/*****************************************************************/

void main()

{
  DoDump();                                 /* Entrer et sortir caractère */
}
```

Listing : DUMPA.ASM

```
;*****************************************************************;
;*                        D U M P A . A S M                      *;
;*---------------------------------------------------------------*;
;*   Fonction       : Un filtre qui lit des caractères sur l'entrée *;
;*                    standard pour les ressortir sur la sortie    *;
;*                    standard sous forme de dump hexa et ASCII    *;
;*---------------------------------------------------------------*;
;*   Auteur         : MICHAEL TISCHER                             *;
;*   Développé le   :  1.08.87                                    *;
;*   Dernière modif. : 20.03.92                                   *;
;*---------------------------------------------------------------*;
;*   Pour assembler : MASM DUMPA;                                 *;
;*                    LINK DUMPA;                                 *;
;*                    EXE2BIN DUMPA DUMP.COM  ou                  *;
;*                                                                *;
;*                    TASM DUMPA                                  *;
;*                    TLINK /T DUMPA                              *;
;*---------------------------------------------------------------*;
;*   Appel          : DUMPA [<Entrée] [>Sortie]                  *;
;*****************************************************************;

;-- Constantes ---------------------------------------------------

NUL     equ 0                           ;Code ASCII caractère NUL
BEL     equ 7                           ;Code ASCII sonnerie
BS      equ 8                           ;Code ASCII Backspace
TAB     equ 9                           ;Code ASCII Tabulateur
LF      equ 10                          ;Code ASCII Line Feed
CR      equ 13                          ;Code ASCII Carriage Return
EOF     equ 26                          ;Code ASCII Fin de fichier
ESC     equ 27                          ;Code ASCII Escape

;-- Ici commence véritablement le programme ----------------------

code    segment para 'CODE'             ;Définition du segment de code

        org 100h

        assume cs:code, ds:code, es:code, ss:code

;-- Routine initiale ---------------------------------------------

dump    label near
        ;-- Lire 9 octets sur le périphérique d'entrée standard ------

        xor  bx,bx                  ;Entrée standard porte le handle 0
        mov  cx,9                           ;Charger 9 caractères
        mov  dx,offset neufoctets           ;Adresse du buffer
        mov  ah,3Fh                 ;Code de fonction pour lecture Handle
        int  21h                            ;Appeler fonction DOS
        or   ax,ax                  ;Des caractères ont-ils été lus ?
        jne  dodump                         ;OUI --> Traiter ligne
        jmp  findump                        ;NON --> FIN DUMP

dodump: mov  dx,ax                  ;Ranger nombre de caractères lus

        ;-- Remplir d'espaces buffer de sortie -----------------------

        mov  cx,15                          ;15 mots (30 octets)
        mov  ax,2020h               ;Code ASCII du " " dans AH et AL
        mov  di,offset dumpBuf  ;Augmenter adresse du buffer de sortie
        cld                                 ;pour instructions de chaînes
        rep  stosw                          ;Remplir buffer d'espaces

        ;-- Mettre en place buffer de sortie -------------------------

        mov  cx,dx              ;Aller chercher nombre de caractères lus
        mov  di,offset dumpBuf+31   ;Pos. codes ASCII dans le buffer
        mov  bx,offset neufoctets   ;Pointeur sur le buffer d'entrée
        mov  si,offset dumpBuf      ;Pos. codes hexa dans le buffer

octetin: mov ah,[bx]                                ;Lire un octet
        push si                             ;Ranger SI sur la pile
        mov  si,offset sptab            ;Adr. table de car. spéciaux
        mov  dx,offset sptext-6         ;Adr. texte de car. spéciaux
sptest: add  dx,6           ;Prochaine entrée dans les textes spéciaux
        lodsb               ;Charger code de table de caractères spéciaux
        cmp  al,255                         ;Fin de la table atteinte ?
        je   nosp                   ;OUI --> Pas caractère spécial
        cmp  ah,al                  ;Les codes correspondent-ils ?
        jne  sptest     ;NON --> Tester prochain élément de la table

        ;-- Le code était un caractère spécial ----------------------
        push cx                                     ;Ranger compteur
        mov  si,dx                                  ;Copier DX dans SI
        lodsb           ;Lire nombre de caractères du code de commande
        mov  cl,al          ;Transférer nombre de caractères dans CL
        rep  movsb                  ;Copier désignation dans buffer
        pop  cx                                     ;Retirer compteur
        pop  si                             ;Ramener SI de la pile
        mov  al,ah                          ;Copier caractère dans AL
        jmp  short hex                      ;Calculer code hexa

nosp:   pop  si                             ;Retirer SI de la pile
        mov  al,ah                          ;Copier caractère dans AL
        stosb                               ;Sauver dans le buffer

hex:    mov  al,ah                          ;Code du caractère dans AL
        and  ah,1111b               ;Masquer 4 bits du haut dans AH
        shr  al,1                   ;Décaler AL de 4 bits en tout
        shr  al,1                                   ;sur la droite
        shr  al,1
        shr  al,1
        or   ax,3030h               ;Convertir AH et AL en codes ASCII
        cmp  al,"9"                 ;AL représente-t-il une lettre ?
        jbe  nobal                          ;NON --> Pas de correction
        add  al,"A"-"1"-9                   ;Corriger AL
nobal:  cmp  ah,"9"                 ;AH représente-t-il une lettre ?
        jbe  hexout                         ;NON --> Pas de correction
        add  ah,"A"-"1"-9                   ;Corriger AH
hexout: mov  [si],ax                ;Placer code hexa dans le buffer
        add  si,3                           ;Désigner position suivante

        inc  bx                     ;Fixer pointeur sur prochain octet
        loop octetin                        ;Traiter octet suivant

        mov  al,219                         ;Fixer caractère spécial
        stosb

        mov  ax,LF shl 8 + CR               ;CR et LF terminent le buffer
        stosw                               ;Ecrire dans buffer

        ;-- Envoyer Dump sur périphérique de sortie standard ---------

        mov  bx,1                   ;Sortie standard porte le handle 1
        mov  cx,di                          ;Calculer nombre de caractères
        sub  cx,offset dumpBuf                      ;transférés
        mov  dx,offset dumpBuf              ;Adresse du buffer
        mov  ah,40h                 ;Code de fonction pour Ecrire handle
        int  21h                            ;Appeler fonction DOS
        jmp  dump                           ;Lire 9 octets suivants

findump label near

        mov  ax,4C00h               ;Numéro fonction pour terminer programme
        int  21h                    ;Terminer programme avec code de fin

;-- Données ------------------------------------------------------

neufoctets  db 9 dup (?)                    ;Les 9 octets lus chaque fois
dumpBuf     db 30 dup (?), 219                      ;Le buffer de sortie
            db 49 dup (?)

sptab   db NUL,BEL,BS,TAB           ;Table des caractères de commande
        db LF,CR,EOF,ESC
        db 255

sptext  equ this byte                       ;Textes des caractères spéciaux
        db 5,"<NUL>"                                        ;NUL
        db 5,"<BEL>"                                    ;Sonnerie
        db 4,"<BS> "                                    ;Backspace
        db 5,"<TAB>"                                   ;Tabulateur
        db 4,"<LF> "                                    ;Line Feed
        db 4,"<CR> "                                  ;Carriage Return
        db 5,"<EOF>"                                   ;Fin du fichier
        db 5,"<ESC>"                                        ;Escape

;-- Fin ----------------------------------------------------------

code    ends                                ;Fin du segment de CODE
        end  dump
```

19. Gestion de fichiers

En dehors des fonctions d'entrée et de sortie de caractères, les fonctions de gestion de fichiers sont certainement les fonctions les plus élémentaires qu'un système d'exploitation ait à offrir au programmeur. Celui qui programme dans un langage évolué tel que BASIC, Pascal ou C se trouve quelque peu désemparé puisque tous les langages évolués disposent de fonctions et instructions destinées spécialement à la gestion de fichiers.

Ce chapitre explicite tous les concepts liés aux fonctions de fichiers du DOS et aux diverses fonctions utiles pour l'accès aux fichiers.

19.1. Les deux visages du DOS

Dès qu'on évoque les fonctions de gestion de fichiers, on pense tout de suite aux fonctions :

- ✓ de création et suppression d'un fichier,
- ✓ d'ouverture et fermeture d'un fichier,
- ✓ de lecture d'un fichier et écriture dans un fichier.

Les systèmes d'exploitation aussi perfectionnés que le DOS comprennent cependant également une série d'autres fonctions de gestion de fichiers. Sous DOS, il s'agit de fonctions fournissant des informations particulières sur un fichier mais aussi de fonctions spéciales, par exemple pour renommer un fichier. Une des particularités du DOS est à cet égard que chacune de ces fonctions existe sous deux formes. Cette situation peut sembler très curieuse de prime abord mais elle s'explique par le double souci de compatibilité avec CP/M et avec UNIX, auquel nous avons déjà fait allusion. Il existe donc pour chaque fonction de fichier compatible avec CP/M une fonction équivalente compatible avec UNIX.

Origine des fonctions FCB

Toujours pour garantir la compatibilité avec les versions ultérieures, les fonctions tournées vers le CP/M n'ont jamais été éliminées du DOS et se rencontrent encore aujourd'hui. Les rares programmes qui les utilisent s'en servent dans des domaines bien particuliers. Microsoft continue toujours d'annoncer que ces fonctions ne seront plus reconnues dans les versions futures de DOS mais les quelques Ko logés dans le coeur du système d'exploitation ne pèsent plus très lourd.

Les fonctions compatibles avec CP/M sont appelées fonctions FCB car elles reposent sur une structure de données appelée FCB. Le nom FCB signifie ici File Control Block (Bloc de contrôle de fichier). Le DOS emploie cette structure de données, pendant le

travail sur un fichier, pour y stocker des informations importantes sur ce fichier. Bien que ce soit le DOS qui utilise en fait cette structure de données au premier chef, l'utilisateur doit réserver la place nécessaire pour cette structure à l'intérieur de son programme. Il peut ensuite accéder à travers ce FCB aux fonctions FCB telles que l'ouverture, la fermeture, l'écriture, la lecture, etc...

Comme les fonctions FCB ont été développées par souci de compatibilité avec les fonctions de CP/M et que CP/M ne dispose pas d'un système de fichiers hiérarchisé, les fonctions FCB ne soutiennent pas cette importante innovation de la version 2.0 du DOS. Il en résulte que les fonctions FCB permettent uniquement d'accéder aux fichiers placés dans le répertoire actuel.

Le concept des fonctions Handle

Ce problème n'existe pas pour les fonctions handle qui sont compatibles avec UNIX. Ces fonctions doivent leur nom au fait qu'elles utilisent comme code pour l'accès à un fichier donné une valeur numérique appelée handle en anglais. Ce code est transmis au programme d'appel à la suite de l'ouverture d'un fichier. Le DOS doit naturellement également stocker quelque part un certain nombre d'informations lors d'un accès de fichier orienté handle. Mais contrairement à ce qui se passe pour l'accès de fichier orienté FCB, ces informations sont stockées dans une structure de données mise en place dans une zone de mémoire occupée par le DOS et non dans une zone de mémoire fournie par le programme d'appel.

Pas de différence dans la structure de fichiers

Notez qu'il n'existe aucune différence dans la structure de fichiers des deux groupes de fonctions. Les fichiers créés et traités à l'aide d'une fonction FCB peuvent être ouverts, lus ou supprimés par la suite à l'aide d'une fonction Handle et inversement.

Lors de la programmation, il est important que vous vous décidiez pour l'un des deux groupes dans la mesure où vous ne programmez pas dans un langage évolué et que le compilateur ne vous accorde pas le contrôle.

Nous allons maintenant décrire séparément et en détail les différentes fonctions pour souligner les différences entre fonctions Handle et fonctions FCB.

19.2. Fonctions Handle

L'accès aux fichiers à travers les fonctions handle est beaucoup plus commode pour le programmeur que l'accès à travers les fonctions FCB car il n'a pas besoin de fournir au DOS une structure de données sous forme d'un FCB. De même que pour les fonctions

du système d'exploitation UNIX, l'accès à un fichier se fait en premier lieu à travers le nom de ce fichier. Ce nom est transmis à la fonction d'ouverture ou de création du fichier avant tout premier accès en lecture ou écriture, sous forme d'une chaîne de caractères ASCII. Cette chaîne peut comprendre non seulement le nom de fichier proprement dit mais aussi une désignation de périphérique, une indication de chemin complète ainsi qu'une extension de fichier. Elle doit se terminer par un caractère de fin (code ASCII 0).

Prise de possession d'un handle

Si le fichier n'a pu être ouvert ou créé, les deux fonctions renverront une valeur d'une largeur de 16 bits, appelée handle, que le programme devra noter dans ses tablettes. C'est en effet à travers ce handle que s'effectuera ensuite tout autre accès à ce fichier. Pour appeler la fonction de lecture par exemple, ce sera donc ce handle, et non le nom du fichier, qu'il faudra transmettre. Le terme de handle, qui ne signifie rien d'autre en français que poignée de porte, paraît donc particulièrement bien choisi pour désigner la fonction de ce code qui permet d'ouvrir la porte permettant d'accéder à un fichier.

Comme nous l'avons déjà indiqué, le DOS doit naturellement enregistrer au fur et à mesure un certain nombre d'informations sur ce fichier. Contrairement à ce qui se passe pour les fonctions FCB, cela ne se fait cependant pas dans une zone fournie par le programme mais dans des tables internes du DOS. La place mémoire prévue à cet effet n'est toutefois pas illimitée et le DOS, Nombre de fichiers ouvertsnombre de fichiers pouvant être ouverts simultanément, et donc du même coup le nombre de handles, est limité. Le nombre total de handles disponibles est fixé par un paramètre spécifié dans le fichier de configuration du système, le fichier CONFIG.SYS. Il s'agit de l'entrée :

```
FILES = X
```

X représente ici le nombre maximum de handles disponibles. La version 3.0 du DOS admet ici une valeur maximale de 255. L'utilisateur peut modifier cette entrée du fichier de configuration à l'aide d'un éditeur pour adapter cette valeur à ses besoins personnels. Toute modification ne s'applique cependant qu'à partir du prochain lancement du système car le DOS n'examine ce fichier qu'à l'occasion de l'opération de lancement initial du système, pour mettre en place les tables appropriées et configurer le système en fonction des paramètres définis dans ce fichier.

Le nombre total de handles est donc fixé par l'instruction Files dans le fichier de configuration mais le DOS n'autorise pas plus de 20 handles par programme. Il y a là une contradiction apparente qu'il convient d'expliquer :

Lorsqu'un programme ouvre par exemple 3 fichiers, 3 handles sont mis à sa disposition à cet effet. Le nombre de handles restant disponibles est donc réduit de 3. Si ce programme appelle un autre programme, les 3 fichiers qu'il avait ouverts restent ouverts. Si le nouveau programme ouvre alors à son tour d'autres fichiers, le nombre de handles

restants est encore réduit. Mais même s'il reste encore 30 handles disponibles, il ne peut s'en attribuer plus de 20. Il ne faut pas oublier non plus, à ce propos, que 5 handles sont déjà réservés, lors de l'appel d'un programme, aux périphériques tels que le clavier, l'écran et l'imprimante. 15 handles restent donc en fait seulement disponibles pour le programme.

Longueurs d'accès variables

Une autre différence par rapport aux fonctions FCB concerne les fonctions de lecture et écriture. Alors que les fonctions FCB travaillent avec une longueur d'enregistrement définie à l'intérieur du FCB, on doit indiquer chaque fois expressément aux fonctions handle combien d'octets doivent être lus ou écrits. Le travail avec des structures de données variables ou l'accès simultané à plusieurs enregistrements successifs est donc considérablement facilité. Outre les deux fonctions de lecture et écriture, le DOS offre aussi une fonction pour positionner le pointeur de fichier. Cette fonction n'attend pas non plus un numéro d'enregistrement mais bien le numéro de l'octet sur lequel doit se faire le prochain accès. Si on connaît le numéro de l'enregistrement et le nombre d'octets par enregistrement, une formule élémentaire permet aisément de calculer le numéro de l'octet sur lequel le pointeur de fichier doit être positionné :

```
Octet = numéro d'enregistrement * octets par enregistrement
```

Si toutefois vous voulez accéder au fichier de façon séquentielle, vous n'avez pas à utiliser cette fonction car le DOS fixe automatiquement le pointeur de fichier sur le premier octet du fichier lors de l'ouverture ou de la création d'un fichier. Tout accès ultérieur en lecture ou écriture décale ensuite le pointeur de fichier vers la fin du fichier, à raison du nombre d'octets lus ou écrits, de sorte que le prochain accès au fichier commence toujours à la suite du précédent.

Les fonctions Handle

Nous ne vous fournirons pas dans le présent chapitre une description détaillée des différentes fonctions de gestion des fichiers. Pour vous donner cependant une idée de ces fonctions, voici une petite liste :

Numéro	Fonction	Version DOS
3Ch	Créer ou vider fichier	(à partir de 2.0)
3Dh	Ouvrir fichier	(à partir de 2.0)
3Eh	Fermer fichier	(à partir de 2.0)
3Fh	Lire fichier	(à partir de 2.0)

Numéro	Fonction	Version DOS
40h	Inscrire dans fichier	(à partir de 2.0)
41h	Supprimer fichier	(à partir de 2.0)
42h	Déplacer pointeur de fichier	(à partir de 2.0)
43h/00h	Consulter attribut d'un fichier	(à partir de 2.0)
43h/01h	Fixer attribut d'un fichier	(à partir de 2.0)
45h	Doubler handle	(à partir de 2.0)
46h	Comparer handles	(à partir de 2.0)
56h	Renommer ou déplacer fichier	(à partir de 2.0)
57h/00h	Lire date et heure de la dernière modification d'un fichier	(à partir de 2.0)
57h/01h	Régler date et heure de la dernière modification d'un fichier	(à partir de 2.0)
5Ah	Créer fichier temporaire	(à partir de 2.0)
5Bh	Créer nouveau fichier	(à partir de 3.0)
5Ch/00h	Interdire l'accès à une zone de fichier	(à partir de 3.0)
5Ch/01h	Libérer une zone de fichier verrouillée	(à partir de 3.0)
6Ch	Fonction OPEN étendue	(à partir de 4.0)

Utilisation des fonctions handle

Il convient d'observer, pour appeler ces fonctions, un certain nombre de règles générales, que nous allons évoquer brièvement :

Les fonctions qui attendent comme argument un nom de fichier ou l'adresse d'un nom de fichier (par exemple Créer fichier et Ouvrir fichier) attendent l'adresse de segment du nom dans le registre DS et l'adresse d'offset de ce nom dans le registre DX. Si les fonctions renvoient un handle après avoir été appelées, celui-ci est contenu dans le registre AX.

Les fonctions qui attendent par contre un handle comme argument l'attendent dans le registre BX. Après avoir été appelées, elles indiquent à travers le flag Carry si une erreur est apparue lors de leur exécution. Si c'est le cas, le flag Carry est mis et le registre AX contient un code d'erreur qui indique les causes de l'erreur ou bien l'erreur elle-même.

La fonction 59h de l'interruption du DOS 21h permet d'obtenir des informations très détaillées sur toute erreur apparue lors d'une opération disquette ou lors de l'appel d'une autre fonction. Cette fonction n'est cependant disponible qu'à partir de la version 3.0 du DOS.

19.3. Fonctions FCB

Comme nous l'avons déjà expliqué au chapitre précédent, le DOS a recours pour la manipulation des fichiers à une structure de données appelée FCB. Le programmeur peut d'ailleurs tirer parti de cette structure de données tout autant que le DOS, soit en lisant certaines informations que cette structure peut lui fournir sur le fichier voulu, soit en manipulant lui-même ces informations. Il convient donc que nous nous intéressions tout d'abord à la structure d'un FCB, avant de décrire les différentes fonctions FCB.

Structure d'un FCB

Le FCB est une structure de données d'une longueur de 37 octets qui est divisée en champs de différentes tailles. La figure suivante présente ces différents champs :

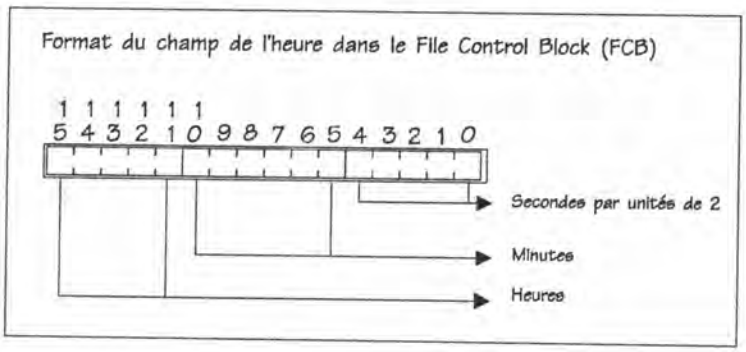
Structure d'un File Control Block (FCB)		
Adresse	**Contenu**	**Type**
+00h	Numéro de périphérique	1 BYTE
+01h	Nom du fichier (comblé avec des espaces)	8 BYTES
+09h	Extension de fichier (comblée avec espaces)	3 BYTES
+0Ch	Numéro de bloc actuel	1 WORD
+0Eh	Taille d'enregistrement	1 WORD
+10h	Taille du fichier	1 DWORD
+14h	Date de la dernière modification	1 WORD
+16h	Heure de la dernière modification	1 WORD
+18h	Réservé	8 BYTES
+20h	Numéro de l'enregistrement actuel	1 BYTE
+21h	Numéro d'enregistrement pour l'accès sélectif	1 DWORD
Longueur : 37 octets		

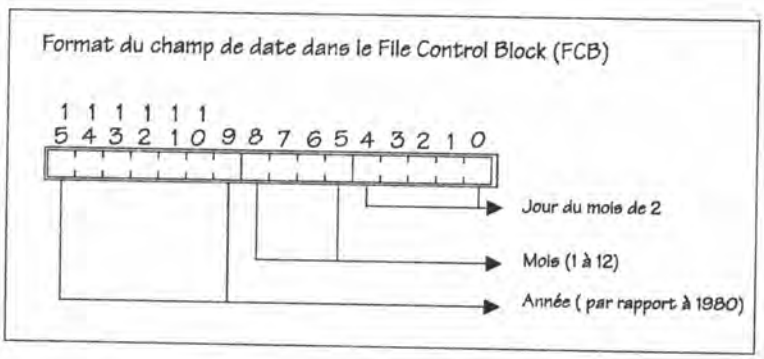
Parmi bien d'autres informations, le FCB doit naturellement contenir le nom du fichier appelé. Il est stocké dans les premiers champs du FCB, de la cellule de mémoire 00h à la cellule de mémoire 0Bh. L'octet dans la cellule de mémoire 00h indique le nom du périphérique sur lequel figure le fichier ou sur lequel il doit être créé. 0 désigne ici le lecteur actuel, 1 le lecteur A, 2 le lecteur B, etc... Le nom de fichier proprement dit est inscrit sous forme d'une chaîne de caractères ASCII à partir de la cellule de mémoire 01h. Ce champ a une taille de 8 octets et il ne peut donc contenir qu'un nom de fichier, sans indication de chemin.

C'est d'ailleurs la raison essentielle pour laquelle les fonctions FCB ne permettent d'accéder qu'aux fichiers situés dans le répertoire actuel. Si le nom de fichier comporte moins de 8 caractères, le reste de ce champ doit être rempli d'espaces (code ASCII 32). Il en va de même pour le champ suivant qui contient l'extension du nom de fichier, d'une longueur maximale de 3 caractères. Le champ suivant, à partir de la cellule de mémoire 0Ch, contient un mot qui indique pendant un accès de fichier séquentiel le numéro du bloc actuellement traité. En liaison avec le numéro d'enregistrement actuel (cellule de mémoire 20h), ce champ sert à l'accès séquentiel, en lecture et écriture, à un fichier.

Le champ à partir de la cellule de mémoire 0Eh reçoit la longueur d'un enregistrement. Celle-ci peut être comprise entre 1 et 65 536 et elle indique le nombre de caractères que le DOS considérera comme formant un registre et qu'il transférera simultanément lors de chaque opération de lecture ou écriture. Ces informations sont suivies de la longueur du fichier en octets, qui est stockée à partir de la cellule de mémoire 10h sous forme DWORD.

Les informations sur la date de la dernière modification du fichier sont stockées à partir de la cellule de mémoire 14h. Vous y trouvez d'ailleurs également l'heure de cette dernière modification. Ces deux informations sont stockées sous une forme codée.





Vient ensuite une zone de données de 8 octets qui est à la disposition exclusive du DOS et qui ne doit pas être modifiée par le programmeur. La signification des différents octets n'est pas fournie de manière précise par la société Microsoft et elle varie d'une version du DOS à l'autre.

La zone de données réservée est suivie du numéro d'enregistrement actuel que nous avons déjà évoqué à propos du numéro de bloc actuel.

Le dernier champ du FCB est utilisé dans le cadre de l'accès de fichier sélectif. Ce mode d'accès consiste à ne pas appeler les enregistrements successivement, dans leur ordre logique à l'intérieur du fichier, mais en les manipulant dans n'importe quel ordre. C'est ce qu'on appelle aussi les fichiers relatifs.

Bien que ce champ ait une taille de 4 octets, seuls les 3 premiers octets sont utilisés pour spécifier le numéro d'enregistrement actuel si la longueur d'enregistrement est supérieure ou égale à 64 octets. Si elle est inférieure à 64 octets, les 4 octets de ce champ seront utilisés.

FCB étendu

Outre ce FCB "normal", le DOS soutient également ce qu'on appelle les FCB étendus. Ces derniers, contrairement aux FCB normaux, permettent d'accéder également aux fichiers dotés d'attributs spéciaux, tels que les fichiers cachés ou les fichiers système. Ils permettent aussi d'accéder aux noms de disquette et de disque dur ainsi qu'aux sous-répertoires (ce qui ne signifie cependant pas qu'on puisse accéder par leur intermédiaire à des fichiers situés dans d'autres répertoires que le répertoire actuel).

Ils sont construits comme un FCB normal mais avec une extension de 7 octets. Ces 7 octets, qui portent à 44 octets la taille du FCB, sont placés au début du FCB. Les champs suivants se trouvent donc décalés de 7 octets par rapport à un FCB normal.

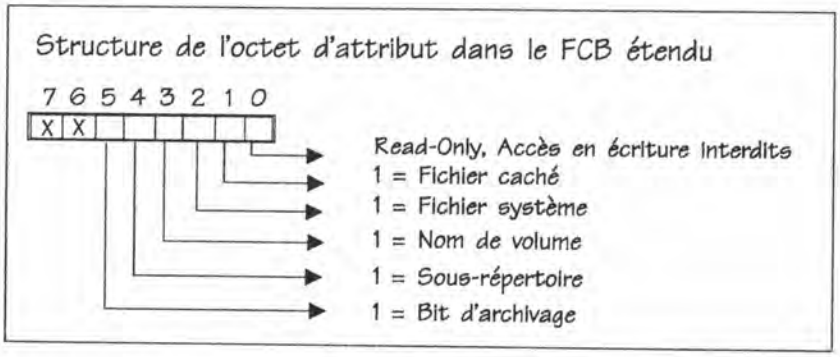
Structure d'un File Control Block étendu (FCB étendu)

Adresse	Contenu	Type
+00h	Marque d'un FCB étendu (FFh)	1 BYTE
+01h	Réservé	5 BYTES
+06h	Attribut de fichier	1 BYTE
+07h	Numéro de périphérique	1 BYTE
+08h	Nom de fichier (comblé avec des espaces)	8 BYTES
+10h	Extension de fichier (comblée avec espaces)	3 BYTES
+13h	Numéro de bloc actuel	1 WORD
+15h	Taille d'enregistrement	1 WORD

Structure d'un File Control Block étendu (FCB étendu)		
Adresse	**Contenu**	**Type**
+17h	Taille du fichier	1 DWORD
+1Bh	Date de la dernière modification	1 WORD
+1Dh	Heure de la dernière modification	1 WORD
+1Fh	Réservé	8 BYTES
+27h	Numéro de l'enregistrement actuel	1 BYTE
+28h	Numéro d'enregistrement pour l'accès sélectif	1 DWORD
Longueur : 44 octets		

Le premier octet d'un FCB étendu contient toujours la valeur 255 qui sert à identifier les FCB étendus. Cette cellule de mémoire contient en effet, dans un FCB normal le numéro de périphérique qui ne peut jamais être 255. Le DOS peut ainsi distinguer à coup sûr entre un FCB normal et un FCB étendu. Cette distinction nette est la raison essentielle qui permet au DOS d'accepter aussi bien les FCB normaux que les FCB étendus lors de tout appel d'une fonction orientée FCB. Ce code d'identification est suivi de 5 octets qui sont réservés à l'usage exclusif du DOS. Le programmeur ne doit pas modifier leur contenu.

Le dernier octet d'extension par rapport à un FCB normal est enfin, dans la cellule de mémoire 06h, l'attribut du fichier qui est en fait au centre du débat. La figure suivante décrit la structure de cet octet où chaque bit correspond à un attribut précis. Les différents attributs s'excluent mutuellement tels par exemple le nom de l'unité et le sous-répertoire puisqu'un sous-répertoire ne peut pas représenter un nom d'unité et inversement.



Définition d'un FCB

Notre programme doit en premier lieu mettre une zone de mémoire à la disposition du système, pour le FCB à créer, car le DOS ne se charge pas lui-même de ce travail. Dans le cas d'un FCB normal, cette zone devra s'étendre sur 37 octets contre 44 pour un FCB étendu. Vous avez la possibilité de réserver cette zone de mémoire directement à l'intérieur du segment de données de notre programme ou bien de la faire réserver à l'aide d'une fonction du DOS.

Il faut ensuite inscrire dans le FCB le nom du fichier auquel il s'agit d'accéder. Vous pouvez bien sûr écrire cette information directement dans les différentes cellules de mémoire du FCB mais nous vous conseillons chaque fois que possible de n'accéder aux différents champs du FCB qu'à travers les fonctions appropriées du DOS. Cela permet en effet une programmation plus claire et cela vous dispense en même temps d'avoir chaque fois à vous demander s'il s'agit d'un FCB normal ou d'un FCB étendu. Vous risqueriez sinon trop aisément d'accéder à un champ erroné du FCB, ce qui risquerait dans le pire des cas d'entraîner une destruction irréversible du contenu d'un fichier.

Inscription d'un nom de fichier dans le FCB

Mais revenons à notre nom de fichier. Vous pouvez l'inscrire dans les champs du FCB prévus à cet effet à l'aide d'une fonction spéciale du DOS, la fonction 29h de l'interruption 21h. Cette fonction attend l'adresse (de départ) du FCB dans la paire de registres ES:DI et l'adresse du nom de fichier dans la paire de registres DS:SI. Ce nom de fichier doit figurer sous forme d'une chaîne de caractères ASCII terminée par une marque de fin (code ASCII 0). Ce nom peut comporter une extension et une désignation de périphérique (ces deux informations sont toutefois facultatives) ainsi que ce qu'on appelle les jokers, à savoir l'étoile ou le point d'interrogation.

Le registre AH doit en outre contenir le numéro de fonction et le registre AL certaines informations sur la transposition du nom de fichier. Ces dernières sont représentées par les bits inférieurs de ce registre. Nous indiquerons leur signification précise lorsque nous en viendrons à la description détaillée des fonctions (Annexe C). Vous pouvez donc vous contenter, pour vos premiers essais, d'attribuer systématiquement la valeur 0 au registre AL.

Une fois le nom de fichier correctement inscrit dans le FCB, le fichier (et donc du même coup le FCB) peut être ouvert à travers une fonction DOS. Cette opération consiste pour le DOS à inscrire dans le FCB des informations importantes sur le fichier, informations tirées en partie de l'entrée du fichier dans le répertoire. Il s'agit notamment, par exemple, de la taille du fichier et des date et heure de dernière modification. Dès cet instant, le FCB peut être considéré comme ouvert.

Par souci de compatibilité avec CP/M, le DOS fixe la longueur d'enregistrement, lors de l'ouverture du FCB, sur 128 octets. Si c'est toutefois une autre longueur d'enregistrement qui est censée être utilisée, la longueur voulue doit être inscrite, en octets, dans le champ approprié du FCB. Notez bien à cet égard que la longueur d'enregistrement doit être inscrite dans ce champ seulement après et non avant l'ouverture du FCB. Elle serait sinon remplacée par la longueur d'enregistrement standard de 128 octets.

Configuration de la DTA

Si le programmeur spécifie une longueur d'enregistrement supérieure à 128, il doit déplacer la DTA avant le premier accès en lecture ou écriture au fichier. Cette zone de mémoire appelée zone de transfert de données (en anglais : Disk Transfer Area) sert de buffer de données au DOS pour tous les accès de fichier orientés FCB. Le DOS met en place ce buffer lors de l'appel de tout programme, à l'intérieur du PSP (Program Segment Prefix) qui précède chaque programme. Ce buffer a cependant une taille de 128 octets seulement. Tout accès à un fichier dont la longueur d'enregistrement excède 128 octets aurait donc pour conséquence que le DOS effacerait le début du programme. Cela pourrait bien sûr avoir des conséquences catastrophiques et entraîner un plantage du système.

Pour éviter cela, il faut attribuer une nouvelle zone de mémoire, plus importante, à la DTA. Comme pour les FCB, il est possible de réserver cette zone de mémoire directement dans le segment de données ou bien de se faire attribuer la place mémoire nécessaire à l'aide d'une fonction du DOS. Cette zone de mémoire doit avoir la même taille que la longueur d'enregistrement pour l'accès de fichier séquentiel. Pour l'accès de fichier sélectif (fichiers relatifs), la taille de cette zone dépend du nombre maximum d'enregistrements que vous voulez faire lire à la fois par le DOS lors de chaque appel de fonction. Il est en effet possible dans ce cas de lire plusieurs enregistrements simultanément. La taille nécessaire sera alors le produit du nombre maximum d'enregistrements lus par la longueur d'enregistrement.

Au point où nous en sommes, le DOS ne sait toujours pas que la DTA a été transférée dans une autre zone de la mémoire. Il faut donc le lui indiquer en appelant la fonction 1Ah de l'interruption 21h. L'adresse de la nouvelle DTA doit être transmise à travers les registres DS (segment) et DX (offset). Il n'est pas nécessaire d'indiquer au DOS la taille du buffer car le DOS considère systématiquement que le buffer a une taille suffisante pour recevoir les données transmises.

Edition d'un fichier

Nous pouvons maintenant accéder au fichier. Avec l'accès de fichier séquentiel, on commencera obligatoirement par traiter le premier enregistrement du fichier. Les accès

suivants concerneront ensuite successivement les enregistrements 2, 3, 4, c'est-à-dire les différents enregistrements dans l'ordre.

Le DOS actualise au fur et à mesure les informations contenues dans le FCB, comme par exemple la taille du fichier et le pointeur de fichier qui désigne l'adresse du prochain enregistrement à l'intérieur du fichier.

En cas d'accès direct (relatif) au fichier, il est possible d'accéder aux différents enregistrements dans n'importe quel ordre. Cela implique naturellement que le numéro de l'enregistrement voulu soit communiqué au DOS avant tout accès au fichier en lecture ou écriture. Le numéro de l'enregistrement doit être à cet effet inscrit dans le dernier champ du FCB. Il est cependant possible de passer, pendant la durée de l'accès à un fichier, du mode séquentiel d'accès au mode direct et inversement car le mode d'accès dépend exclusivement de chaque appel de fonction. Il existe en effet deux séries de fonctions pour l'accès séquentiel et pour l'accès direct.

Utilisation des fonctions FCB

Il ne nous est toutefois pas possible de décrire les différentes fonctions FCB dans le cadre de ce chapitre. Voici cependant une liste de toutes les fonctions FCB avec les numéros de fonction correspondants (par rapport à l'interruption 21h).

Numéro	Fonction
0Fh	Ouvrir fichier
10h	Fermer fichier
13h	Supprimer fichier
14h	Lecture séquentielle
15h	Ecriture séquentielle
16h	Créer fichier
17h	Renommer fichier
1Ah	Fixer l'adresse DTA
21h	Lecture sélective (d'un enregistrement)
22h	Ecriture sélective (d'un enregistrement)
23h	Déterminer taille du fichier
24h	Fixer le numéro d'enregistrement pour l'accès direct
27h	Lecture sélective (d'un ou plusieurs enregistrements)
28h	Ecriture sélective (d'un ou plusieurs enregistrements)
29h	Inscription du nom de fichier dans le FCB

Il convient d'indiquer ici quelques règles de base qui régissent l'appel de ces fonctions :

Les fonctions FCB permettent naturellement d'accéder à plusieurs fichiers. Il existe dans ce cas un FCB particulier pour chacun. Pour indiquer au DOS auquel de ces fichiers il s'agit d'accéder à l'aide d'une fonction FCB, chaque fonction FCB attend l'adresse du FCB attribué au fichier concerné. L'adresse de segment doit être placée dans le registre DS et l'adresse d'offset dans le registre DX. Après avoir été exécutées, la plupart de ces fonctions renvoient dans le registre AL une valeur qui indique si la fonction a pu être exécutée sans erreur. Si c'est bien le cas, c'est la valeur 0 qui sera renvoyée. En cas d'erreur, les fonctions de création, suppression, ouverture et fermeture d'un fichier renvoient la valeur 255 alors que les fonctions de lecture et écriture d'enregistrements renvoient un code d'erreur qui permet d'identifier les différentes sources et causes d'erreur possibles. La fonction 59h fournit des informations très détaillées sur toute erreur apparue lors d'une opération disquette. Cette fonction n'est cependant disponible qu'à partir de la version 3.0 du DOS.

19.4. Handles contre FCB

Après vous avoir présenté les principes à la base des deux groupes de fonctions que le DOS met à votre disposition, essayons de passer rapidement en revue les avantages et inconvénients des différentes fonctions. Comme nous l'avons déjà indiqué, le choix ira de soi pour tous ceux qui veulent transposer sur le DOS un programme du système d'exploitation CP/M ou UNIX. Notre objet est donc ici d'aider ceux qui veulent développer un nouveau programme sous le DOS à prendre leur décision.

Deux avantages essentiels des fonctions handle, qui constituent du même coup des inconvénients des fonctions FCB, sautent immédiatement aux yeux. Il s'agit d'une part de la possibilité d'accéder à l'aide des fonctions handle aux fichiers situés dans n'importe quel répertoire de la mémoire de masse. D'autre part, il n'est pas nécessaire avec les fonctions handle de prévoir dans le programme utilisateur la mémoire requise pour chaque accès de fichier orienté FCB.

Il convient cependant de prendre également en compte un certain nombre d'aspects qui pour être moins évidents n'en sont pas moins tout aussi importants pour certaines applications. C'est ainsi par exemple que seul un FCB étendu (qui n'est toutefois plus compatible avec CP/M) permet d'accéder au nom d'une mémoire de masse. En ouvrant un FCB, on accède très facilement à certaines informations telles que la taille du fichier ou la date de la dernière modification. Sous les versions 1.0 ou 2.0 du moins, les FCB présentent l'avantage, pour les programmes qui ont besoin d'accéder simultanément à un grand nombre de fichiers, de permettre l'ouverture simultanée d'un nombre de fichiers illimité. Les fonctions handle présentent par contre l'avantage pour le programmeur de le dispenser de prévoir une zone de transfert de données. Avec les fonctions handle, l'accès de fichier peut en outre être détaché de la taille d'enregistrement qui joue un si grand rôle dans les fonctions FCB.

Comme vous le voyez de nombreux arguments plaident pour ou contre chacun des deux groupes de fonctions. Dans la perspective du développement futur du DOS, une tendance très nette se dessine toutefois vers un soutien accru des fonctions handle. On peut donc envisager que les fonctions FCB disparaissent un jour ou l'autre du jeu de fonctions du DOS.

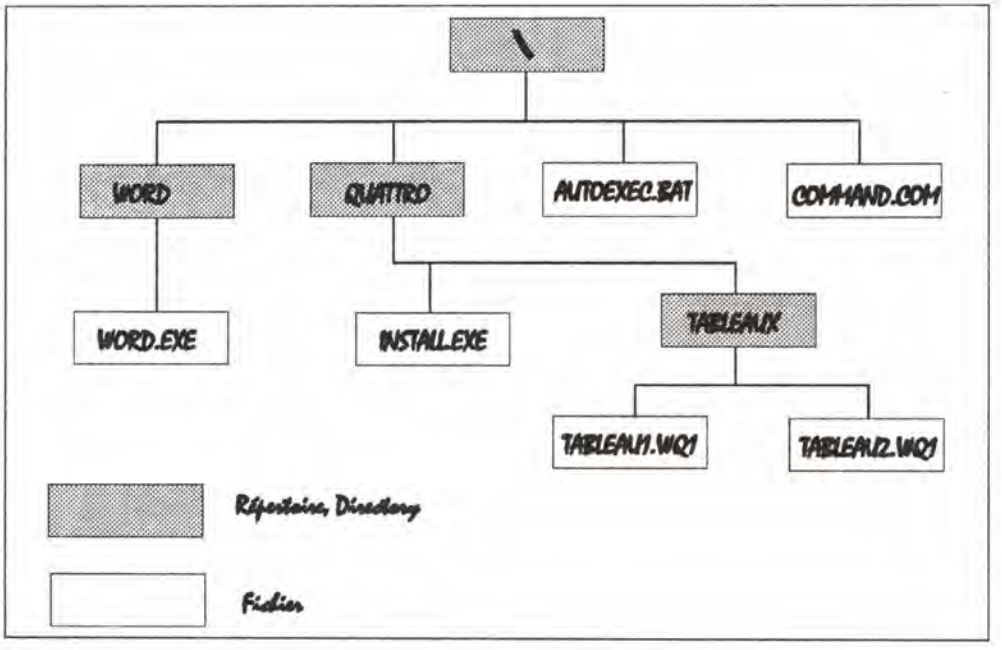
20. Accès aux répertoires et lecteurs

Les fonctions du DOS qui concernent les répertoires se divisent en deux groupes. Il y a d'abord les fonctions qui permettent de manipuler des sous-répertoires et ensuite d'autres fonctions qui servent à rechercher des fichiers dans des zones déterminées d'une mémoire de masse.

Nous traiterons donc ces deux groupes de fonctions séparément, dans ce chapitre, en commençant par la manipulation des sous-répertoires.

20.1. Gestion des répertoires

Peut-être savez-vous qu'il existe, à partir de la version 2.0 du DOS, la possibilité d'attribuer à chaque répertoire d'une unité de mémoire de masse, en partant du répertoire racine, plusieurs sous-répertoires, de façon à former une sorte d'arbre de répertoires.



Un arbre de répertoires

Le nombre et le nom des sous-répertoires d'un répertoire ne sont pas définis de manière fixe par le DOS. L'utilisateur doit donc avoir la possibilité d'ajouter des répertoires à la ramification des répertoires. Il doit aussi pouvoir supprimer des répertoires (à condition que ceux-ci ne contiennent plus de fichiers). Il faut enfin qu'un répertoire puisse être

déclaré comme étant le répertoire actuel. Le DOS supposera alors que font partie de ce répertoire tous les fichiers qui seront appelés en indiquant seulement le nom de fichier au sens strict, sans indication de chemin complète.

Au niveau utilisateur, ces fonctions sont offertes par le DOS sous la forme des instructions MD (Créer répertoire), RD (Supprimer répertoire) et CD (Changer le répertoire actuel). De façon interne, ces instructions correspondent aux fonctions 39h, 3Ah et 3Bh de l'interruption 21h. Ce sont ces fonctions que nous allons donc décrire maintenant.

Fonction	Tâche
0Eh	Sélection du lecteur en cours
19h	Déterminer le lecteur en cours
39h	Créer un répertoire
3Ah	Supprimer un répertoire
3Bh	Commuter vers le répertoire en cours

Appel de fonctions

Le rôle des registres lors de l'appel et après retour de ces fonctions est largement uniforme :

Il faut chaque fois transmettre le numéro de fonction dans le registre AH ainsi que le nom de chemin du répertoire à manipuler. La spécification du chemin peut comprendre naturellement la désignation complète du chemin mais aussi un nom de périphérique (placé en premier et suivi d'un double point) et elle doit se terminer par une marque de fin (Code ASCII 0).

Si le nom de périphérique est omis, le DOS considérera que l'appel de fonction concerne le lecteur actuel.

Chaque fonction attend l'adresse de la spécification de chemin dans la mémoire dans les registres DS (adresse de segment) et DX (adresse d'offset). Après exécution, toutes les fonctions indiquent par un flag Carry annulé que l'exécution s'est effectuée sans problème. En cas d'erreur, le flag Carry sera au contraire mis et un code d'erreur sera fourni dans le registre AX.

Création de répertoires

La fonction 39h sert à créer un répertoire. Le chemin qu'elle attend doit comprendre comme dernier élément le nom du nouveau répertoire. Une erreur peut se produire si un ou plusieurs des répertoires indiqués dans la spécification de chemin n'existent pas ou bien si le répertoire à créer existe déjà. Une autre source d'erreur tient au fait que le nombre de fichiers dans le répertoire racine est limité. Or le DOS considère en quelque sorte les sous-répertoires comme une certaine forme de fichiers. Il peut donc arriver que la création d'un sous-répertoire dans le répertoire racine soit refusée parce que la racine est déjà limitée.

Suppression de répertoires

C'est la fonction 3Ah qui sert à supprimer un répertoire. Le DOS peut refuser d'exécuter cette fonction dans différents cas de figure, par exemple si le répertoire à supprimer n'est pas vide. Vous ne pouvez en effet supprimer un répertoire tant qu'il contient encore des fichiers ou à plus forte raison d'autres sous-répertoires. Il ne faut pas non plus que le répertoire à supprimer soit le répertoire actuel.

Sélection du répertoire et lecteur actuel

Lorsque vous voulez fixer le sous-répertoire actuel avec la fonction 3Bh, vous devez veiller à ce que tous les noms de chemin indiqués dans la spécification de chemin existent effectivement. C'est d'ailleurs la seule source d'erreur possible pour cette fonction.

S'il s'agit de redéfinir non seulement le répertoire actuel mais aussi le périphérique actuel, vous pouvez utiliser à cet effet la fonction 0Eh. Elle attend simplement, outre naturellement le numéro de fonction dans le registre AH, le code de périphérique du nouveau lecteur actuel dans le registre DL. Le code 0 correspond ici au lecteur A, 1 à B, 2 à C, etc...

Déterminer le répertoire et lecteur actuel

Avant de définir le répertoire actuel à travers la fonction 3Bh, il est parfois nécessaire de déterminer quel est momentanément le répertoire actuel. Ici aussi, le DOS offre au programmeur une fonction très pratique, la fonction 47h. Elle peut fournir le chemin du répertoire actuel non seulement pour le périphérique actuel mais aussi pour tous les périphériques. Il faut donc lui communiquer également le périphérique concerné par l'appel de la fonction. S'il s'agit du périphérique actuel, il faut transmettre à la fonction la valeur 0 dans le registre DL.

Pour tous les autres périphériques, on aura les valeurs 1 (lecteur A), 2 (B), 3 (C), etc...

La fonction attend cependant aussi, en dehors du code de périphérique, l'adresse d'un buffer de 64 octets que vous devez réserver à l'intérieur de votre programme. Le registre DS recevra l'adresse de segment et le registre SI l'adresse d'offset de ce buffer. Après appel de la fonction, ce buffer contiendra la spécification de chemin du répertoire actuel, qui sera terminée par une marque de fin (Code ASCII 0). Il convient cependant de noter à cet égard que la spécification de chemin n'est précédée ni du nom de périphérique approprié ni du caractère "\". Si le répertoire actuel est la racine, le buffer contiendra donc comme premier (et comme dernier) caractère la marque de fin. Si un code de périphérique inconnu du DOS a été transmis lors de l'appel de fonction, le flag Carry sera mis après appel de la fonction et le registre AX contiendra le code d'erreur 0Fh.

Outre le répertoire actuel, vous souhaitez certainement déterminer aussi le lecteur actuel. Utilisez à cet effet la fonction 19h qui attend uniquement le numéro de fonction dans le registre AH en guise de paramètres. L'évaluation des informations retournées s'effectue tout aussi simplement que l'appel de la fonction. Il suffit d'évaluer le registre AL après l'appel de la fonction. Il contient le code de périphérique du lecteur en cours où 0 représente le lecteur A, 1 le lecteur B, 2 le C et ainsi de suite.

Mais notez toutefois que la procédure à suivre ici est différente de celle de la fonction 0Eh où 0 représente le lecteur en cours et la valeur 1 correspond au lecteur A.

20.2. Recherche de fichiers

Intéressons-nous maintenant à la recherche d'un ou plusieurs fichiers dans le répertoire actuel sur le périphérique actuel. Ici réapparaît l'opposition entre fonctions handle et FCB. Il existe en effet deux groupes de fonctions pour la recherche de fichiers. Le groupe des fonctions FCB présente l'inconvénient de ne permettre de rechercher des fichiers que dans le répertoire actuel d'un périphérique déterminé, alors que les fonctions handle permettent de rechercher des fichiers dans tous les répertoires des différents périphériques. Le terme de fonctions handle est d'ailleurs assez impropre en l'occurrence puisque ces fonctions ne sont pas appelées à travers un handle. Ce terme se justifie plutôt par le fait que ces fonctions ont été intégrées dans le jeu de fonctions du DOS avec la version 2.0, à l'occasion de l'introduction des sous-répertoires (et donc du même coup des fonctions handle). Les fonctions FCB étaient au contraire disponibles dès la version 1.0.

Fonctionnement des fonctions de recherche

Bien que les fonctions Handle et FCB présentent des différences dans le mode de recherche de fichiers dans les répertoires "étrangers", elles comportent toutefois des similitudes. Dans les deux cas, deux fonctions entrent en jeu. Elles sont désignées par

FindFirst et FindNext. FindFirst n'est appelée qu'une seule fois car elle lance la recherche. Elle attend par conséquent le nom du fichier à rechercher ainsi que l'attribut de fichier.

En ce qui concerne le nom recherché, vous pouvez préciser un nom bien spécifique (exemple, "COURRIER.TXT" ou "C:\DOS\COURRIER.TXT") ou utiliser les caractères génériques * et ? pour rechercher plusieurs fichiers correspondant au modèle indiqué. Les caractères génériques permettent d'utiliser FindFirst et FindNext pour afficher tous les fichiers d'un répertoire, tout comme la commande DIR.

FindFirst retourne seulement le premier nom de fichier trouvé indépendamment du nom de recherche à condition qu'un fichier correspondant au nom ou au modèle de nom spécifié soit trouvé dans le répertoire désigné. Tous les autres fichiers peuvent alors être déterminés par des appels successifs de FindNext où à chaque appel cette fonction retourne le nom de fichier suivant correspondant au modèle.

Fonction	Tâche
11h	FindFirst (FCB)
12h	FindNext (FCB)
4Eh	FindFirst (Handle)
4Fh	FindNext (Handle)

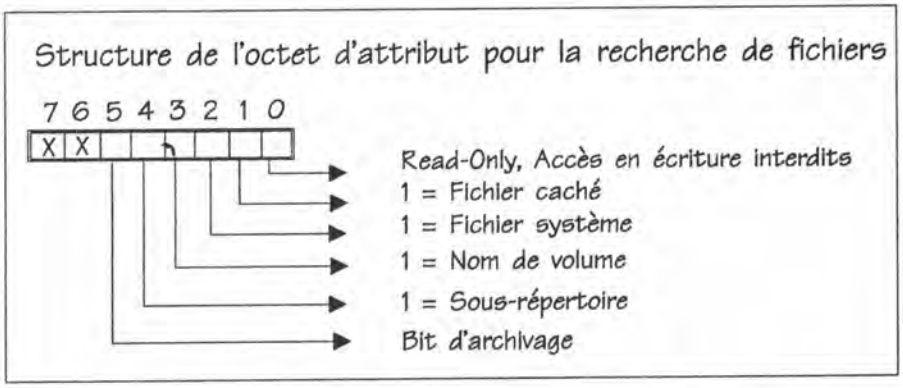
Le rôle de l'attribut de fichier

Toutes les fonctions utilisent les attributs standard tels qu'ils sont représentés dans la figure suivante. L'octet d'attribut renferme les différents bits décrivant des attributs bien précis. Ainsi, le bit 4 est par exemple réglé lorsqu'une entrée de répertoire n'est pas un fichier mais un sous-répertoire. Quant aux bits 2 et 3, ils sont réglés lorsqu'il s'agit d'un fichier système caché que DOS n'affiche pas lorsque la commande DIR est appelée.

Dès qu'une entrée est trouvée, l'octet d'attribut peut servir à déterminer tous les attributs définis au-dessous et contrôler ainsi le type de l'entrée. Ils jouent d'ailleurs un rôle important dès le début de la recherche. Lors de l'appel, les deux versions de la fonction FindFirst attendent la transmission d'un "attribut de recherche" spécifiant les fichiers à rechercher. Dans ce cas, l'attribut Read Only et le bit d'archivage ne jouent aucun rôle. Si la valeur 0 a été par exemple précisée comme attribut de recherche, vous obtenez tous les fichiers normaux, qu'ils soient munis de l'attribut Read Only ou que leur bit d'archivage soit réglé.

La procédure est tout à fait différente en ce qui concerne les entrées de répertoire décrivant des fichiers cachés, des fichiers système, des noms d'unités et des sous-répertoires. Ils sont exclus de la recherche de fichiers tant que le bit correspondant n'est pas

réglé dans l'attribut de recherche. Réglez par exemple le bit 4 dans l'attribut de recherche et vous constatez que les sous-répertoires sont également retournés.



Cette méthode ne permet toutefois pas d'exclure tous les fichiers normaux de la recherche, de ne rechercher en quelque sorte que les fichiers cachés ou les sous-répertoires. Pour vous aider, vous pouvez évaluer les attributs des fichiers trouvés et ne considérer que les fichiers dont le flag d'attribut est réglé.

Examinons maintenant les fonctions FCB puis les fonctions Handle.

20.2.1. Recherche de fichiers avec les fonctions FCB

Pour rechercher des fichiers avec les fonctions FCB, on appelle essentiellement les deux fonctions 11h et 12h. Elles permettent de rechercher non seulement un fichier dont le nom est défini précisément mais aussi tous les fichiers ou encore tous les fichiers dotés d'une extension déterminée. La fonction 11h sert à rechercher le premier fichier à l'intérieur du répertoire actuel, alors que la fonction 12h recherche tous les fichiers suivants. Les FCB jouent un rôle essentiel dans cette opération.

C'est en effet à travers eux que s'effectue l'échange de données entre le programme d'appel et les deux fonctions. Essayons donc d'abord de comprendre comment se déroule l'opération de recherche de fichiers dans un répertoire :

Le programme doit tout d'abord réserver la place nécessaire pour deux FCB, soit en réservant la place mémoire voulue dans la zone de données du programme, soit en réclamant au DOS la place mémoire correspondante à l'aide de la fonction 48h. Le programmeur est libre de travailler ici avec des FCB normaux ou étendus. Ces derniers présentent toutefois par rapport aux FCB normaux l'avantage de permettre également de rechercher les fichiers dotés d'attributs particuliers (système ou caché) ainsi que le nom (Volume) de la disquette ou du disque dur et des sous-répertoires. Les deux fonctions acceptent donc aussi bien des FCB normaux qu'étendus. L'un des deux FCB sert à recevoir le nom du fichier qu'il s'agit de rechercher alors que le DOS inscrit dans l'autre

le nom du ou des fichiers trouvés. Pour pouvoir distinguer les deux FCB dans les explications qui vont suivre, nous les désignerons sous les noms de FCB_recherché et FCB_trouvé.

Spécifier l'adresse FCB

L'adresse du FCB_trouvé doit être communiquée au DOS à l'aide de la fonction 1Ah. En appelant cette fonction, vous fixez le FCB_trouvé comme la nouvelle zone de transfert de données (DTA). Cette zone joue cependant un rôle important non seulement pour ces deux fonctions mais aussi pour toutes les fonctions servant à la transmission des données entre l'ordinateur et le disque. C'est pourquoi il est conseillé de déterminer l'adresse de la DTA actuelle, à l'aide de la fonction 2Fh, avant d'activer la nouvelle DTA. Une fois la recherche de fichiers terminée, la DTA pourra être alors réinstallée dans son ancienne position à l'aide de la fonction 1Ah.

Traiter le FCB_trouvé

Après que la DTA ait été placée sur le FCB_trouvé, la prochaine étape consiste à inscrire le nom du fichier à rechercher dans le FCB_recherché. Si vous ne voulez pas que la recherche se limite à un fichier bien précis, vous pouvez insérer ce qu'on appelle des jokers ('*' et '?') dans le nom de fichier. Ce dernier peut être inscrit directement dans le FCB_recherché ou bien à l'aide de la fonction 29h. Si vous voulez par exemple faire rechercher tous les fichiers, vous devrez spécifier le nom de fichier '*.*'. Si vous travaillez avec un FCB étendu, vous pouvez inscrire en outre une valeur dans le champ d'attribut du FCB_recherché pour circonscrire la recherche aux fichiers dotés d'attributs déterminés.

La recherche

Les préparatifs sont maintenant terminés et la recherche du premier fichier figurant dans le repertoire actuel peut commencer. On appellera à cet effet la fonction 11h avec le numéro de fonction dans le registre AH, l'adresse de segment du FCB_recherché dans le registre DS et son adresse d'offset dans le registre DX. Si un fichier portant le nom spécifié a pu être identifié, le registre AL contiendra la valeur 0 après appel de la fonction, sinon 255. Le nom du fichier trouvé et de son attribut (si vous avez travaillé avec des FCB étendus) figureront alors dans le FCB_trouvé. Pour la suite de la recherche de fichiers, ce ne sera plus alors la fonction 11h mais la fonction 12h qu'il faudra appeler. La définition des registres pour cette deuxième fonction est identique à celle de la fonction 11h, aussi bien pour l'appel que pour le retour. Si elle renvoie la valeur 255 dans le registre AL après l'un des appels, cela signifie que la recherche est terminée car il n'y a plus d'autres fichiers portant le nom recherché dans le repertoire actuel.

Un exemple

Pour illustrer la recherche de fichiers avec les fonctions FCB 11h et 12h, nous vous proposons le programme FF.ASM écrit en Assembleur. Il est conçu comme un programme COM. FF signifie FileFind puisque la tâche du programme consiste à rechercher un fichier précis ou un groupe de fichiers dans tous les répertoires d'un lecteur. Les fonctions FCB peuvent également être utilisées pour rechercher des fichiers dans divers répertoires. Il suffit alors de spécifier le répertoire de recherche comme le répertoire en cours au début de la recherche.

L'appel de FF s'effectue avec la syntaxe suivante :

```
FF [Lecteur:] Nom de fichier [+ ou - ou = Date]
```

Le paramètre Nom de fichier indique le nom du fichier à rechercher. Entrez par exemple ici FF.ASM pour que la recherche porte sur le fichier FF.ASM. Si votre entrée est en revanche *.ASM, la recherche concerne alors tous les programmes Assembleur du lecteur en cours. Avec *.*, vous pouvez afficher les noms de tous les fichiers à l'écran. Dans ce cas, vous obtenez non seulement le nom du fichier trouvé mais aussi le répertoire dans lequel il a élu domicile.

Si la recherche ne doit pas s'effectuer sur le lecteur en cours, faites précéder le nom de fichier de la désignation du lecteur comme vous avez l'habitude de le faire avec les diverses commandes DOS.

Un autre critère de recherche consiste à spécifier une date derrière le nom de fichier précédée d'un signe plus, moins ou égal. Comme le montre le tableau suivant, l'affichage des fichiers trouvés tient compte de la date de leur dernière modification. Ainsi, vous pouvez par exemple trouver toutes les lettres écrites dès le début du mois en cours.

Paramètre	Fonction
-Date	Afficher uniquement les fichiers créés avant la date spécifiée ou dernièrement modifiés
+Date	Afficher uniquement les fichiers créés après la date spécifiée ou dernièrement modifiés
=Date	Afficher uniquement les fichiers créés à la date spécifiée ou dernièrement modifiés

La commande

```
ff*.txt+1.10.1991
```

cherche tous les fichiers texte créés après le 1er octobre 1991.

Listing : FF.ASM

```
;*****************************************************************
;*                    F F . A S M    ( FileFind )                *
;*---------------------------------------------------------------*
;*   Fonction        : Recherche des fichiers sur                *
;*                     un lecteur donné                          *
;*---------------------------------------------------------------*
;*   Auteur          : MICHAEL TISCHER                           *
;*   Développé le    : 12.09.1990                                *
;*   Dernière modif. : 14.03.1992                                *
;*---------------------------------------------------------------*
;*   Pour assembler  : MASM FF;                                  *
;*                     LINK FF;                                  *
;*                     EXE2BIN FF FF.COM              ou         *
;*                                                               *
;*                     TASM FF                                   *
;*                     TLINK FF /t                               *
;*---------------------------------------------------------------*
;*   Appel           : FF [Unit:]nom de fichier [+ ou - ou =Date] *
;*****************************************************************

;-- Constantes ---------------------------------------------------

CMD_OFS     equ   81h    ;Début de la ligne d'instruction dans le PSP
CR          equ   13                              ;Carriage Return
LF          equ   10                                    ;Linefeed
DELIMITER   equ   "."          ;Séparateur pour les éléments de date

            ;--Fonctions DOS ---------------------------------------

GET_LW_NR   equ   19h                  ;Déterminer le lecteur actuel
WRT_CHAR    equ   02h                   ;Sortir caractère sur STDOUT
SEARCH_FIRST equ  4Eh                     ;Rechercher le 1er fichier
SEARCH_NEXT equ   4Fh                  ;Rechercher le fichier suivant
SET_DTA     equ   1Ah                        ;Fixer nouvelle zone DTA
PRINT_STR   equ   09h                  ;Sortir chaîne terminée par $

ATTR_DIR    equ   10h                  ;Rechercher un sous-répertoire
ATTR_NRM    equ   00h                  ;Rechercher des fichiers normaux

DATCOMP_NO  equ   0                       ;Pas de comparaison de date
DATCOMP_KL  equ   1                 ;Comparaison de date avant le ....
DATCOMP_GL  equ   2                    ;Comparaison de date le ....
DATCOMP_GR  equ   3                 ;Comparaison de date après le ...

;-- Structures ---------------------------------------------------

PSP   struc                              ;Constitue la structure du PSP
      intcall  dw (?)                 ;Appel d'interruption 20h
      endadr   dw (?)                           ;Adresse de fin
               db (?)                                  ;Réservé
      farcall  db  5 dup (?)       ;FAR-CALL sur l'interruption 21h
      int22h   dd (?)                          ;Copie de l'inter. 22h
      int23h   dd (?)                          ;Copie de l'inter. 23h
      int24h   dd (?)                          ;Copie de l'inter. 24h
               db 22 dup (?)                           ;Réservé
      envseg   dw (?)        ;Adresse de segment de l'environnement
               db 46 dup (?)                           ;Réservé

      ;-- Voici le premier FCB ----------------------------------

      drvcode1 db (?)                        ;Numéro de périphérique
      namfich1 db  8 dup (?)                      ;Nom de fichier
      extfich1 db  3 dup (?)                   ;Extension de fichier
      numbloc1 dw (?)                        ;Numéro de bloc actuel
      lenfich1 dd (?)                          ;Longueur de fichier
      datmod1  dw (?)              ;Date de la dernière modification
      timemod1 dw (?)              ;Heure de la dernière modification
               dw  4 dup (?)                           ;Réservé
      actset1  db (?)                  ;Numéro de descripteur actuel
      setfree1 dd (?)          ;Numéro de descripteur pour accès libre

      ;-- Voici le second FCB -----------------------------------

      drvcode2 db (?)                        ;Numéro de périphérique
      namfich2 db  8 dup (?)                      ;Nom de fichier
      extfich2 db  3 dup (?)                   ;Extension de fichier
      numbloc2 dw (?)                        ;Numéro de bloc actuel
      lenfich2 dd (?)                          ;Longueur de fichier
      datmod2  dw (?)              ;Date de la dernière modification
      timemod2 dw (?)              ;Heure de la dernière modification
               dw  4 dup (?)                           ;Réservé
      actset2  db (?)                  ;Numéro de descripteur actuel
      setfree2 dd (?)          ;Numéro de descripteur pour accès libre

      db 128 dup (?)                          ;Ligne d'instruction
PSP   ends

;-----------------------------------------------------------------

DTA   struc          ;Structure de DTA lors de la recherche de fichiers
      reserve  db 21 dup (?)                             ;Réservé
      attr     db (?)                  ;Attribut du fichier trouvé
      timemod  dw (?)              ;Heure de la dernière modification
      datemod  dw (?)              ;Date de la dernière modification
      lenfich  dd (?)                          ;Longueur de fichier
      datn     db 13 dup (?)                        ;Nom de fichier
DTA   ends

;-----------------------------------------------------------------

DATE  struc                          ;Reçoit des informations de date
      jour     db (?)                          ;Jour du mois (1-31)
      mois     db (?)                            ;Mois (1-12)
      an       dw (?)                  ;année y compris le siècle
DATE  ends

;-- Ici commence le programme proprement dit ---------------------

code      segment para 'CODE'          ;Définition du segment de code

          org 100h                                ;Start derrière PSP

          assume cs:code, ds:code, es:code, ss:code

start:    jmp startff                  ;Sauter vers le début du programme

;-- Données ------------------------------------------------------

datcomp       db  DATCOMP_NO     ;Par défaut, pas de recherche de date
sdate     DATE < 0, 0, 0>              ;Date de la ligne d'instruction
tdate     DATE < 0, 0, 0>                  ;Date du fichier à examiner

nomrep        db  64, ":\"                          ;Nom du répertoire
ffirstdir     db  79 dup (0)                                ;en cours
nomrech       db  "*.*", 0               ;Nom des fichiers à rechercher
              db   9 dup (0)

jokname       db  "*.*", 0       ;Nom pour la recherche de répertoires
found         dw  0                        ;Nombre de fichiers trouvés

mes           db  CR, LF, "FF: Pas de fichier(s) trouvé(s)."
crlf          db  CR, LF, "$"

datbuffer     db "ddddd.dd.dddd  ",0      ;Buffer pour conversion date
datpend   equ this byte                                   ;en ASCII

errmes        db "FF  -  (c) 1990, 92 by MICHAEL TISCHER", 13,10,10
              db "Appel: FF [L:]nom_fichier [+ ou - ou =Date]",13,10
              db "$"

;-- Début de programme effectif ----------------------------------

startff:  ;-- Evaluer les arguments de la ligne d'instruction ---------
          cld                          ;En avant avec les instructions chaîne
          mov   di,offset nomrech
          mov   si,CMD_OFS   ;SI sur le début de la ligne d'instruction

arg1:     ;-- Evaluer ensuite la désignation de périphérique ----------
          lodsb       ;Charger les caractères de la ligne d'instruction
          cmp   al,CR         ;Fin de la ligne d'instruction découverte?
          je    arg6                        ;Oui ---> Terminer l'évaluation
          cmp   al," "              ;Non, contient peut-être des espaces?
          jbe   arg1               ;Non, il doit s'agir de Tab, continuer

          ;-- Une lettre a été trouvée ------------------------------

          cmp   ds:[0].drvcode1,0         ;Désignation de périphérique?
          je    arg2   ;Non, sinon elle se trouverait déjà dans le FCB
          lodsb                                ;Charger le double-point
          lodsb                           ;Charger le caractère suivant
          cmp   al," "                             ;Espace ou inférieur?
          jbe   arg3                       ;Oui, ---> Evaluation terminée

arg2:     ;-- Entrer les noms de fichier dans le buffer NOMRECH -------

          stosb                      ;Placer les caractères dans NOMRECH
          lodsb                           ;Charger le caractère suivant
          cmp   al," "                               ;Espace ou inférieur?
          je    arg2                             ;Non, lettre, continuer

arg3:     ;-- Evaluation du nom de fichier terminée --------------------
```

```
        cmp   al,CR                     ;Fin de ligne atteinte?
        je    arg6

        ;-- Tester et évaluer éventuellement l'indication de date ---

arg4:   lodsb                           ;Charger le caractère suivant
        cmp   al," "                    ;Space et Tab autorisés
        je    arg4
        cmp   al,9
        je    arg4
        cmp   al,CR                     ;Fin de ligne atteinte?
        je    arg6

        cmp   al,"="                            ;Même date?
        jne   arg4a                     ;Non, continuer le test
        mov   al,DATCOMP_GL             ;Flag pour la comparaison de date
        jmp   short arg5

arg4a:  cmp   al,"-"                            ;Date avant le ...
        jne   arg4b                     ;Non, continuer le test
        mov   al,DATCOMP_KL             ;Flag pour la comparaison de date
        jmp   short arg5

arg4b:  cmp   al,"+"                            ;Date après le ...
        jne   argerr            ;Non, erreur dans la ligne d'instruction
        mov   al,DATCOMP_GR             ;Flag pour la comparaison de date

arg5:   mov   datcomp,al                ;Apercevoir le flag de date
        mov   di,offset sdate           ;Pointeur sur structure de date
        call  getdat            ;Lire la date dans la ligne d'instruction
        jc    argerr                    ;Erreur ---> Terminer le programme

arg6:   ;-- Evaluation ligne d'instruction terminée, tout va bien ---

        mov   al,ds:[0].drvcode1        ;Charger désignation périph.
        or    al,al                             ;Une trouvée?
        jne   arg7                      ;Oui, ---> actuelle non déterminée

        ;-- Pas de désignation de périph. spécifiée, sélect. actuelle

        mov   ah,GET_LW_NR      ;Déterminer numéro de lecteur actuel
        int   21h                                       ;(0=A)
        inc   al                ;Augmenter n° lecteur parce que 1 = A

arg7:   add   nomrep,al                         ;Dès, pér. en ASCII
        mov   dx,offset startdta                ;Adresse de premier DTA
        mov   bx,offset firstdir                ;Ici nom du répertoire
        call  go                                ;Lancer la recherche

        mov   ax,found          ;Charger le nombre de fichiers trouvés
        or    ax,ax                     ;Aucun fichier trouvé?
        je    arg8                ;Oui, sortir la chaîne inchangée
        mov   si,offset mes+6                   ;Pointeur sur buffer
        call  toint

arg8:   mov   ah,09h                            ;Afficher le message
        mov   dx,offset mes
        int   21h

        mov   ax,4C00h                  ;Terminer le programme par
        int   21h                               ;la fonction 4Ch

argerr: ;-- Erreur lors de l'évaluation de la ligne d'instruction ---

        mov   ah,09h                    ;Sortir la chaîne erronée
        mov   dx,offset errmes
        int   21h

        mov   ax,4C01h  ;Terminer le programme avec le code d'erreur
        int   21h

;-- GO: Gère la recherche des fichiers spécifiés ----------------------
;-- Entrée  : BX = Ptr sur chaîne recherchée pour insérer nom répertoire
;--               ou de fichier
;--           DX = Offset de DTA antérieur
;-- Sortie  : Aucune
;-- Registres : AX, SI, CX et FLAGS modifiés

go      proc near

        push  dx                ;Sauvegarder le pointeur sur DTA en cours

        mov   si,offset nomrech  ;Copier le nom du fichier recherché
        call  strcopy                           ;dans le buffer

        mov   cx,ATTR_NRM       ;Rechercher uniq. les fichiers normaux
        call  startsearch                       ;Commencer la recherche
        jc    go2               ;Aucun fichier trouvé -> Rech. répertoire

        call  printname                         ;Fichier trouvé

go1:    ;-- Continuer la recherche des fichiers restants ------------

        call  nextfile                  ;Rechercher le fichier suivant
        jc    go2                               ;Plus de fichier trouvé
        call  printname                         ;Fichier trouvé
        jmp   short go1

go2:    ;-- Consulter encore une fois le répertoire en cours, -------
        ;-- mais en recherchant également les sous-répertoires ------

        pop   dx                        ;Replacer le pointeur sur DTA
        push  dx                        ;et sauvegarder immédiatement
        mov   si,offset jokname         ;*.* Ajouter aux noms de rép.
        call  strcopy

        mov   cx,ATTR_DIR               ;Rechercher sous-répertoires
        call  startsearch       ;Rechercher le premier sous-répertoire
        jc    go_end            ;Aucun trouvé ---> Terminer la routine

        mov   si,dx                     ;Trouvé mais c'est un répertoire
        test  [si].attr,ATTR_DIR
        jne   go4                               ;Oui ---> Evaluer

go3:    ;-- Pas de répertoire, continuer la recherche ---------------

        call  nextfile                  ;Rechercher le fichier suivant
        jc    go_end            ;Aucun trouvé ---> Terminer la routine
        mov   si,dx                     ;Trouvé mais c'est un répertoire
        test  [si].attr,ATTR_DIR
        je    go3                                       ; Jump if zero

go4:    ;-- Sous-répertoire trouvé -----------------------------------

        cmp   [si].datn,"."                             ;"." ou ".."
        je    go3                               ;Oui, ne pas traiter

        push  di                        ;Sauver DX et BX sur la pile
        push  bx
        mov   si,dx         ;Placer SI sur le nom de fichier dans DTA
        add   si,offset datn        ;y placer sur le nom de répertoire
        mov   di,bx                 ;DI sur nom de répertoire antérieur

go5:    ;-- Inclure le nom de répertoire au nom antérieur ------------

        lodsb       ;Charger le caractère depuis le nom de répertoire
        stosb                           ;et inclure au nom antérieur
        or    al,al                     ;Dernier octet traité?
        jne   go5                               ;Non ---> Continuer

        mov   bx,di             ;Ranger nouvelle fin de buffer dans BX
        stosb                           ;Ecrire encore un octet NUL
        mov   byte ptr [bx-1],"\"   ;Le faire précéder d'un Backslash
        call  go                                ;Appel récursif

        pop   bx                        ;Relire BX et DI depuis la pile
        pop   di
        mov   byte ptr [bx],0           ;Dissocier dern. nom de rép.
                                        ;de l'appel précédent

        mov   ah,SET_DTA        ;Replacer DTA sur l'ancienne adresse
        int   21h                                       ;en DX

        jmp   short go3

go_end: pop   dx                ;Replacer le pointeur sur l'ancien DTA
        ret                                     ;Retour à l'appelant

go      endp

;-- STARTSEARCH: Lancer la recherche d'un fichier ---------------------
;-- Entrée  : CX = Attribut du fichier recherché
;--           DX = Offset de DTA antérieur
;-- Sortie  : DX = Nouvelle adresse de DTA
;--           Carry-Flag: 0 = o.k., 1 = Pas de fichier trouvé
;-- Registres : AX, DX, BP et FLAGS sont modifiés
;-- Infos     : Nom du fichier à rech. à extrait du buffer nomrep
;--

startsearch     proc near

        push  cx                                ;Sauver CX sur la pile

        ;-- Placer d'abord nouveau DTA derrière le DTA en cours -----

        add   dx,2ch            ;La longueur d'un DTA est de 42 octets
        mov   ah,SET_DTA                        ;Fixer nouveau DTA
        int   21h

        ;-- Rech. 1er fichier doté d'un nom dans le buffer nomrep ---

        mov   bp,dx                             ;Sauvegarder DX
        mov   ah,SEARCH_FIRST           ;Rechercher premier fichier
        mov   dx,offset nomrep      ;DS:DX = Ptr sur nom de fichier
        int   21h

        mov   dx,bp                             ;Retirer DX
        pop   cx                        ;Retirer CX de la pile
```

```
;           ret                                         ;Retour à l'appelant
;
startsearch     endp
;
;;-- NEXTFILE: Recherche le fichier suivant ----------------------------
;;-- Entrée  : Aucune
;;-- Sortie  : Carry-Flag: 0 = o.k., 1 = Pas de fichier trouvé
;;-- Registres : AY et FLAGS sont modifiés
;
nextfile        proc near
;
            mov   bp,dx
            mov   ah,SEARCH_NEXT            ;Rechercher le fichier suivant
            mov   dx,offset nomrep          ;Chemin dans le buffer nomrep
            int   21h
            mov   dx,bp
            ret                                         ;Retour à l'appelant
;
nextfile        endp
;
;;-- STRCOPY : Copie une chaîne ASCII terminée par ---------------------
;;--           un octet NUL
;;-- Entrée  : SI = Offset de la chaîne source
;;--           BX = Offset de la chaîne cible
;;--           DS = Segment de la chaîne source
;;--           ES = Segment de la chaîne cible
;;-- Sortie  : Aucune
;;-- Registres : AL, DI, SI et FLAGS sont modifiés
;
strcopy         proc near
;
            mov   di,bx                 ;Placer DI sur la chaîne cible
;
scl:        ;-- Boucle de recopie ---------------------------------------
;
            lodsb               ;Charger caractère depuis la chaîne source
            stosb                       ;Placer dans la chaîne cible
            or    al,al                         ;Dernier caractère?
            jne   scl                   ;Non ---> Recommencer le tout
;
            ret                                         ;Retour à l'appelant
;
strcopy         endp
;
;;-- PRINTNAME: Sortir le nom d'un fichier trouvé ----------------------
;;-- Entrée  : DX = Pointeur sur DTA actuel muni d'un nom
;;--           BX = Offset de la chaîne cible
;;--           DS = Segment de la chaîne source
;;--           ES = Segment de la chaîne cible
;;-- Sortie  : Aucune
;;-- Registres : AL, DI, SI et FLAGS sont modifiés
;;-- Infos     : Le nom du fichier n'est imprimé que s'il ne s'agit pas
;;--             d'un sous-répertoire et si un test de date déterminé
;;--             a permis d'établir la relation souhaitée.
;
printname       proc near
;
            mov   bp,dx             ;Sauvegarder pointeur sur DTA en BP
            cmp   [bp].datn,"."                         ;"." ou ".."
            je    prnend                        ;Oui, pas de sortie
;
            ;-- Lancer la comparaison de date si souhaitée --------------
;
            cmp   datcomp,DATCOMP_NO                    ;Test de date?
            je    pn1                       ;Non, Sortir directement le nom
;
            mov   ax,[bp].datemod           ;Oui, lire date et décompacter
            mov   si,offset tdate
            call  entpackdat
;
            mov   di,offset sdate                       ;Comparer la date
            call  cmpdat
            cmp   al,datcomp                    ;Relation souhaitée?
            jne   prnend                        ;Non, ne pas sortir
;
            call  printdat                      ;Date o.k., sortir
;
            ;-- Sortir ensuite les noms de répertoires ------------------
;
pn1:        mov   dx,offset nomrep          ;Pointeur sur nom de répertoire
            xor   al,al             ;Lire le dernier caractère du répertoire
            xchg  al,[bx]                           ;et régler sur 0
            mov   di,ax                     ;Ranger le car. dans DI
            call  printasciiz               ;Sortir nom de répertoire
;
            xchg  ax,di             ;Recharger AX avec l'ancienne valeur
            mov   [bx],al           ;Remettre l'ancien caractère à sa place
;
            ;-- Insérer les noms de fichiers ----------------------------
;
            mov   dx,bp                     ;Replacer le pointeur sur DTA
            add   dx,offset datn        ;Adresser le nom de fichier en DTA
            call  printasciiz                           ;et sortir
;
            mov   ah,PRINT_STR              ;Sortir saut de ligne à l'aide
            mov   dx,offset crlf                        ;du DOS
            int   21h
;
            fnc   found                         ;Un autre fichier trouvé
            mov   dx,bp                             ;Relire l'ancien DX
;
prnend:     ret                                         ;Retour à l'appelant
;
printname       endp
;
;;-- PRINTASCIIZ : Affiche une chaîne ASCII terminée par un octet NUL ---
;;--               sur STDOUT
;;-- Entrée  : DX = Pointeur sur le début de chaîne
;;--           DS = Segment de la chaîne
;;-- Sortie  : Aucune
;;-- Registres : AX, DX, SI et FLAGS sont modifiés
;
printasciiz     proc near
;
            mov   si,dx                     ;Charger chaîne depuis DX
            mov   ah,WRT_CHAR               ;N° fonct. pour sortie car.
            lodsb                           ;Charger premier caractère
;
pa1:        ;-- Boucle de sortie ----------------------------------------
;
            mov   dl,al             ;Caractère pour fonction DOS vers DL
            int   21h                                   ;et sortir
;
            lodsb                               ;Charger le car. suivant
            or    al,al                 ;Fin de la chaîne atteinte?
            jne   pa1                           ;Non, sortir car.
;
            ret                                         ;Retour à l'appelant
;
printasciiz     endp
;
;;-- TOINT: Convertit un nombre binaire en ASCII et le place -----------
;;--        dans un buffer de l'appelant
;;-- Entrée  : DS:SI = Adresse du début de buffer
;;--           AX    = Le nombre binaire à convertir
;;-- Sortie  : DS:SI = Adresse du premier chiffre
;;-- Registres : AX, SI et FLAGS sont modifiés
;;-- Infos     : - Le buffer doit contenir de la place pour cinq car.
;;--             - Le nombre entré dans le buffer est justifié à droite.
;
toint           proc near
;
            push  dx                        ;Sauver les registres modifiés
            push  bx                                    ;sur la pile
;
            ;-- Compléter le buffer par des espaces ---------------------
;
            mov   word ptr [si],   32 shl 8 + 32
            mov   word ptr [si+2], 32 shl 8 + 32
            mov   byte ptr [si+4], 32
;
            ;-- Dans une boucle, diviser toujours un nombre par 10 -------
            ;-- et convertir la position de poids faible au format ASCII -
            ;-- puis reporter dans le buffer ----------------------------
;
            add   si,5                      ;Placer SI sur la fin du buffer
            mov   bx,10                       ;Le diviseur est toujours 10
;
tl1:        dec   si                            ;SI sur le car. précédent
            xor   dx,dx                         ;Le dividende est DX:AX
            div   bx                            ;Diviser DX:AX par 10
            or    dl,'0'                ;Transformer DL au format ASCII
            mov   [si],dl                       ;Placer dans le buffer
            or    ax,ax                         ;Y a-t-il un reste?
            jne   tl1                           ;OUI --> chiffre suivant
;
            pop   bx                            ;Retirer les registres
            pop   dx
;
            ret                                         ;Retour à l'appelant
;
toint           endp
;
;;-- GETINT: lit un nombre décimal positif dans la ligne d'instruction et
;;--         le convertit au format binaire
;;-- Entrée  : SI = Ptr sur le caractère suivant à lire dans le buffer
;;--               de la ligne d'instruction
;;-- Sortie  : Carry = 0: Nombre correct, 1 = Erreur
;;--           SI    = Pointeur derrière le dernier caractère lu
;;--           AX    = Nombre (seulement si Carry = 0)
;;-- Registres : AX, CX, SI et FLAGS sont modifiés
;
getint          proc near
;
            push  bx                        ;Sauver les registres modifiés
            push  dx
```

```
        push    dl

        mov     dl,10                   ;Le facteur est toujours 10
        xor     bx,bx                   ;BX reçoit le nombre binaire
        mov     ah,bh                   ;Hi-Byte de AH toujours 0

        ;-- Commencer à travailler depuis le début du nombre ---------

gi1:    lodsb                           ;Charger le car. suivant
        cmp     al,'0'                  ;Test sur le chiffre
        jb      gi2
        cmp     al,'9'
        jbe     gi5                     ;Chiffre trouvé --> GI5
gi2:    cmp     al,' '          ;Pas de chiffre, donc SPACE seulement et
        je      gi1                     ;TAB autorisé
        cmp     al,9
        je      gi1

    ;-- Lire le nombre chiffre par chiffre et convertir en binaire ---

gi4:    lodsb                           ;Lire le caractère suivant
        cmp     al,'0'                  ;Est-ce un chiffre?
        jb      gi7             ;NON --> Traiter le caractère suivant
        cmp     al,'9'
        ja      gi7             ;NON --> Traiter le caractère suivant

        ;-- OUI, c'est un chiffre -------------------------------------

gi5:    xchg    ax,bx           ;Lire le caractère en BX, le nombre en AX
        mul     dl                      ;Multiplier AX par 10
        or      dx,dx                   ;Produit supérieur à 655367
        jne     gierr                   ;OUI --> Nombre trop grand
        and     bl,0Fh          ;Il ne reste que les 4 bits inférieurs
        add     ax,bx                   ;Y ajouter le chiffre
        xchg    ax,bx                   ;Permuter AX et BX
        jmp     gi4                     ;Lire le caractère suivant
gi6:    clc                     ;Lire correctement le nombre
        mov     ax,bx                   ;Placer le nombre en AX
        jmp     short giend             ;Retour à l'appelant

gi7:    dec     si              ;Un caractère lu trop loin mais
        jmp     gi6                     ;nombre o.k

gierr:  stc                             ;Afficher l'erreur

giend:  pop     dl                      ;Relire les registres
        pop     dx                      ;modifiés
        pop     bx
        ret                             ;Retour à l'appelant

getint  endp

;-- GETDAT: Lit une date au format "Jour.Mois.Année" dans la  ---------
;--         ligne d'instruction, la convertit en binaire et la place
;--         dans la structure spécifiée
;-- Entrée  : SI = Ptr sur le caractère suivant à lire dans le buffer
;--               de la ligne d'instruction
;--           DI = Pointeur sur la structure de données devant recevoir
;--               la date convertie
;-- Sortie  : Carry = 0: Nombre o.k., 1 = Erreur
;--           SI    = Pointeur derrière le dernier caractère lu
;-- Registres : AX, CX, SI et FLAGS sont modifiés

getdat  proc near

        call    getint                          ;Lire le jour
        jc      gderr                   ;Erreur? ---> Annuler la routine

        or      ax,ax                   ;Jour o.k., mais aussi valable?
        je      gderr                     ;Null non autorisé --> Erreur
        cmp     ax,31                           ;31 est le maximum
        ja      gderr                   ;Dépassement ---> Erreur

        mov     [di].jour,al    ;Jour o.k., range et continue évaluation
        lodsb                           ;Un point doit apparaître
        cmp     al,DELIMITER
        jne     gderr                   ;mais n'est pas apparu ---> Erreur

        ;-- Le jour était correct, évaluer maintenant le mois --------

        call    getint                          ;Lire le mois
        jc      gderr                   ;Erreur? ---> Annuler la routine

        or      ax,ax                   ;Mois o.k., mais aussi valable?
        je      gderr                     ;Null est autorisé --> Erreur
        cmp     ax,12                           ;12 est le maximum
        ja      gderr                   ;Dépassement ---> Erreur

        mov     [di].mois,al                    ;Mois o.k

        lodsb                           ;Un point doit apparaître également
        cmp     al,DELIMITER
        jne     gderr                   ;mais n'est pas apparu ---> Erreur

        ;-- Le mois était correct, évaluer maintenant l'année --------

        call    getint                          ;Lire l'année
        jc      gderr                   ;Erreur? ---> Annuler la routine

        cmp     ax,100                          ;Sans indication de siècle?
        ja      gd1                             ;Non, --> Evaluer
        add     ax,1900     ;Oui, prendre également le siècle en compte

gd1:    cmp     ax,1980                         ;Avant 1980 non admis
        jb      gderr                           ;Avant ---> Erreur

        mov     [di].an,ax              ;L'année est correct, tout va bien
        clc
        ret

gderr:  stc                             ;Impossible de traiter la date
        ret                             ;Retour à l'appelant

getdat  endp

;-- ENTPACKDAT: Décompacte une date stockée au format DOS et la --------
;--            la reporte dans une structure de type DATE
;-- Entrée  : AX = La date à décompacter au format DOS
;--           SI = Pointeur sur le bloc de date devant recevoir
;--                les informations
;-- Sortie  : Aucune
;-- Registres : AX, CX et FLAGS sont modifiés

entpackdat      proc near

        push    bx                              ;Sauver BX sur la pile
        mov     bl,al                   ;Lire Lo-Byte de la date en BX
        and     bl,31                   ;Dissocier les 3 bits supérieurs
        mov     [si].jour,bl                    ;Ranger le jour du mois

        mov     bx,ax                   ;Lire la totalité de la date en BX
        mov     cl,5                    ;Décaler de 5 bits vers la droite
        shr     bx,cl
        and     bl,15                   ;Dissocier les 4 bits supérieurs
        mov     [si].mois,bl                    ;Ranger le mois

        shr     ah,1            ;Hi-Byte d'une position vers la droite
        mov     al,ah           ;année par rapport à 1980 dans LSB
        xor     ah,ah                   ;Hi-Byte sur 0
        add     ax,1980                 ;Construire une année absolue
        mov     [si].an,ax              ;Placer dans la structure

        pop     bx                              ;Relire BX
        ret                             ;Retour à l'appelant

entpackdat endp

;-- CMPDAT: Compare deux indications de date ---------------------------
;-- Entrée  : SI = Pointeur sur le premier bloc de date
;--           DI = Pointeur sur le second bloc de date
;-- Sortie  : Aucune
;-- Registres : AX et FLAGS sont modifiés

cmpdat  proc near

        ;-- Comparer d'abord les années -------------------------------

        mov     ax,[si].an                      ;Charger année 1
        cmp     ax,[di].an              ;et comparer avec année 2
        jb      avant                   ;Date 1 avant date 2 --> AVANT
        ja      apres                   ;Date 1 après Date 2 --> APRES

        ;-- Les années sont identiques, comparer maintenant les mois -

        mov     al,[si].mois                    ;Charger mois 1
        cmp     al,[di].mois            ;et comparer avec mois 2
        jb      avant                   ;Date 1 avant Date 2 --> AVANT
        ja      apres                   ;Date 1 après Date 2 --> APRES

        ;-- Les mois étant identiques, comparer le jour du mois ------

        mov     al,[si].jour                    ;Charger jour 1
        cmp     al,[di].jour            ;et comparer avec jour 2
        jb      avant                   ;Date 1 avant Date 2 --> AVANT
        ja      apres                   ;Date 1 après Date 2 --> APRES

        mov     al,DATCOMP_GL           ;Les deux données sont identiques
        ret                             ;Retour à l'appelant

avant:  mov     al,DATCOMP_KL                   ;Date 1 avant Date 2
        ret                             ;Retour à l'appelant

apres:  mov     al,DATCOMP_GR                   ;Date 1 après Date 2
        ret                             ;Retour à l'appelant

cmpdat  endp
```

```
;-- PRINTDAT: Affiche une date issue d'une structure de date sur l'écran
;-- Entrée  : SI = Pointeur sur structure de date
;-- Sortie  : Aucune
;-- Registres : AX, DX, SI et FLAGS sont modifiés

printdat        proc near

        push    si                      ;Sauver SI, BX et DX sur la pile
        push    bx
        push    dx

        mov     bx,si           ;Placer BX sur la structure de date
        mov     si,offset datpend-8   ;SI à l'endroit réservé à l'année
        mov     ax,[bx].an                      ;Charger l'année
        call    toint                   ;et convertir en ASCII
        mov     byte ptr [si-1],DELIMITER     ;Un point devant l'année

        sub     si,6                    ;SI à l'endroit réservé au mois
        mov     al,[bx].mois                    ;Charger le mois en AX
        xor     ah,ah
        call    toint                   ;et convertir en ASCII
        cmp     [bx].mois,10            ;Faire précéder d'un zéro?
        jae     pd1                     ;Non, donc transmettre

        dec     si
        mov     byte ptr [si],"0"

pd1:    mov     byte ptr [si-1],DELIMITER     ;Un point devant le mois
        sub     si,6                    ;SI à l'endroit réservé au jour
        mov     al,[bx].jour                    ;Charger le jour en AX
        xor     ah,ah
        call    toint                           ;et convertir en ASCII

        cmp     [bx].jour,10        ;Espace devant le premier chiffre?
        jae     pd2                         ;Oui, transmettre tel quel

        dec     si                      ;Non, SI sur le premier chiffre
        mov     byte ptr [si]," "               ;et insérer un espace

pd2:    mov     dx,si                                   ;Début en DX
        call    printascfiz                     ;Sortir la chaîne ASCII

pd3:    pop     dx                      ;Retirer DX, BX et SI de la pile
        pop     bx
        pop     si

        ret

printdat        endp

;-- A partir d'ici les DTA seront conservés pendant les appels récursifs

startdta  equ this byte

;-- Fin ---------------------------------------------------------------

code    ends                                    ;Fin du segment de code
        end   start
```

20.2.2. Recherche de fichiers avec les fonctions Handle

En comparaison, le travail avec les fonctions handle est à la fois plus pratique et plus simple. Ici aussi, on dispose de deux fonctions, une pour rechercher le premier fichier (la fonction 4Eh) et l'autre pour poursuivre la recherche, la fonction 4Fh. Les deux fonctions placent dans la DTA les données des fichiers trouvés. C'est pourquoi la DTA doit être transférée dans une zone de mémoire accessible au programme avant d'appeler ces deux fonctions. Cette zone de mémoire, comme nous allons le voir, doit avoir une taille d'au moins 43 octets. Ici aussi, comme nous l'avons déjà expliqué pour les fonctions FCB, la DTA devra être réinstallée dans sa zone d'origine une fois la recherche terminée.

Pour appeler la fonction 4Eh, le numéro de fonction doit être transmis dans le registre AH, l'attribut recherché dans le registre CX et l'adresse dans la mémoire du nom de fichier recherché dans la paire de registres. Le nom de fichier lui-même doit être stocké sous forme d'une séquence de caractères ASCII terminée par une marque de fin (Code ASCII 0). Il peut comporter un nom de périphérique et une spécification de chemin complète ainsi bien sûr que les jokers '*' et '?' qui permettent de faire rechercher tout un groupe de fichiers. Si aucun chemin n'est indiqué, le DOS considère que la recherche doit se faire dans le répertoire actuel du périphérique spécifié ou bien (si aucun périphérique n'est spécifié) dans le périphérique actuel. Après appel de la fonction, le flag Carry indique si un fichier a été trouvé. Si ce n'est pas le cas, le flag Carry est mis et le registre AX contient un code d'erreur. Ce code sera 2 si le chemin indiqué n'existe pas et 12h si aucun fichier n'a pu être trouvé. Si le flag Carry est annulé, la DTA contient les informations sur le fichier trouvé. Ces informations présentent la structure suivante :

Structure d'une entrée du répertoire telle que les fonctions 4Eh et 4Fh du DOS la renvoient		
Adresse	**Contenu**	**Type**
+00h	Réservé	21 BYTES
+15h	Octet d'attribut du fichier	1 BYTE
+16h	Heure de la dernière modification	1 WORD
+18h	Date de la dernière modification	1 WORD
+1Ah	Taille du fichier	1 DWORD
+1Eh	Nom de fichier et extension séparés par un point, mais sans indication de chemin	12 BYTES
Longueur : 42 octets		

Toute recherche ultérieure se fera ensuite à travers la fonction 4Fh qui n'aura besoin d'aucun autre paramètre que le numéro de fonction dans le registre AH. Le flag Carry indique ici aussi s'il y a encore dans le répertoire d'autres fichiers auxquels corresponde le critère de recherche.

Exemples de programmes

Pour illustrer le mode d'accès aux fichiers, nous vous proposons des programmes écrits en BASIC, Pascal et en C. Tout comme l'instruction DIR, ils reçoivent, en guise de paramètres, les noms des fichiers à afficher derrière le nom du programme. Contrairement à DIR, l'affichage des noms de fichier, leur taille et leurs attributs s'effectue dans une fenêtre de l'écran et la sortie des noms de fichier s'interrompt dès que cette fenêtre est entièrement remplie de nouveaux noms de fichier. Vous pouvez ainsi examiner en toute quiétude les fichiers affichés. Vous pouvez ensuite faire reprendre la sortie des autres noms de fichier en frappant une touche quelconque.

Comme pour la plupart des programmes de ce livre, nous avons tâché de fournir des codes que vous pouvez facilement intégrer dans vos programmes. Ainsi, vous n'aurez plus de problèmes à inviter l'utilisateur à sélectionner les fichiers directement sur l'écran.

Bien que les programmes soient proposés en trois langages, vous trouverez cinq listings dans les pages suivantes. En fait, nous avons écrit les programmes Pascal et C en deux versions différentes. Notre but n'est pas de présenter la différence entre les compilateurs Microsoft et Borland, mais de décrire l'appel direct et indirect des fonctions DOS 4Eh et 4Fh. En effet, les bibliothèques Turbo Pascal ainsi que les divers compilateurs C de la société Microsoft ou Borland disposent de fonctions prédéfinies appelées FINDFIRST et FINDNEXT. Elles prennent en charge l'appel direct des fonctions DOS 4Eh et 4Fh et offrent en outre d'autres techniques intéressantes que nous aurons l'occasion de rencontrer.

Les remarques précédentes ne concernent pas le BASIC puisque QuickBasic ne contient qu'une seule instruction rudimentaire qui est FILES et qui permet d'afficher seulement le contenu du répertoire en cours sans que ces informations ne soient accessibles par le programme. L'appel direct des fonctions DOS 4Eh et 4Fh ne s'applique donc pas au BASIC.

Structure des programmes

Les cinq programmes sont bâtis autour d'une structure de base identique : la ligne de commande est d'abord évaluée dans le programme principal pour communiquer le nom du fichier à afficher. Si le nom du programme n'est suivi d'aucun paramètre, "*.*" est automatiquement sélectionné. Si le nom du programme est exactement suivi d'un paramètre, ce dernier est considéré comme le nom du fichier à afficher. En présence de plusieurs paramètres, le programme s'interrompt en renvoyant un message d'erreur.

Le nom du fichier à afficher est ensuite transmis à la fonction (ou procédure), sans oublier l'attribut qui doit également être indiqué. En principe, tous les bits d'attribut sont réglés dans ce cas, y compris les noms de volume, les sous-répertoires, les fichiers système et les fichiers cachés. L'utilisateur du programme ne peut pas agir sur cette définition qui peut toutefois être modifiée à l'intérieur du code de programme.

Les trois programmes ayant un rapport avec les fonctions 4Eh et 4Fh ne peuvent pas se servir des fonctions de bibliothèque prédéfinies et le DTA est donc réglé sur une structure de données locale à l'aide de la fonction/procédure SETDTA. SETDTA exécute un appel de la fonction DOS 1Ah pour décaler le DTA vers l'adresse spécifiée.

La structure de données prévue pour la réception du DTA est du type DIRSTRUCT dans les trois programmes. Elle représente un registre contenant la description des divers champs dans lesquels les fonctions 4Eh et 4Fh placent les informations obtenues sur les fichiers trouvés.

Après l'initialisation du DTA, la procédure/fonction ConfigEcran est appelée à l'intérieur de DIR. Elle ouvre la fenêtre sur l'écran devant contenir ultérieurement les informations concernant les fichiers.

L'appel suivant concerne FINDFIRST qui doit lancer la recherche. Si l'appel aboutit, l'exécution doit commencer par une boucle qui appelle en permanence FINDNEXT jusqu'à ce que tous les fichiers soient traités. Les fichiers trouvés sont alors affichés sur l'écran au moyen de la procédure/fonction PRINTDATA. En guise d'argument, elle attend DIRSTRUCT où DOS a placé les informations à propos du fichier. Ces informations sont décodées et affichées à l'écran sous une forme lisible.

Voici la version BASIC de notre petit créateur de liste de répertoires.

Listing : DIRB.BAS

```
'*****************************************************************
'*                          D I R B                              *
'*---------------------------------------------------------------*
'* Fonction  :  Affiche tous les fichiers d'un répertoire quelconque *
'*              y compris les sous-répertoires et noms de volumes *
'*              sur l'écran                                      *
'*---------------------------------------------------------------*
'* Auteur          : Michael Tischer                             *
'* Développé le    :  8.07.1987                                  *
'* Dernière modif.: 15.03.1992                                   *
'*****************************************************************

DECLARE FUNCTION MakeWord! (Nombre AS INTEGER)
DECLARE FUNCTION Dat$ (Valeur AS INTEGER)
DECLARE SUB SetDTA (Segment AS LONG, Offset AS LONG)
DECLARE SUB ConfigEcran ()
DECLARE FUNCTION Mois$ (Mon AS INTEGER)
DECLARE FUNCTION FindNext% ()
DECLARE FUNCTION FindFirst% (NomFichier AS STRING, Attr AS INTEGER)
DECLARE SUB PrintData (DirBuf AS ANY)
DECLARE SUB Dir (Pfad AS STRING, Attr AS INTEGER)

'$INCLUDE: 'QB.BI'          'FichierInclude pour appels d'interruptions

'-- Stucture d'entrée de répertoire comme dans fonctions DOS &H4E et ---
'-- retourner &H4F-----------------------------------------------------

TYPE DirStruct
   Reserve AS STRING * 21
   Attrib AS STRING * 1
   Time AS INTEGER
   Date AS INTEGER
   Size AS LONG
   DatName AS STRING * 13
END TYPE

'-- Constantes -------------------------------------------------------

CONST TRUE = -1                         'Déclarer les valeurs de vérité
CONST FALSE = NOT TRUE

CONST FCARRY = 1                                          'Carry-Flag
CONST EIntr = 14          'Autant d'entrées sont visibles simultanément
CONST BildTop = (20 - EIntr) \ 2        'Ligne sup. de fenêtre de sortie

CONST FAReadOnly = &H1                          ' Attributs de fichier
CONST FAHidden = &H2
CONST FASysFile = &H4
CONST FAVolumeID = &H8
CONST FADirectory = &H10
CONST FAArchive = &H20
CONST FAAnyFile = &H3F

'-- Programme principal -----

IF COMMAND$ = "" THEN                       'Aucun nom de fichier précisé?
  CALL Dir("*.*", FAAnyFile)       'Non, aff. ts fic. dans rép. en cours
ELSE
  CALL Dir(COMMAND$, FAAnyFile)    'Oui, afficher les fichiers désignés
END IF

'*****************************************************************
'* ConfigEcran : Configure l'écran pour la sortie                *
'*               d'un répertoire                                 *
'* Entrée      : Aucune                                          *
'* Sortie      : Aucune                                          *
'*****************************************************************

SUB ConfigEcran

CONST Lr = "                "           'Décale marge gauche vers position
DIM Compteur AS INTEGER                                'Compteur de boucle

CLS                                                    'Effacer l'écran
VIEW PRINT (BildTop + 1) TO (BildTop + 5 + EIntr)

PRINT Lr + ""
PRINT Lr + "  Nom          Taille     Date        Heure   RHSVD"
PRINT Lr + ""

FOR Compteur = 1 TO EIntr
PRINT Lr + "                                                  "
NEXT

PRINT Lr + ""
VIEW PRINT (BildTop + 4) TO (BildTop + 3 + EIntr)

END SUB

'*****************************************************************
'* Dat$    : Convertit valeur dans chaîne de car. dont le format *
'*           correspond à celui utilisé pour sorties de date/heure *
'* Entrée  : Valeur à convertir                                  *
'* Sortie  : Chaîne de car. correspondant à la valeur            *
'* Infos   : STR$ ou RPINT USING ne retournent pas de zéros de fin *
'*****************************************************************

FUNCTION Dat$ (Valeur AS INTEGER)

DIM Memoire AS STRING                  'Sauvegarde la valeur intermédiaire

Memoire = LTRIM$(STR$(Valeur))
WHILE LEN(Memoire) < 2             'La mémoire contient moins de 2 chiffres
  Memoire = "0" + Memoire
WEND

Dat$ = Memoire

END FUNCTION

'*****************************************************************
'* Dir     : Contrôle la lecture et la sortie du répertoire      *
'* Entrée  : Aucune                                              *
'* Sortie  : Aucune                                              *
'*****************************************************************

SUB Dir (Chemin AS STRING, Attr AS INTEGER)

DIM NbEntrees AS INTEGER                  'Nombre total d'entrées trouvées
DIM NbImage   AS INTEGER                   'Nombre d'entrées dans l'image
DIM DirBuf    AS DirStruct              'Reçoit des informations de fichier

CALL SetDTA(VARSEG(DirBuf), VARPTR(DirBuf))          'DirBuf = nouveau DTA
CLS                                                    'Effacer l'écran
CALL ConfigEcran          'Construire l'écran pour la sortie de répertoire

NbImage = -1          'Aucune entrée n'est encore affichée dans la fenêtre
NbEntrees = 0                  'Aucune entrée na été encore trouvée
IF FindFirst(Chemin, Attr) THEN         'Rech. 1ère entrée (Attribut egal)
  DO                                                 'Afficher toutes les entrées
    NbEntrees = NbEntrees + 1                          'Une autre entrée trouvée
    NbImage = NbImage + 1                             'Une autre entrée affichée
    IF NbImage = EIntr THEN                                  'Fenêtre pleine ?
    '-- Oui, attendre appui sur touche puis aff. nouvelle page de tableau
      VIEW PRINT (BildTop + 5 + EIntr) TO (BildTop + 6 + EIntr)
      PRINT "                  Veuillez appuyer sur une touche "
      SLEEP                                  'Attendre l'appui sur une touche
      VIEW PRINT (BildTop + 4) TO (BildTop + 3 + EIntr)
      NbImage = 0                  'Afficher d'autres entrées dans la fenêtre
    END IF
    CALL PrintData(DirBuf)                 'Afficher les données de l'entrée
  LOOP UNTIL NOT FindNext                          'Y a-t-il une autre entrée
END IF

VIEW PRINT (BildTop + 5 + EIntr) TO (BildTop + 6 + EIntr)
CLS

SELECT CASE NbEntrees
  CASE 0
    PRINT "Aucun fichier trouvé"
  CASE 1
    PRINT "Un fichier trouvé"
  CASE ELSE
    PRINT STR$(NbEntrees); " fichiers trouvés"
END SELECT
VIEW PRINT 1 TO 25

END SUB

'*****************************************************************
'* FindFirst : Lire la première entrée de répertoire             *
'* Entrée    : Le nom de fichier, Les attributs de fichier       *
'* Sortie    : TRUE, si l'entrée a été trouvée sinon FALSE       *
'* Infos     : L'entrée est lue dans les variables DirBuf        *
'*****************************************************************

FUNCTION FindFirst% (NomFichier AS STRING, Attr AS INTEGER)

DIM Fichier AS STRING * 65          'Mém. tampon pour nom fichier (s. Text)
DIM Regs AS RegTypeX                              'Registres du processeur

Fichier = NomFichier                        'Transmettre le nom de fichier
Fichier = Fichier + CHR$(0)          'Terminer le nom de fichier par <NUL>
Regs.ax = &H4E00              'AH = Numéro de fonction pour Search for first
```

```
Regs.cx = Attr                                  'Attributs recherchés
Regs.ds = VARSEG(Fichier)                'Adresse de segment du nom de fichier
Regs.dx = VARPTR(Fichier)                 'Adresse d'offset du nom de fichier

CALL INTERRUPTX(&H21, Regs, Regs)              'Appeler l'interruption DOS
IF (Regs.flags AND FCARRY) = 0 THEN                      'Tester Flag Carry
  FindFirst = TRUE                                'Non réglé: Fichier trouvé
ELSE                                              'Réglé: Fichier non trouvé
  FindFirst = FALSE
END IF

END FUNCTION

'*************************************************************************
'* FindNext : Lit l'entrée de répertoire suivante                        *
'* Entrée : Aucune                                                       *
'* Sortie : TRUE, si l'entrée a été trouvée sinon FALSE                  *
'* Infos    : La fonction ne peut être appelée que si l'appel de         *
'*            GetFirst a abouti, l'entrée est lue                        *
'*            dans les variables DirBuf                                  *
'*************************************************************************
'
FUNCTION FindNext%

DIM Regs AS RegType                  'Reg. proc. pour appels d'interruption

Regs.ax = &H4F00                'AH = 4F: Numéro de fonction pour for next
CALL INTERRUPT(&H21, Regs, Regs)               'Appeler l'interruption DOS

IF (Regs.flags AND FCARRY) = 0 THEN                      'Tester Flag Carry
  FindNext = TRUE                                 'Non réglé: Fichier trouvé
ELSE                                              'Réglé: Fichier non trouvé
  FindNext = FALSE
END IF

END FUNCTION

'*************************************************************************
'* Makeword : Transforme un Integer en Long car en BASIC                 *
'*            les opérations de déplacement de bits réalisées par des    *
'*            divisions en nombres entiers s'effectuent incorrectement   *
'*            avec un nombre négatif.                                    *
'* Entrée  : Le nombre Integer                                           *
'* Sortie  : Le nombre Long correspondant au modèle de bits              *
'*************************************************************************
'
FUNCTION MakeWord! (Nombre AS INTEGER)

IF Nombre < 0 THEN
  MakeWord = 65536! + Nombre
ELSE
  MakeWord = Nombre
END IF

END FUNCTION

'*************************************************************************
'* Mois   : Affiche un mois sous forme d'une chaîne                      *
'* Entrée : Le numéro du mois                                            *
'* Sortie : Le nom du mois sous forme d'une chaîne                       *
'*************************************************************************
'
FUNCTION Mois$ (Mon AS INTEGER)

SELECT CASE Mon
  CASE 1
    Mois$ = "Jan"
  CASE 2
    Mois$ = "Fev"
  CASE 3
    Mois$ = "Mar"
  CASE 4
    Mois$ = "Avr"
  CASE 5
    Mois$ = "Mai"
  CASE 6
    Mois$ = "Jun"
  CASE 7
    Mois$ = "Jul"
  CASE 8
    Mois$ = "Aug"
  CASE 9
    Mois$ = "Sep"
  CASE 10
    Mois$ = "Oct"
  CASE 11
    Mois$ = "Nov"
  CASE 12
    Mois$ = "Dec"
END SELECT

END FUNCTION

'*************************************************************************
'* PrintData : Afficher des informations à propos d'une entrée           *
'* Entrée    : Le DirBufType avec des informations de fichier            *
'* Sortie    : Aucune                                                    *
'*************************************************************************
'
SUB PrintData (DirBuf AS DirStruct)

DIM Compteur AS INTEGER                                  'Compteur de boucle

PRINT                          'Afficher une nouvelle ligne dans le tableau
LOCATE BildTop + EIntr + 3, 15       'dernière ligne de fenêtre de sortie
PRINT "";

Compteur = 1
WHILE MID$(DirBuf.DatName, Compteur, 1) <> CHR$(0)          'Jusqu'à <NUL>
  PRINT MID$(DirBuf.DatName, Compteur, 1);             'Afficher car. du nom
  Compteur = Compteur + 1                                'Caractère suivant
WEND

'-- Calculer et afficher la taille du fichier ------------------------

LOCATE BildTop + EIntr + 3, 30                    'Positionner le curseur
PRINT USING " ########  "; DirBuf.Size;

'-- Afficher la date et l'heure ---------------------------------------
LOCATE BildTop + EIntr + 3, 40
PRINT " "; Dat$(MakeWord(DirBuf.Date) AND 31); " ";                'jour
PRINT Mois$((MakeWord(DirBuf.Date) \ 32) AND 15);                  'mois
PRINT USING " ####"; (MakeWord(DirBuf.Date) \ 512) + 1980;         'année

LOCATE BildTop + EIntr + 3, 53
PRINT " "; Dat$(MakeWord(DirBuf.Time) \ 2048); " Heure";           'heure
PRINT " "; Dat$((MakeWord(DirBuf.Time) \ 32) AND 63);              'minute

'-- Afficher les attributs de fichier --------------------------------

LOCATE BildTop + EIntr + 3, 66
PRINT "";

Compteur = 1
WHILE (Compteur < 32)
  IF (ASC(DirBuf.Attrib) AND Compteur) <> 0 THEN          'Read- Only ?
    PRINT "X";
  ELSE
    PRINT " ";
  END IF
  Compteur = Compteur * 2
WEND
PRINT "";                                             'Cadre droit du champ

END SUB

'*************************************************************************
'* SetDTA  : Fixe l'adresse de DTA                                       *
'* Entrée : Adresses de segment et d'offset du buffer pour le DTA        *
'* Sortie : Aucune                                                       *
'*************************************************************************
'
SUB SetDTA (Segment AS LONG, Offset AS LONG)

DIM Regs AS RegTypeX                              'Registres du processeur

Regs.ax = &H1A00           'AH = &H1A Fixer le numéro de fonction de DTA
Regs.ds = Segment               'Adresse de segment dans le registre DS
Regs.dx = Offset                'Adresse d'offset dans le registre DX
CALL INTERRUPTX(&H21, Regs, Regs)          'Appeler l'interruption DOS

END SUB
```

Appel direct des fonctions 4Eh et 4Fh en Pascal et C

Dans les deux programmes suivants en Pascal et C, les fonctions 4Eh et 4Fh sont appelées directement sans qu'il soit nécessaire d'utiliser les procédures ou fonctions FINDFIRST et FINDNEXT prédéfinies.

Si la configuration de l'écran le permet, il est plus intéressant pour nous de travailler en Turbo Pascal. Ce langage fournit une commande permettant de définir une zone quelconque de l'écran comme une fenêtre de sortie. En C, il faut au contraire utiliser la fonction Scroll de l'interruption 10h du BIOS pour remonter à chaque fois d'une ligne la fenêtre des répertoires. Comme la sortie écran avec PRINTF() ne s'avère pas du tout commode et que les bibliothèques des compilateurs Borland et Microsoft semblent ignorer les fonctions de sortie écran, une fonction appelée PRINT est définie à cet effet dans le programme C. Elle fonctionne comme PRINTF() et attend que l'utilisateur indique la position de sortie de la chaîne de caractères et la couleur d'affichage outre les arguments habituels. Ces informations sont utiles pour transmettre directement la sortie dans la RAM vidéo.

Listing : DIRP1.PAS

```
{*********************************************************************}
{*                        D I R P 1 . P A S                          *}
{*-------------------------------------------------------------------*}
{*   Fonction        : Affiche tous les fichiers d'un répertoire     *}
{*                     quelconque y compris les sous-répertoires et  *}
{*                     noms de volumes sur l'écran.                  *}
{*                     La demande s'effectue à travers les deux      *}
{*                     fonctions DOS $4E et $4F.                     *}
{*                     Reportez-vous également au programme          *}
{*                     DIRP2.PAS.                                    *}
{*-------------------------------------------------------------------*}
{*   Auteur          : MICHAEL TISCHER                               *}
{*   Développé le    : 8.07.1988                                     *}
{*   Dernière modif. : 15.03.1992                                    *}
{*********************************************************************}

program DIRP1;

Uses Crt, Dos;                          { Insérer les unités CRT et DOS }

{-- Déclarations de type ---------------------------------------------}

type DirBufTyp = record     { Structure de données des fonc.$4E et $4F }
                 Reserve   : array [1..21] of char;
                 Attr      : byte;
                 Time      : integer;
                 Date      : integer;
                 Size      : longint;
                 Name      : array [1..13] of char
               end;

     MonVec    = array[1..12] of string[3]; {Tableau contenant les mois}

{-- Constantes -------------------------------------------------------}

const FA_ReadOnly  = $01;                       { Attributs de fichier }
      FAHidden     = $02;
      FA_SysFile   = $04;
      FA_VolumeID  = $08;
      FA_Directory = $10;
      FA_Archive   = $20;
      FA_AnyFile   = $3F;

      EINTR = 14;       { Autant d'entrées sont visibles simultanément }
      Mois : MonVec = ( 'Jan', 'Fev', 'Mar', 'Avr', 'Mai', 'Jun',
                        'Jul', 'Aug', 'Sep', 'Oct', 'Nov', 'Dec');

{*********************************************************************}
{* FindFirst: Lire la première entrée de répertoire                  *}
{* Entrée : Aucune                                                   *}
{* Sortie : TRUE ou FALSE, selon qu'une entrée a été trouvée         *}
{*          ou non                                                   *}
{*********************************************************************}

function FindFirst(NomFichier : string;       { Fichiers à rechercher }
                   Attribut   : integer) : boolean; {Attribut recherché }

var Regs : Registers;
                  { Registres du processeur pour l'appel d'interruption }

begin
  NomFichier := NomFichier + #0;          { Terminer NomFichier par NUL }
  Regs.ah := $4E;                 { Numéro de fonction pour Search for first }
  Regs.cx := Attribut;                                { Attributs recherchés }
  Regs.ds := Seg(NomFichier);      {Adresse de segment du nom de fichier}
  Regs.dx := succ(Ofs(NomFichier)); {Adresse d'offset du nom de fichier}
  MsDos( Regs );
                                                  { Appeler l'interruption DOS $21 }
  FindFirst := ( (Regs.flags and 1) = 0 )           { Tester Flag Carry }
end;

{*********************************************************************}
{* FindNext: Lire l'entrée de répertoire suivante                    *}
{* Entrée : Aucune                                                   *}
{* Sortie : true ou false selon qu'une entrée a été trouvée          *}
{*          ou non                                                   *}
{* Infos    : Cette fonction ne peut être appelée que si l'appel de  *}
{*            FinFirst a abouti                                      *}
{*********************************************************************}

function FindNext : boolean;

var Regs : Registers;
                  { Registres du processeur pour l'appel d'interruption}

begin
  Regs.ah := $4F;                 { Numéro de fonction pour Search for next }
  MsDos( Regs );                                  { Appeler l'interruption DOS $21 }
  FindNext := ( (Regs.flags and 1) = 0 )            { Tester Flag Carry }
end;

{*********************************************************************}
{* SetDTA: Spécifie l'adresse de DTA                                 *}
```

```
{(* Entrée : SEGMENT = Adresse de segment du nouveau DTA              *)
{(*          OFFSET  = Adresse d'offset du nouveau DTA               *)
{(* Sortie : Aucune                                                  *)
{(*******************************************************************)

procedure SetDTA(Segment.          {Nouvelle adresse de segment du DTA)
                 Offset  : integer);  {Nouvelle adresse d'offset du DTA}

var Regs : Registers;
                { Registres du processeur pour l'appel d'interruption }

begin
  Regs.ah := $1A;          { Spécifier le numéro de fonction pour le DTA }
  Regs.ds := Segment;          { Adresse de segment dans le registre DS }
  Regs.dx := Offset;           { Adresse d'offset dans le registre DX }
  MsDos( Regs );                    { Appeler l'interruption DOS $21 }
end;

{(*******************************************************************)
{(* PRINTDATA: Afficher les informations à propos d'une entrée       *)
{(* Entrée : DIRBUF = Structure de données avec informations de fichier*)
{(* Sortie : Aucune                                                  *)
{(*******************************************************************)

procedure PrintData( DirBuf : DirBufTyp );

var Compteur : byte;

begin
  writeln;          { Fait avancer la fenêtre d'une ligne vers le haut }

  Compteur := 1;              { Commencer par la première lettre du nom }
  while (DirBuf.Name[Compteur]<>#0) do          { Répéter jusqu'à NUL }
   begin
    write(DirBuf.Name[Compteur]);     { Afficher les caractères du nom }
    Compteur := succ(Compteur)          { Traiter le caractère suivant }
   end;

  GotoXY(13, EINTR);
  write('', DirBuf.Size:7);
  GotoXY(21, EINTR);
  write(' ', DirBuf.Date and 31:2,'.');          { Déterminer le jour }
  write (Mois[DirBuf.Date shr 5 and 15]);          { Afficher le mois }
  write(DirBuf.Date shr 9 + 1980:5);           { Déterminer l'année }
  GotoXY(34, EINTR);
  write('  ', DirBuf.Time shr 11:2, 'h');         { Déterminer l'heure }
  write(DirBuf.Time shr 5 and 63:2);            { Déterminer la minute }

  GotoXY(44, EINTR);
  write('');          { Trait de séparation avec le champ précédent }
  Compteur := 1;                                { Afficher les attributs }
  while ( Compteur < 32 ) do
    begin
      if (DirBuf.Attr and Compteur) <> 0 then write('X')
                                         else write(' ');
      Compteur := Compteur shl 1;
    end;
  write('');                          { Côté droit du cadre de fenêtre }
end;

{(*******************************************************************)
{(* ConfigEcran: Configure l'écran pour la sortie                    *)
{(*              du répertoire                                       *)
{(* Entrée : Aucune                                                  *)
{(* Sortie : Aucune                                                  *)
{(*******************************************************************)

procedure ConfigEcran;

var Compteur : integer;                                { Compteur de boucle }

begin
  ClrScr;                                              { Effacer l'écran }
  Window(14,(20-EINTR) shr 1+1,64,(20-EINTR) shr 1 +5+EINTR);
  GotoXY(1,1);     { Curseur dans le coin supérieur gauche de la fenêtre }

  write('');
  write(' Nom          Taille   Date       Heure  RHSVD');
  write('');

  for Compteur := 1 to EINTR do
    write('                                                ');
  write('');

  Window(15,(20-EINTR) shr 1+4,66,(20-EINTR) shr 1 +3+EINTR);
  GotoXY(1, EINTR);          { Curseur dans coin sup. gauche de fenêtre }
end;

{(*******************************************************************)
{(* Dir: Contrôle la lecture et la sortie du répertoire              *)
{(* Entrée : CHEMIN     = Chemin de recherche avec masque de fichier *)
{(*          ATTRIBUT = Attribut de recherche                        *)
{(* Sortie : Aucune                                                  *)
{(*******************************************************************)

procedure Dir( CHEMIN : string; Attr : byte );

var NbEntrees,                          { Nombre total des entrées trouvées }
    NbImage     : integer;              { Nombre d'entrées dans l'image }
    Touche      : char;     { Sert à attendre l'appui sur une touche }
    DirBuf      : DirBufTyp;                    { Reçoit une entrée Dir }

begin
  SetDTA(Seg(DirBuf), Ofs(DirBuf));          { DirBuf est le nouveau DTA }
  clrscr;                                                { Effacer l'écran }
  ConfigEcran;        { Construire l'écran pour la sortie du répertoire }

  NbImage := -1; { Aucune entrée n'est encore affichée dans la fenêtre }
  NbEntrees := 0;                { Aucune entrée n'est encore trouvée }
  if FindFirst( CHEMIN, Attr ) then    { Rechercher la première entrée }
    repeat
      NbEntrees := succ(NbEntrees);          { Une autre entrée trouvée }
      NbImage := succ(NbImage);      { Une autre entrée dans la fenêtre }
      if NbImage = EINTR then           { La fenêtre est-elle pleine? }
        begin                                                  { OUI }
          Window( 14, (20-EINTR) shr 1 + 5+ EINTR,
                  66 ,(20-EINTR) shr 1 + 6+ EINTR );
          GotoXY(1, 1);                { Curseur dans ligne inf. de fenêtre }
          TextBackground( LightGray );                   { Fond blanc }
          TextColor( Black );                          { Ecriture noire }
          write('          Veuillez appuyer sur une touche          ');
          Touche := ReadKey;                             { Lire la touche }
          GotoXY(1, 1);     { Curseur dans coin sup. gauche de fenêtre }
          TextBackground( Black );                          { Fond noir }
          TextColor( LightGray );                    { Ecriture en blanc }
          write('                                                  ');
          Window(15,(20-EINTR) shr 1+4,65,(20-EINTR) shr 1 +3+EINTR);
          GotoXY(1, EINTR);        { Replacer curseur sur anc. position }
          NbImage := 0;                          { Reprendre le calcul à 0 }
        end;
      PrintData( DirBuf );          { Afficher les données de l'entrée }
    until not(FindNext);                  { Y a-t-il une autre entrée ? }

  Window(14,(20-EINTR) shr 1 +5+EINTR,65,(20-EINTR) shr 1 +6+EINTR);
  GotoXY(1, 1);  { Curseur dans le coin supérieur gauche de la fenêtre }
  TextBackground( LightGray );                           { Fond blanc }
  TextColor( Black );                                  { Ecriture noire }
  write('                                                  ');

  GotoXY(2, 1);
  case NbEntrees of
    0 : write('Aucun fichier trouvé');
    1 : write('Un fichier trouvé');
    else write(NbEntrees,' fichiers trouvés')
  end;

  Window(1, 1, 80, 25);          { Reconfigurer l'écran comme une fenêtre }
end;

{(*******************************************************************)
{(**                          PROGRAMME PRINCIPAL                  **)
{(*******************************************************************)

begin
  case ParamCount of          { Réagir en fonction du nombre d'arguments }
    0  : Dir( '*.*', FA_AnyFile );    { Tous fichiers du rép. en cours }
    1  : Dir( ParamStr(1), FA_AnyFile );     { Afficher fich. spécifié }
    else writeln('Nombre de paramètres incorrect');
  end;
end.
```

Listing : DIRC1.C

```
/*****************************************************************
*                          D I R C 1                             *
*****************************************************************
*   Fonction        : Affiche tous les fichiers d'un répertoire  *
*                     quelconque y compris les sous-répertoires et*
*                     noms de volumes sur l'écran.               *
*                     La demande des fichiers s'effectue à travers*
*                     un appel direct des fonctions DOS          *
*                     4Eh et 4Fh.                                *
*                     Reportez-vous également au programme DIR2.C *
*****************************************************************
*   Auteur          : MICHAEL TISCHER                            *
*   Développé le    : 15.08.1987                                 *
*   Dernière modif. : 15.03.1992                                 *
*****************************************************************
*   Modèle de mémoire : SMALL                                    *
*****************************************************************/

/*-- Insérer les fichiers Include ---------------------------------*/

#include <dos.h>
#include <stdio.h>
#include <string.h>
#include <stdarg.h>
#include <stdio.h>
#include <conio.h>

/*-- Typedefs -----------------------------------------------------*/

typedef unsigned char BYTE;     /* Voilà comment se bricoler un OCTET */
typedef struct {            /* Structure DIR des fonctions 4Eh et 4Fh */
                BYTE          Reserve[21];
                BYTE          Attribut;
                unsigned int  Time;
                unsigned int  Date;
                unsigned long Size;
                char          Name[13];
} DIRSTRUCT;

/*-- Constantes ---------------------------------------------------*/

/*-- Attributs pour la recherche de fichiers ----------------------*/

#define ATTR_RDONLY 0x01                        /* Lecture seule */
#define ATTR_HIDDEN 0x02                        /* Caché */
#define ATTR_SYSTEM 0x04                        /* Système */
#define ATTR_LABEL  0x08                        /* Nom de Volume */
#define ATTR_DIREC  0x10                        /* Répertoire */
#define ATTR_ARCH   0x20                        /* Archive */
#define ATTR_ALL    0x3F                /* Tous types de fichiers */

#define TRUE   ( 0 == 0 )     /* Les constantes facilitent la lecture */
#define FALSE  ( 0 == 1 )                     /* Texte du programme */

#define EINTR  14     /* Autant d'entrées sont visibles simultanément */
#define EZ     ((20-EINTR) >> 1)     /* 1ère ligne de fenêtre du rép. */
#define NOF    0x07                  /* Ecriture blanche sur fond noir */
#define INV    0x70  /* Ecriture noire sur fond blanc (vidéo inverse) */

/*-- Macros -------------------------------------------------------*/

#ifdef MK_FP                            /* Macro MK_FP déjà définie? */
  #undef MK_FP                          /* Si oui, effacer la macro */
#endif

#define MK_FP(s,o) ((void far *) ( ((unsigned long)(s)<<16) |(o)))

/*****************************************************************
* PRINT : Similaire à PRINTF mais inscrit directement la chaîne de *
*         caractères dans la RAM vidéo                           *
* Entrée : COLONNE = Colonne de sortie                           *
*          LIGNE   = Ligne de sortie                             *
*          COULEUR = Couleur d'affichage                         *
*          STRING  = Pointeur sur la chaîne PRINTF               *
*          ...     = Autres arguments                            *
* Sortie : Aucune                                                *
*****************************************************************/

void Print( int Colonne, int Ligne, BYTE Couleur, char * String, ...)
{
  struct vr
  {                       /* Une position d'écran sous forme de 2 octets */
    BYTE caractere,                                   /* Code ASCII */
         attribut;                                    /* L'attribut approprié */
  } far  *lptr;                   /* Pointeur pour l'accès à la RAM vidéo */
  va_list parameter;              /* Liste de paramètres pour VA_... Macros */
  char    Sortie[255],                          /* Buffer pour chaîne formatée */
          *aptr = Sortie;                       /* Pour consulter la chaîne */
  union REGS          Registre;                 /* Registres d'appel d'int. */

  if( vioseg == 0 )                                     /* Premier appel? */
  {               /* Oui, déterminer l'adresse de segment de la RAM vidéo */
    Registre.h.ah = 0x0F;
    int86(0x10, &Registre, &Registre);
    vioseg = ( Registre.h.al == 7 ? 0xb000 : 0xb800 );
  }

  va_start( parameter, String );            /* Convertir les paramètres */
  vsprintf( Sortie, String, parameter );                 /* Formater */
  lptr = (struct vr far *)
         MK_FP( vioseg, ( Ligne * 80 + Colonne ) << 1 );

  for( ; *aptr ; )                                      /* Examiner la chaîne */
  {
    lptr->caractere = *aptr++;            /* Caractère dans la RAM vidéo */
    lptr++->attribut = Couleur;       /* Définir l'attribut du caractère */
  }
}

/*****************************************************************
* SCROLLUP: Fait avancer une zone écran d'une ou plusieurs lignes *
*           vers le haut ou l'efface                             *
* Entrée : NOMBRE  = Nombre de lignes à faire défiler            *
*          COULEUR = Couleur ou Attribut des lignes vides        *
*          COLSG   = Colonne du coin supérieur gauche de la zone *
*          LIGNSG  = Ligne du coin supérieur gauche de la zone   *
*          COLID   = Colonne du coin inférieur droit de la zone  *
*          LIGNID  = Ligne du coin inférieur droit de la zone    *
* Sortie : Aucune                                                *
* Infos  : Si 0 transmis (nombre), la zone est alors complétée   *
*          par des espaces                                       *
*****************************************************************/

void ScrollUp( BYTE Nombre, BYTE Couleur, BYTE ColSG,
               BYTE LignSG, BYTE ColID, BYTE LignID)
{
  union REGS  Registre;           /* Registres pour appel d'interruption */

  Registre.h.ah = 6;                                /* Numéro de fonction */
  Registre.h.al = Nombre;                           /* Nombre de lignes */
  Registre.h.bh = Couleur;                  /* Couleur de ligne(s) vide(s) */
  Registre.h.ch = LignSG;                   /* Définir les coordonnées de la */
  Registre.h.cl = ColSG;                    /* Fenêtre écran à faire défiler ou */
  Registre.h.dh = LignID;                                 /* à effacer */
  Registre.h.dl = ColID;
  int86(0x10, &Registre, &Registre);      /* Appeler l'interruption 10h */
}

/*****************************************************************
* SETPOS: Définit la position du curseur dans la page écran actuelle *
* Entrée : COLONNE = Nouvelle colonne du curseur                 *
*          LIGNE   = Nouvelle ligne du curseur                   *
* Sortie : Aucune                                                *
*****************************************************************/

void SetPos( BYTE Colonne, BYTE Ligne)
{
  union REGS Registre;            /* Registres pour appel d'interruption */

  Registre.h.ah = 2;                                /* Numéro de fonction */
  Registre.h.bh = 0;                                      /* Page écran */
  Registre.h.dh = Ligne;                            /* Ligne d'écran */
  Registre.h.dl = Colonne;                          /* Colonne d'écran */
  int86(0x10, &Registre, &Registre);      /* Appeler l'interruption 10h */
}

/*****************************************************************
* CLS : Effacer la page écran actuelle et placer curseur dans le coin *
*       supérieur gauche                                         *
* Entrée : Aucune                                                *
* Sortie : Aucune                                                *
*****************************************************************/

void Cls( void )
{
  ScrollUp(0, NOF, 0, 0, 79, 24);                   /* Effacer l'écran */
  SetPos(0, 0);                                     /* Placer le curseur */
}

/*****************************************************************
* CONFIGECRAN: Configure l'écran pour la sortie du répertoire    *
* Entrée : Aucune                                                *
* Sortie : Aucune                                                *
```

```
/*****************************************************************/

void ConfigEcran( void )
{
 BYTE  i;                                        /* Compteur de boucle */

 Cls();                                          /* Effacer l'écran */
 Print( 14, EZ, NOF,
        "");
 Print( 14, EZ+1, NOF,
        " Nom        Taille  Date       Heure  RHSVD");
 Print( 14, EZ+2, NOF,
        "");
 for (i = EZ+3; i < EZ+3+EINTR; i++)
  Print( 14, i, NOF,
         "                                                ");
 Print( 14, EZ+EINTR+3, NOF,
        "");
}

/*****************************************************************
* PRINTDATA : Sortir les informations à propos d'une entrée      *
* Entrée : SAISIE = Ptr sur une structure Dir avec des informations *
*                   de fichier                                   *
*          LIGNE = Ligne d'écran de la saisie                    *
* Sortie : Aucune                                               *
*****************************************************************/

void PrintData(DIRSTRUCT *Saisie, BYTE Ligne)
{
 BYTE        i, j;                               /* Compteur de boucle */
 static char *Mois[] =          /* Vecteur avec pointeurs sur les mois */
        {
          "Jan", "Fév", "Mar", "Avr", "Mai", "Jun",
          "Jul", "Aut", "Sep", "Oct", "Nov", "Déc"
 };

         /*-- Afficher les informations de fichier ----------------*/

 Print( 15, Ligne, NOF, "%s", Saisie->Name);

 Print( 28, Ligne, NOF, "%7lu", Saisie->Size);
 Print( 36, Ligne, NOF, " %2d %s %4d", Saisie->Date & 31,
        Mois[((Saisie->Date >> 5) & 15) - 1],
        (Saisie->Date >> 9) + 1980);
 Print(49, Ligne, NOF, "  %2d:%2d", Saisie->Time >> 11,
                                    (Saisie->Time >> 5) & 63 );

 for( i = j = 1; i <= 16; i <<= 1, ++j )      /* Attributs de fichier */
   Print( 58+j, Ligne, NOF, "%c", (Saisie->Attribut & i) ? 'X' : ' ' );
}

/*****************************************************************
* FINDFIRST: Lire la première entrée de répertoire              *
* Entrée : CHEMIN    = Ptr sur chemin de rech. avec masque fichier *
*          ATTRIBUT = Attributs de recherche                    *
* Sortie : TRUE, si une entrée a été trouvée sinon FALSE        *
* Infos    : L'entrée est lue dans le DTA                       *
*****************************************************************/

BYTE findfirst( char *Chemin, BYTE Attribut )
{
 union REGS   Registre;         /* Registres pour appel d'interruption */
 struct SREGS Segments;           /* Reçoit les registres de segment */

 segread(&Segments);       /* Lire le contenu des registres de segment */
 Registre.h.ah = 0x4E;           /* Numéro de fonction pour FindFirst */
 Registre.x.cx = Attribut;                    /* Attributs recherchés */
 Registre.x.dx = (unsigned int) Chemin;  /* Offset du chemin de rech. */
 intdosx(&Registre, &Registre, &Segments);    /* Interruption DOS 21h */
 return( !Registre.x.cflag );     /* Carry-Flag = 0: Un fichier trouvé */
}

/*****************************************************************
* FINDNEXT : Lire l'entrée de répertoire suivante               *
* Entrée   : Aucune                                             *
* Sortie   : TRUE, si une entrée a été trouvée sinon FALSE      *
* Infos    : L'entrée est lue dans le DTA                       *
*****************************************************************/

BYTE findnext( void )
{
 union REGS Registre;           /* Registres pour appel d'interruption */

 Registre.h.ah = 0x4F;            /* Numéro de fonction pour FindNext */
 intdos(&Registre, &Registre);      /* Appeler l'interruption DOS 21h */
 return( !Registre.x.cflag );     /* Carry-Flag = 0: Un fichier trouvé */
}

/*****************************************************************
* SETDTA : Place le DTA sur une variable du segment de données  *
* Entrée : OFFSET = Offset du DTA dans le segment de données    *
* Sortie : Aucune                                               *
*****************************************************************/

void SetDTA( unsigned int Offset )
{
 union REGS   Registre;         /* Registres pour appel d'interruption */
 struct SREGS Segments;           /* Reçoit le registre de segment */

 segread(&Segments);       /* Lire le contenu des registres de segment */
 Registre.h.ah = 0x1A;   /* Définir le numéro de fonction pour le DTA */
 Registre.x.dx = Offset;         /* Adresse d'offset dans le registre DX */
 intdosx(&Registre, &Registre, &Segments);    /* Interruption DOS 21h */
}

/*****************************************************************
* DIR    : Contrôle la lecture et la sortie du répertoire       *
* Entrée : CHEMIN  = Ptr sur chemin de rech. avec masque fichier *
*          ATTRIBUT = Attributs de recherche                    *
* Sortie : Aucune                                               *
*****************************************************************/

void Dir(char *Chemin, BYTE Attribut )
{
 int       NbEntrees,             /* Nombre total des entrées trouvées */
           NbImage;                 /* Nombre d'entrées dans l'image */
 DIRSTRUCT Saisie;                       /* Une entrée de répertoire */

 SetDTA( (unsigned int) &Saisie );            /* SAISIE = nouveau DTA */
 ConfigEcran();    /* Construire l'écran pour la sortie de répertoire */

 NbImage = NbEntrees = 0;   /* Aucune entrée encore aff. dans fenêtre */
                            /* Aucune entrée n'a été encore trouvée */
 if (findfirst(Chemin, Attribut))    /* Rechercher la première entrée */
 {                                          /* Un fichier trouvé */
   do    /* Sortir le fichier et rechercher le suivant avec GetNext() */
   {
     PrintData(&Saisie, EZ+EINTR+2);              /* Sortir l'entrée */
     if (++NbImage == EINTR )          /* La fenêtre est-elle pleine? */
     {
       NbImage = 0;                  /* Compléter à nouveau une fenêtre */
       Print(14, EZ+4+EINTR, INV,
             "        Veuillez appuyer sur une touche         ");
       getch();                      /* Attendre l'appui sur une touche */
       Print(14, EZ+4+EINTR, NOF,
             "                                                ");
     }
     ScrollUp(1, NOF, 15, EZ+3, 63, EZ+2+EINTR);
     Print(15, EZ+2+EINTR, NOF,
           "                                                ");
     ++NbEntrees;
   }
   while ( findnext() ); /* Fin de boucle s'il n'y a plus de fichiers */
 }

 SetPos(14, EZ+4+EINTR);
 switch (NbEntrees)
 {
   case 0  : printf("Pas de fichier trouvé\n");
             break;
   case 1  : printf("Un fichier trouvé\n");
             break;
   default : printf("%d Fichiers trouvés\n", NbEntrees);
             break;
 }
}

/*****************************************************************
**                        PROGRAMME PRINCIPAL                    **
*****************************************************************/

void main( int Nombre, char *Arguments[] )
{
 switch ( Nombre )        /* Réagir en fonction du nombre d'arguments */
 {                                                    /* transmis */
   case 1  : Dir( "*.*", ATTR_ALL ); /* Afficher tous les fichiers du */
             break;                        /* répertoire en cours */
   case 2  : Dir( Arguments[1], ATTR_ALL );  /* Aff. ts fich. du rép. */
             break;                                  /* spécifié */
   default : printf("Nombre de paramètres incorrect\n");
 }
}
```

Utilisation des fonctions/procédures prédéfinies en Pascal et C

Les deux programmes DIRP2.PAS et DIRC2.C utilisent les fonctions FINDFIRST et FINDNEXT prédéfinies de Turbo Pascal ou des divers compilateurs C énumérés dans ce livre. Cela facilite naturellement l'affichage du répertoire parce qu'on n'a pas par exemple besoin de s'inquiéter de l'emplacement du DTA.

Dans les deux cas, les procédures/fonctions utilisent les fonctions DOS 4Eh et 4Fh définies par les compilateurs pour recevoir les informations sur les fichiers. En Turbo Pascal, cette structure de données s'appelle SEARCHREC. Elle est définie dans l'unité DOS en même temps que FINDFIRST et FINDNEXT. Elle renferme également les diverses constantes permettant de fixer confortablement les différents flags afin de spécifier l'attribut de fichier.

Mais la structure SEARCHREC regroupe les champs heure et date dans un paramètre longint, ce qui oblige quelque peu à chercher une astuce pour atteindre les différents Words. Le programme DIRP2.PAS montre comment y parvenir. Mais n'oubliez pas que dans la version Pascal, FINDFIRST et FINDNEXT ne sont pas des fonctions mais des procédures. Avant d'appeler la fonction, il convient donc de vérifier si un fichier peut être trouvé en déterminant la variable globale DOSERROR. Turbo Pascal y conserve notamment le contenu du registre AX après l'appel des fonctions 4Eh et 4Fh. La valeur obtenue est en effet 0 si un fichier est trouvé.

Listing : DIRP2.PAS

```
{*********************************************************************}
{*                          D I R P 2                                *}
{*-------------------------------------------------------------------*}
{*   Fonction       : Affiche les fichiers d'un rép. quelconque      *}
{*                    y compris sous-répertoires et noms de volumes  *}
{*                    sur l'écran.                                   *}
{*                    La demande s'effectue avec les deux fonctions  *}
{*                    Pascal FirstFirst et FindNext de l'unité DOS.  *}
{*                    Reportez-vous également au programme           *}
{*                    DIRP1.PAS.                                     *}
{*-------------------------------------------------------------------*}
{*   Auteur         : MICHAEL TISCHER                                *}
{*   Développé le   :  8.07.1988                                     *}
{*   Dernière modif. : 15.03.1992                                    *}
{*********************************************************************}

program DIRP2;

Uses Crt, Dos;                          { Insérer les unités CRT et DOS }

{-- Déclarations de type ---------------------------------------------}

type MonVec    = array[1..12] of string[3];        { Tableau des mois }

{-- Constantes -------------------------------------------------------}

const EINTR = 14;       { Autant d'entrées sont visibles simultanément }
      Mois : MonVec = ( 'Jan', 'Fev', 'Mar', 'Avr', 'Mai', 'Jun',
                        'Jul', 'Aug', 'Sep', 'Oct', 'Nov', 'Dec');

{*********************************************************************}
{* PRINTDATA: Afficher les informations à propos d'une entrée        *}
{* Entrée : DIRBUF = Structure de données avec informations de fichier*}
{* Sortie : Aucune                                                   *}
{*********************************************************************}

procedure PrintData( DirBuf : SearchRec );

var Compteur : byte;
    Date,                       { Pour tronquer le champ TIME en SearchRec }
    Time     : word;

type longrec = record           { Pour découper un mot LONG en deux Word }
                 LoWord,
                 HiWord  : word
               end;

begin
  writeln;              { Fait avancer la fenêtre d'une ligne vers le haut }

  write( DirBuf.Name );         { Le nom a été déjà converti en Pascal }

  GotoXY(13, EINTR);
  write('', DirBuf.Size:7);

  Date := longrec(DirBuf.Time).HiWord;    { Date et Heure de SearchRec }
  Time := longrec(DirBuf.Time).LoWord;

  GotoXY(21, EINTR);
  write(' ', Date and 31:2,' ');                  { Déterminer le jour }
  write (Mois[Date shr 5 and 15]);                  { Afficher le mois }
  write(Date shr 9 + 1980:5);                      { Déterminer l'année }
  GotoXY(34, EINTR);
  write('  ', Time shr 11:2, 'h');                 { Déterminer l'heure }
  write(Time shr 5 and 63:2);                     { Déterminer la minute }

  GotoXY(44, EINTR);
  write('');              { Trait de séparation avec le champ précédent }
  Compteur := 1;                                  { Afficher les attributs }
  while ( Compteur < 32 ) do
    begin
      if( DirBuf.Attr and Compteur) <> 0 then
        write('X')
      else
        write(' ');
      Compteur := Compteur shl 1;
```

```
    end;
  write('');                          { Côté droit du cadre de fenêtre }
end;

{*********************************************************************}
{* ConfigEcran: Configure l'écran pour la sortie du                  *}
{*              répertoire                                           *}
{* Entrée : Aucune                                                   *}
{* Sortie : Aucune                                                   *}
{*********************************************************************}

procedure ConfigEcran;

var Compteur : integer;                          { Compteur de boucle }

begin
 ClrScr;                                          { Effacer l'écran }
 Window(14,(20-EINTR) shr 1+1,64,(20-EINTR) shr 1 +5+EINTR);
 GotoXY(1,1);       { Curseur dans le coin supérieur gauche de l'écran }

 write('');
 write(' Nom         Taille   Date       Heure  RHSVD');
 write('');

 for Compteur := 1 to EINTR do
   write('                                          ');
 write('');

 Window(15,(20-EINTR) shr 1+4,66,(20-EINTR) shr 1 +3+EINTR);
 GotoXY(1, EINTR);          { Curseur dans coin sup. gauche de l'écran }
end;

{*********************************************************************}
{* Dir: Contrôle la lecture et la sortie du répertoire               *}
{* Entrée : CHEMIN    = Chemin de recherche avec masque de fichier   *}
{*          ATTRIBUT  = Attribut de recherche                        *}
{* Sortie : Aucune                                                   *}
{*********************************************************************}

procedure Dir( Chemin : string; Attr : byte );

var NbEntrees,                       { Nombre total des entrées trouvées }
    NbImage      : integer;          { Nombre d'entrées dans l'image }
    Touche       : char;     { Sert à attendre l'appui sur une touche }
    DirBuf       : SearchRec;              { Reçoit une entrée Dir }

begin
 clrscr:                                          { Effacer l'écran }
 ConfigEcran:       { Construire l'écran pour la sortie de répertoire }

 NbImage := -1; { Aucune entrée n'est encore affichée dans la fenêtre }
 NbEntrees := 0;             { Aucune entrée n'a été encore trouvée }
 FindFirst( Chemin, Attr, DirBuf );   { Rechercher la première entrée }
 if DOSError = 0 then
  repeat
   NbEntrees := succ(NbEntrees);          { Une autre entrée trouvée }
   NbImage := succ(NbImage);      { Une autre entrée dans la fenêtre }
   if NbImage = EINTR then             { La fenêtre est-elle pleine? }
     begin                                                  { OUI }
       Window(14, (20-EINTR) shr 1 + 5 + EINTR,
              66, (20-EINTR) shr 1 + 6+ EINTR );
       GotoXY(1, 1);                { Curseur sur ligne inf. de fenêtre }
       TextBackground( LightGray );                  { Fond blanc }
       TextColor( Black );                       { Ecriture en noir }
       write('          Veuillez appuyer sur une touche          ');
       Touche := ReadKey;                          { Lire la touche }
       GotoXY(1, 1);     { Curseur dans coin sup. gauche de fenêtre }
       TextBackground( Black );                        { Fond noir }
       TextColor( LightGray );                   { Ecriture en blanc }
       write('                                                  ');
       Window(15,(20-EINTR) shr 1+4,65,(20-EINTR) shr 1 +3+EINTR);
       GotoXY(1, EINTR);          { Replace curseur sur ancienne pos. }
       NbImage := 0;                          { Reprendre le calcul à 0 }
     end;
   PrintData( DirBuf );              { Afficher les données de l'entrée }
   FindNext( DirBuf );                  { Rechercher le fichier suivant }
  until DOSError <> 0;     { jusqu'à ce qu'il n'y ait plus de fichier }

 Window(14,(20-EINTR) shr 1 +5+EINTR,65,(20-EINTR) shr 1 +6+EINTR);
 GotoXY(1, 1);  { Curseur dans le coin supérieur gauche de la fenêtre }
 TextBackground( LightGray );                          { Fond blanc }
 TextColor( Black );                                { Ecriture en noir }
 write('                                                          ');

 GotoXY(2, 1);
 case NbEntrees of
   0 : write('Aucun fichier trouvé');
   1 : write('Un fichier trouvé');
   else write(NbEntrees,' fichiers trouvés')
 end;

 Window(1, 1, 80, 25);          { Reconfigurer l'écran comme une fenêtre }
end;

{*********************************************************************}
{**                          PROGRAMME PRINCIPAL                    **}
{*********************************************************************}

begin
 case ParamCount of          { Réagir en fonction du nombre d'arguments }
   0  : Dir( '*.*', AnyFile ); { Tous fichiers ds répertoire en cours }
   1  : Dir( ParamStr(1), AnyFile );   { Afficher le fichier spécifié }
   else writeln('Nombre de paramètres incorrect');
 end;
end.
```

Le programme C DIRC2.C souffre quelque peu des différences existantes entre les bibliothèques des compilateurs Microsoft et Borland concernant la création des interfaces DOS API. Ce domaine n'étant pas standardisée selon la norme ANSI, chaque fabricant de compilateur n'en fait qu'à sa guise.

Dans les deux familles de compilateurs, FINDFIRST et FINDNEXT ne portent pas seulement des noms différents mais travaillent également avec des structures de données différentes dont les champs sont désignés avec des noms variés. Il n'empêche que le programme DIRC2.C peut être traduit sous les compilateurs des deux familles. Pour obtenir ce résultat, tous les noms relatifs aux compilateurs sont enregistrés à l'aide de macros définies par rapport au compilateur concerné. Le listing suivant illustre mieux ce point de vue.

Listing : DIRC2.C

```
/*****************************************************************
*                          D I R C 2                             *
*****************************************************************
*   Fonction        : Affiche tous les fichiers d'un répertoire  *
*                     y compris les sous-répertoires et noms     *
*                     de volumes sur l'écran.                    *
*                     La demande des fichiers s'effectue à travers *
*                     les fonctions prédéfinies C.               *
*                     Reportez-vous également à DIR1.C           *
*****************************************************************
*   Auteur          : MICHAEL TISCHER                            *
*   Développé le    : 15.10.1991                                 *
*   Dernière modif. : 15.03.1992                                 *
*****************************************************************
*   Modèle de mémoire : SMALL                                    *
*****************************************************************/

/*--- Insérer les fichiers Include ---------------------------*/

#include <dos.h>
#include <stdio.h>
#include <string.h>
#include <stdarg.h>
#include <stdio.h>
#include <conio.h>

#ifdef __TURBOC__                          /* Compilateur Turbo C? */
  #include <dir.h>                 /* Insérer les fonctions de répertoire */
#endif

/*--- Typedefs -----------------------------------------------*/

typedef unsigned char BYTE;     /* Voilà comment se bricoler un OCTET */

/*--- Constantes ---------------------------------------------*/

#define TRUE   ( 0 == 0 )     /* Les constantes facilitent la lecture */
#define FALSE  ( 0 == 1 )                  /* du texte du programme */

#define EINTR  14     /* Autant d'entrées sont visibles simultanément */
#define EZ     ((20-EINTR) >> 1)  /* 1ère ligne de fenêtre répertoire */
#define NOF    0x07                /* Ecriture blanche sur fond noir */
#define INV    0x70  /* Ecriture noire sur fond blanc (vidéo inverse) */

#define FA_ALL 0x3F                          /* Tous types de fichiers */

/*--- Macros -------------------------------------------------*/

#ifdef MK_FP                           /* Macro MK_FP déjà définie? */
  #undef MK_FP                          /* Si oui, effacer la macro */
#endif

#define MK_FP(s,o) ((void far *) (((unsigned long) (s)<<16)|(o)))

/*-- Constantes/macros pour rec. de fichiers avec compilateurs -------*/
/*-- Microsoft et Borland                                  -------*/

#ifdef __TURBOC__                          /* Compilateur Turbo C? */
  #define DIRSTRUCT                 struct ffblk
  #define FINDFIRST( path, buf, attr )  findfirst( path, buf, attr )
  #define FINDNEXT( buf )           findnext( buf )
  #define NAME                      ff_name
  #define ATTRIBUT                  ff_attrib
  #define TIME                      ff_ftime
  #define DATE                      ff_fdate
  #define SIZE                      ff_fsize
#else                                        /* Non, Microsoft C */
  #define DIRSTRUCT                 struct find_t
  #define FINDFIRST( path, buf, attr )  _dos_findfirst(path, attr, buf)
  #define FINDNEXT( buf )           _dos_findnext( buf )
  #define NAME                      name
  #define ATTRIBUT                  attrib
  #define TIME                      wr_time
  #define DATE                      wr_date
  #define SIZE                      size
#endif

/*****************************************************************
* PRINT : Comme PRINTF mais inscrit directement la chaîne        *
*         dans la RAM vidéo                                      *
* Entrée : COLONNE = Colonne de sortie                           *
*          LIGNE   = Ligne de sortie                             *
*          COULEUR = Couleur d'affichage                         *
*          STRING  = Pointeur sur la chaîne PRINTF               *
*          ...     = Autres arguments                            *
* Sortie : Aucune                                                *
*****************************************************************/

void Print( int Colonne, int Ligne, BYTE Couleur, char * String, ...)
{
  struct vr {                  /* Une position écran sous forme de 2 octets */
              BYTE caractere,                                  /* Le code ASCII */
                   attribut;                        /* L'attribut approprié */
            } far * lptr;       /* Pointeur pour l'accès à la RAM vidéo */
  va_list parameter;            /* Liste de paramètres pour macros VA_... */
  char    sortie[255],                        /* Buffer pour chaîne formatée */
          *aptr = sortie;                     /* Pour consulter la chaîne */
  static unsigned int vioseg = 0;
  union REGS          Registre;               /* Registres d'appel d'int */

  if( vioseg == 0 )                                    /* Premier appel? */
  {                 /* Oui, déterminer l'adresse de segment de la RAM vidéo */
    Registre.h.ah = 0x0F;
    int86(0x10, &Registre, &Registre);
    vioseg = ( Registre.h.al == 7 ? 0xb000 : 0xb800 );
  }

  va_start( parameter, String );            /* Convertir les paramètres */
  vsprintf( sortie, String, parameter );                 /* Formater */
  lptr = (struct vr far *)
         MK_FP( vioseg, ( Ligne * 80 + Colonne ) << 1 );

  for( ; *aptr ; )                                  /* Examiner la chaîne */
  {
    lptr->caractere = *aptr++;            /* Caractère dans la RAM vidéo */
    lptr++->attribut = Couleur;      /* Définir l'attribut du caractère */
  }
}

/*****************************************************************
* SCROLLUP: Fait avancer une zone d'écran d'une ou plusieurs lignes *
*           vers le haut ou l'efface                             *
* Entrée : NOMBRE  = Nombre de lignes à faire défiler            *
*          COULEUR = Couleur ou Attribut des lignes vides        *
*          COLSG   = Colonne du coin supérieur gauche de la zone *
*          LIGNSG  = Ligne du coin supérieur gauche de la zone   *
*          COLID   = Colonne du coin inférieur droit de la zone  *
*          LIGNID  = Ligne du coin inférieur droit de la zone    *
* Sortie : Aucune                                                *
* Infos  : Si 0 transmis comme nombre : zone écran complétée     *
*          par des espaces                                       *
*****************************************************************/

void ScrollUp( BYTE Nombre, BYTE Couleur, BYTE ColSG,
               BYTE LignSG, BYTE ColID, BYTE LignID)
{
  union REGS Registre;            /* Registres pour appel interruption */

  Registre.h.ah = 6;                              /* Numéro de fonction */
  Registre.h.al = Nombre;                           /* Nombre de lignes */
  Registre.h.bh = Couleur;                /* Couleur de ligne(s) vide(s) */
  Registre.h.ch = LignSG;                 /* Définir les coordonnées de la */
  Registre.h.cl = ColSG;                /* fenêtre écran à faire défiler ou */
  Registre.h.dh = LignID;                                 /* ou à effacer */
  Registre.h.dl = ColID;
  int86(0x10, &Registre, &Registre);     /* Appeler l'interruption 10h */
}

/*****************************************************************
* SETPOS : Définit la position du curseur dans la page écran     *
*          actuelle                                              *
* Entrée : COLONNE = Nouvelle colonne du curseur                 *
*          LIGNE   = Nouvelle ligne du curseur                   *
* Sortie : Aucune                                                *
*****************************************************************/

void SetPos( BYTE Colonne, BYTE Ligne)
{
  union REGS Registre;            /* Registres pour appel interruption */

  Registre.h.ah = 2;                              /* Numéro de fonction */
  Registre.h.bh = 0;                                    /* Page écran */
  Registre.h.dh = Ligne;                                /* Ligne écran */
  Registre.h.dl = Colonne;                            /* Colonne écran */
  int86(0x10, &Registre, &Registre);     /* Appeler l'interruption 10h */
}

/*****************************************************************
* CLS : Effacer page écran actuelle et placer curseur dans le coin *
*       supérieur gauche                                         *
* Entrée : Aucune                                                *
* Sortie : Aucune                                                *
*****************************************************************/
```

```
void Cls( void )
{
 ScrollUp(0, NOF, 0, 0, 79, 24);                    /* Effacer l'écran */
 SetPos(0, 0);                                      /* Placer le curseur */
}

/*********************************************************************
* CONFIGECRAN : Configure l'écran pour la sortie du                  *
*               répertoire                                           *
* Entrée : Aucune                                                    *
* Sortie : Aucune                                                    *
*********************************************************************/

void ConfigEcran( void )
{
 BYTE i;                                            /* Compteur de boucle */

 Cls();                                             /* Effacer l'écran */
 Print( 14, EZ, NOF,
        "");
 Print( 14, EZ+1, NOF,
        " Nom        Taille   Date       Heure  RHSVD");
 Print( 14, EZ+2, NOF,
        "");
 for (i = EZ+3; i < EZ+3+EINTR; i++)
  Print( 14, i, NOF,
         "                                                ");
 Print( 14, EZ+EINTR+3, NOF,
        "");
}

/*********************************************************************
* PRINTDATA: Afficher les informations à propos d'une entrée         *
* Entrée : SAISIE = Ptr sur une structure Dir avec des informations  *
*                   de fichier                                       *
*          LIGNE = Ligne écran de l'entrée                           *
* Sortie : Aucune                                                    *
*********************************************************************/

void PrintData(DIRSTRUCT *Saisie, BYTE Ligne)
{
 BYTE i, j;                                         /* Compteur de boucle */
 static char *Mois[] =          /* Vecteur avec pointeurs sur les mois */
        {
          "Jan", "Fév", "Mar", "Avr", "Mai", "Jun",
          "Jul", "Aug", "Sep", "Oct", "Nov", "Déc"
        };

         /*-- Afficher les informations de fichier ----------------*/

 Print(15, Ligne, NOF, "%s", Saisie->NAME);

 Print( 28, Ligne, NOF, "%7lu", Saisie->SIZE);
 Print( 36, Ligne, NOF, " %2d %s %4d", Saisie->DATE & 31,
        Mois[((Saisie->DATE >> 5) & 15) - 1],
        (Saisie->DATE >> 9) + 1980);
 Print(49, Ligne, NOF, "  %2d:%2d", Saisie->TIME >> 11,
                                    (Saisie->TIME >> 5) & 63 );

 for (i = j = 1; i <= 16; i <<= 1, ++j)       /* Attributs de fichier */
  Print( 58+j, Ligne, NOF, "%c", (Saisie->ATTRIBUT & i) ? 'X' : ' ' );
}

/*********************************************************************
* DIR : Contrôle la lecture et la sortie du répertoire               *
* Entrée : CHEMIN   = Ptr sur chemin de rech. avec masque fichier    *
*          ATTRIBUT = Attributs de recherche                         *
* Sortie : Aucune                                                    *
*********************************************************************/

void Dir( char *Chemin, BYTE Attribut )
{
 int       NbEntrees,                 /* Nombre total d'entrées trouvées */
           NbImage;                   /* Nombre d'entrées dans l'image */
 DIRSTRUCT Saisie;                    /* Une entrée de répertoire */

 ConfigEcran();    /* Construire l'écran pour la sortie de répertoire */

 NbImage = NbEntrees = 0; /* Aucune entrée encore affichée ds fenêtre */
                          /* Aucune entrée n'a été encore trouvée */
 if( !FINDFIRST(Chemin, &Saisie, Attribut) )     /* Rech. 1ère entrée */
 {                                               /* Un fichier trouvé */
  do    /* Sortir le fichier et rechercher le suivant avec GetNext() */
  {
    PrintData(&Saisie, EZ+EINTR+2);              /* Afficher l'entrée */
    if( ++NbImage == EINTR )          /* La fenêtre est-elle pleine? */
    {
      NbImage = 0;                   /* Remplir à nouveau une fenêtre */
      Print(14, EZ+4+EINTR, INV,
            "       Veuillez appuyer sur une touche          ");
      getch();                       /* Attendre l'appui sur une touche */
      Print(14, EZ+4+EINTR, NOF,
            "                                                ");
    }
    ScrollUp(1, NOF, 15, EZ+3, 63, EZ+2+EINTR);
    Print(15, EZ+2+EINTR, NOF,
          "                                                ");
    ++NbEntrees;
  }
  while( !FINDNEXT( &Saisie ) );  /* Fin s'il n'y a plus de fichiers */
 }

 SetPos(14, EZ+4+EINTR);
 switch (NbEntrees)
 {
   case 0  : printf("Aucun fichier trouvé");
             break;
   case 1  : printf("Un fichier trouvé");
             break;
   default : printf("%d fichiers trouvées", NbEntrees);
             break;
 }
}

/*********************************************************************/
/**                        PROGRAMME PRINCIPAL                      **/
/*********************************************************************/

void main( int Nombre, char *Arguments[] )
{
 switch ( Nombre )        /* Réagir en fonction du nombre d'arguments */
 {                                                          /* transmis */
   case 1  : Dir("*.*", FA_ALL );  /* Afficher tous fichiers du rép. */
             break;                                         /* en cours */
   case 2  : Dir( Arguments[1], FA_ALL ); /* Affiche fichiers du rép. */
             break;                                         /* spécifié */
   default : printf("Nombre de paramètres incorrect\n");
 }
}
```

21. Date et heure

L'horloge en temps réel de l'AT fournit la date et l'heure pour les routines BIOS et les fichiers. Quatre fonctions DOS, qui étaient disponibles depuis la version 1.0 du DOS, peuvent être utilisées pour accéder à l'heure et à la date.

Fonctions date et heure	
Fonction	**Utilisation**
2AH	Lire la date
2BH	Régler la date
2CH	Lire l'heure
2DH	Régler l'heure

DOS et l'horloge temps réel de l'AT

Dans les versions 3.2 et précédentes, ces quatre fonctions transmettaient les informations depuis et à partir des variables d'environnement DOS. Toutefois, dans les versions DOS 3.3 et supérieures, ces fonctions passaient les mêmes informations depuis et à partir de l'horloge temps réel alimentée par piles. L'information date et heure reste toujours disponible même si l'ordinateur est éteint. La date et l'heure sont gérées à travers des fonctions BIOS décrites dans le chapitre 11.

Cela est aussi possible dans les anciennes versions du DOS. Toutefois, le système doit être pourvu d'un driver horloge intégré car le DOS ne communique qu'avec un driver de périphérique sur carte nommé $CLOCK.

Lire et régler la date

Les fonctions 2AH et 2BH contrôlent la date en cours.

Fonction 2AH : Lire la date

En plaçant le numéro de fonction 2AH dans le registre AH, la date en cours est renvoyée dans les registres suivants :

Registres de sortie : Fonction 2AH	
Registre	**Contenu**
AL	Jour de semaine *
CX	Année
DH	Mois
DL	Jour
*	0=Dimanche, 1=Lundi, etc.

Fonction 2BH : Régler la date

La fonction 2BH place les informations de date dans les mêmes registres que ceux utilisés par la fonction 2AH. Cela est utile pour changer la date de création ou de modification d'un fichier.

Registres d'entrée : Fonction 2BH	
Registre	**Contenu**
AH	2BH
CX	Année
DH	Mois
DL	Jour

Le tableau précédent montre que la fonction 2BH nécessite la date actuelle (mais non le jour de la semaine). C'est la seule information dont le DOS a besoin.

Rappelez-vous que lorsque vous utilisez cette fonction, la plus ancienne date autorisée par le système est le 1er janvier 1980 (01-01-1980). Si une erreur se produit, vérifiez le registre AL. Si ce dernier contient la valeur 0, les données fournies par les autres registres sont valides. Si le registre AL contient la valeur 255, les données ne pourront pas être lues.

Lire et régler l'heure

Les fonctions 2CH et 2DH sont similaires aux fonctions 2AH et 2BH, sauf que 2CH et 2DH gèrent l'heure.

Fonction 2CH : Lire l'heure

La fonction 2CH lit l'heure en cours selon le modèle décrit dans le tableau qui suit. Placez le numéro 2Ch de l'interruption 21H dans le registre AH pour lire les registres suivants :

Registres de sortie : Fonction 2CH	
Registre	**Contenu**
CH	Heure
CL	Minute
DH	Seconde
DL	Centième de seconde

Fonction 2DH : Régler l'heure

La fonction 2DH règle l'heure. Le registre AL indique si la date est valide (par exemple, 36:48 engendre la valeur 255, qui signifie une erreur).

Registres d'entrée : Fonction 2CH	
Registre	**Contenu**
AH	2DH
CH	Heure
CL	Minute
DH	Seconde
DL	Centième de seconde

BIOS ou DOS ?

Vous pouvez utiliser le BIOS ou le DOS pour accéder à l'heure et à la date. Les deux méthodes ont leur avantage et elles sont disponibles sur tout PC.

22. Gestion de la mémoire

Une des tâches fondamentales d'un système d'exploitation consiste à gérer la mémoire vive (RAM). Cette dernière est le point de rencontre des composants les plus divers du système : drivers, programmes résidents, applications. Tout ce beau monde réclame sa part de gâteau et il faut veiller à ce qu'il n'y ait pas de conflit. DOS doit surveiller tous les programmes pour qu'ils ne s'écrasent pas mutuellement ou qu'ils ne tirent pas à eux la totalité de la mémoire disponible.

Ce chapitre décrit les mécanismes et les fonctions utilisés par DOS pour gérer la mémoire et vous y découvrirez même les actions secrètes menées en coulisse.

22.1. Gestion mémoire sous DOS

Le gestion de la mémoire assurée par DOS repose sur une fonction qui permet de réclamer des blocs mémoire d'une certaine taille. La mémoire ainsi allouée reste en possession du programme qui l'a demandée et reçue, jusqu'à ce qu'il la restitue par le moyen d'une autre fonction réservée à cet usage. Ce n'est qu'à partir de ce moment que le bloc libéré est à la disposition des autres programmes.

Quatre fonctions DOS en tout sont impliquées dans la gestion de la mémoire :

Fonction	Signification
48h	Alloue de la mémoire
49H	Libère de la mémoire
4Ah	Modifie la taille d'un bloc mémoire
58h	Lit ou fixe la stratégie d'allocation

Ce ne sont pas seulement les programmes qui tournent sous DOS qui exploitent les fonctions d'allocation et de libération de la mémoire. DOS lui-même en fait usage dans le module de chargement EXEC. Lorsqu'un programme doit être chargé et lancé, le chargeur EXEC commence par réserver la mémoire dans laquelle il va stocker le programme.

L'espace réclamé dépend du type de programme concerné. Pour les programmes COM, c'est l'intégralité de la mémoire vive qui est exigée. Pour les programmes EXE, la taille découle d'une indication portée dans l'en-tête du fichier EXE. Dans tous les cas, le chargeur ne peut mener à bien son travail que s'il obtient suffisamment de mémoire de DOS. Sinon il interrompt sa tâche en émettant le message bien connu :

```
Insufficient memory
```

Le chargeur n'est pas seul à réclamer de la mémoire, les programmes chargés ont aussi des exigences. Beaucoup ne se satisfont pas de la mémoire accordée par le chargeur, parce qu'ils n'ont pas suffisamment de place pour ranger toutes leurs variables et leurs buffers dans les segments de données et de code prévus initialement. En effet la taille de l'espace à consacrer aux variables et aux buffers se décide souvent au moment de l'exécution, à la suite de diverses saisies effectuées par l'opérateur ou sous l'influence de toutes sortes de facteurs imprévisibles.

Allocation de mémoire par la fonction 48h

Un programmeur en langage évolué n'a pas trop à se tracasser à propos de l'allocation dynamique de la mémoire. Le langage qu'il pratique lui facilite la tâche en prévoyant la gestion de qui est appelé le "tas". Mais le développeur qui travaille en assembleur n'a pas d'autre choix que de collaborer avec DOS pour réserver un bloc mémoire. Il se servira à cet effet de la fonction 48h, qui reste la seule possibilité de demander de la RAM à DOS. Pour mettre en service cette fonction, il faut lui communiquer le numéro 48H dans le registre AH et la taille requise dans BX.

Ce dernier paramètre doit se présenter non pas en nombre d'octets mais en nombre de paragraphes. Un paragraphe valant 16 octets, on voit que la mémoire ne peut se gérer que par multiple de 16 octets. La taille minimale d'un bloc est de 16 octets (BX=1) et la taille maximale de 1 Mo (BX=FFFFh). Notez qu'il n'est pas réaliste de demander 1 Mo à DOS car en mode réel la mémoire disponible n'a pas cette ampleur. Même une demande de 640, 512 voire 400 Ko sera excessive car il ne faut pas oublier que DOS et les drivers consomment eux-mêmes une partie de la mémoire. Sans parler des programmes résidents qui présentent la particularité de ne pas restituer la mémoire lorsqu'ils ont été exécutés, demurant par définition résidents.

Comme DOS ne peut pas toujours répondre aux désirs exprimés à travers la fonction 48h, tout appel à cette fonction doit être accompagné d'un examen soigné de l'indicateur de retenue. S'il est à 1, la mémoire réclamée n'était pas disponible. Le registre BX donne alors la taille de la mémoire effectivement disponible. Comme on peut s'y attendre, cette taille est exprimée en paragraphes, c'est-à-dire qu'il faut la multiplier par 16 pour la convertir en octets.

Si la taille retournée par BX suffit à satisfaire le programme,la fonction 48h doit être rappelée avec cette valeur : elel obtiendra ainsi la totalité de la mémoire présentement disponible.

Dans cette situation, il ne fait aucun doute que l'indicateur de retenue sera désarmé. AX contient alors l'adresse de segment du bloc mémoire que DOS a réservé à l'intention du programme demandeur. Ce bloc mémoire est alors sous le seul contrôle de son nouveau propriétaire, qui peut en faire l'usage qui lui plaît sans rendre de compte à personne.

Il faut cependant qu'il veille à respecter les limites de la taille allouée : ce n'est pas forcément l'iuntégralité du segment dont l'adresse a été renvoyée par la fonction 48h. Si par exemple un seul paragraphe a été demandé, le bloc s'étend de l'adresse AX:0000 à AX:000F et n'occupe donc que 16 octets du segment. Si un nombre plus élevé de paragraphes avait été demandé, le bloc aurait été plus long mais commencerait toujours à l'adresse de segment retournée avec l'offset 0000h.

Que se passe-t-il si on demande plus de 4096 paragraphes ce qui représente plus de 64 Ko ? Dans ce cas le bloc alloué occupe la totalité du segment, de AX:0000 à AX:FFFF et il peut même déborder sur les segments suivants.

Par principe le nouveau propriétaire du bloc alloué n'a le droit d'utiliser que la partie du segment qui lui a été expressément confiée. Tout espace situé au-delà de cette limite peut être attribué à d'autres programmes. Le non-respect de ce principe entraînerait selon toute vraisemblance un plantage du système.

Détermination de la mémoire disponible

Etant donné ses caractéristiques, la fonction 48h ne sert pas seulement à allouer de la mémoire mais aussi à renseigner un programme sur la mémoire disponible. DOS en effet ne possède pas d'autre fonction spécialisée dans cette interrogation.

On opère alors de la façon suivante. Il faut charger la valeur 0FFFFh dans le registre BX pour demander un bloc mémoire de 1 Mo. DOS est incapable par nature de satisfaire une telle exigence. C'est poyrquoi la fonction retourne dans le registre BX le nombre de paragraphes libres, d'où l'on déduit facilement le décompte en octets de la mémoire disponible

Libération de mémoire par la fonction 49h

Dès qu'un bloc mémoire n'est plus indispensable, il doit être libréré à l'aide de la fonction 49h. Cette règle est à respecter impérativement à la clôture d'un programme qui a demandé des blocs au moyen de la fonction 48h. En cas de non-respect, la mémoire abandonnée ne peut plus être réaffectée. Résultat final potentiel : les autres programmes peuvent ne plus avoir assez de mémoire pour tourner. Seul un redémarrage à chaud du système est susceptible de mettre fin à ce malheur.

En plus du numéro de la fonction à mettre dans AH, il faut communiquer à la fonction l'adresse de segment du bloc mémoire en la stockant dans ES. Il suffit d'utiliser à cette fin la valeur renvoyée en AX au moment de l'allocation du bloc.

Si le bloc mémoire a pu être libéré avec succès, la fonction 49h rend la main à l'appelant en mettant à 0 l'indicateur de retenue. Le bloc mémoire revient alors sous le contrôle

de DOS et il ne peut plus être manié par le programme dans la suite de son exécution. Si l'indicateur de retenue est à 1 à l'issue de l'appel, c'est qu'une erreur est survenue. Plusieurs phénomènes peuvent en être la cause, ils sont précisés par un code erreur placé en AX.

Si ce code vaut 9, l'adresse de segment communiquée est inexacte. Il n'y a pas de bloc mémoire précédemment alloué à cette adresse. Il est tout à fait possible que le bloc en question ait bien été en possession du programme mais il a peut être déjà été libéré. Il faut donc vérifier si la désallocation n'a pas été dupliquée.

Une autre erreur commune est signalée par le code 7 qui indique qu'un incident est survenu dans la gestion interne de la mémoire de DOS. Les reponsables tout désignés sont généralement les programmes qui débordent de leur territoire et écrasent des blocs en y écrivant des données sans autorisation. Dans ce cas, la mesure à prendre consiste à terminer le programme en cours de la façon habituelle mais il est conseillé à l'utilisateur de redémarrer ensuite le système.

Modification de la taille des blocs mémoire

La troisième fonction de gestion de la mémoire est la fonction 4Ah qui se charge de modifier la taille de blocs déjà alloués. Il est possible de les réduire ou de les augmenter. Mais évidemment en cas d'augmentation on risque de se voir opposer un refus en cas d'insuffisance de la mémoire disponible.

La fonction 4Ah demande qu'on lui communique en AH son numéro et en ES l'adresse de segment du bloc à modifier. Il faut par ailleurs mettre dans BX la nouvelle taille choisie pour le bloc, évaluée en paragraphes. Au retour de l'appel , les registres ont la même signification que dans le cas de la fonction 48h. En cas d'échec, DOS indique dans BX le maximum de paragraphes réservables. Bien entendu, cet incident ne peut se produire que dans le cadre d'une demande d'augmentation de la taille du bloc, une réduction étant par nature toujours possible.

22.2. TPA et UMB

Jusqu'à la version 5.0 de DOS la mémoire attribuée par la fonction 48h provenait exclusivement de la zone dite "Transient Program Area" = TPA, zone des programmes transitoires. Il s'agit là de l'espace qui s'étend de l'extrémité du noyau résident de DOS jusqu'à la frontière de 640 Ko. Si l'ordinateur possède moins de 640 Ko, la TPA est encore plus réduite mais de nos jours cette situation est plutôt rare.

La mémoire vive conventionnelle s'arrête normalement à la frontière des 640 Ko, car c'est là que commence l'espace reservé dans un PC à la mémoire d'écran, à la ROM du BIOS et aux autres extensions en ROM. Le cadre de pagination d'une carte EMS

est aussi hébergé dans cette zone. Les ordinateurs qui disposent de plus de 1 Mo sont généralement construits de telle sorte que 640 Ko de mémoire vive sont disposés en-deçà de la frontière des 640 Ko tandis que les 384 Ko restants sont établis derrière la limite de 1 Mo.

Davantage de mémoire grâce aux blocs de mémoire supérieure

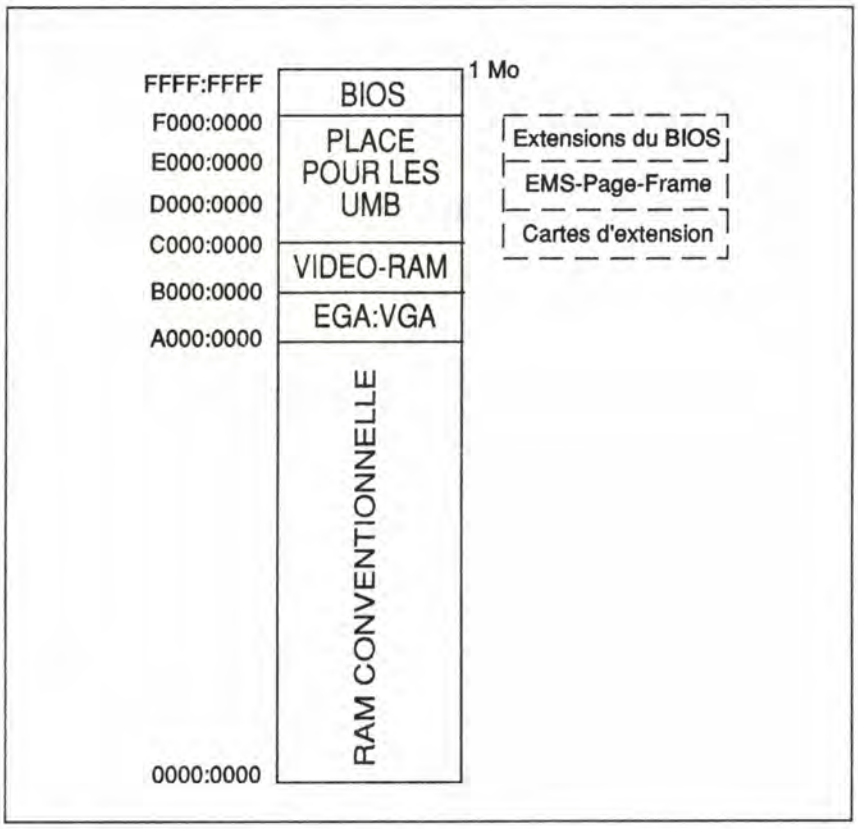
Les octets ainsi déplacés ne sont pas accessibles à DOS en mode réel, mais de toute façon entre les limites de 640 Ko et 1 Mo ils ne lui auraient servi à rien, du moins avant la version 5.0 de DOS. Mais à partir de cette version, DOS gère les "Upper Memory Blocks" (UMB) ou blocs de mémoire supérieure, qui sont précisément situés à cet endroit. Il faut dire qu'en général le domaine compris entre 640 Ko et 1 Mo n'est pas complètement occupé par la mémoire d'écran et les ROM. Entre les différentes cartes connectées il existe de nombreux interstices qui peuvent être affectées à de la mémoire vive.

Cette mémoire est extrêmement précieuse pour les programmes DOS car contrairement à la mémoire étendue située au-delà de la limite de 1 Mo, elle peut être exploitée sans problème en mode réel. Car un programme ne se préoccupe pas de savoir si la mémoire dont il dispose est située avant ou après la limite des 640 Ko : ce qui compte c'est qu'elle soit implantée en-dessous de la limite de 1 Mo.

La mémoire UMB est créée de diverses manières :

- ✓ par des dispositifs logiciels, comme par exemple dans le cas des cartes NEAT qui détournent de la RAM issue de la mémoire étendue dans le domaine UMB
- ✓ par des cartes UMB équipées spécialement de RAM et qui affectent cette mémoire au domaine UMB du PC
- ✓ par des programmes de gestion de la mémoire, tels que EMM386.EXE, 386-TO-The-Max ou QEMM qui déplacent de la mémoire étendue dans le domaine UMB.

C'est surtout cette dernière méthode qui se répand depuis le bon accueil réservé à DOS 5.0. Cette dernière version de DOS contient en effet un driver EMM386.EXE qui convient à tous les ordinateurs équipés de processeurs i386 ou i486.



Les blocs de mémoire supérieure (UMB) dans l'espace mémoire du PC

La quantité exacte de mémoire disponible au titre des UMB dépend de la configuration de chaque PC, c'est-à-dire de sa mémoire vidéo, des cartes d'extension connectées et de la ROM du BIOS. Elle peut aller jusqu'à 260 Ko mais se réduit parfois à 64 Ko. Indépendamment de sa taille, cette mémoire est souvent très fragmentée, contrairement à la mémoire située en dessous des 640 Ko. Il ne faut pas oublier en effet que par principe on comble des interstices.

Pour que DOS exploite les UMB, il faut exécuter l'instruction

```
DOS=HIGH, UMB
```

ou

```
DOS=LOW,UMB
```

dans le fichier CONFIG.SYS. Une fois qu'il a renconté cet ordre, DOS part à la recherche d'un driver XMS chargé de lui fournir les blocs UMB. Ainsi DOS ne se préoccupe pas lui-même de rechercher les emplacements physiques ouverts, c'est un

driver du type EMM386 qui relève ce défi. L'avantage de cette démarche est évident car DOS peut alors travailler aussi bien avec des cartes NEAT qu'avec n'importe quel programme de gestion de la mémoire, quelle que soit la manière dont ils trouvent les blocs UMB.

Il faut cependant que les drivers soutiennent le standard XMS que suit DOS appeler des UMB. Une fois le système d'exploitation lancé, DOS contrôle complètemennt la distribution des UMB et c'est à lui de répartir équitablement cette mémoire entre les drivers de périphériques, les programmes résidents et les applications.

Des commandes spéciales du genre DEVICEHIGH ou LOADHIGH provoquent le chargement en mémoire supérieure c'est-à-dire dans les UMB des drivers de périphériques ou programmes de DOS, pour peu évidemment que la possibilité existe. Du fait de la fragmentation de la mémoire supérieure, cette tâche n'est pas toujours facile. Même si l'espace disponible est important il est souvent encombré par la présence en son milieu de ROM ou de quoi que ce soit d'autre.

Ce sont surtout les petits programmes résidents et les drivers de périphériques qui sont d'excellents candidats à l'hébergement en mémoire supérieure.

Incorporation des UMB dans la gestion ordinaire de la mémoire

La mémoire supérieure n'entre pas seulement en service lorsque les programmes de DOS ou les drivers la réclament par DEVICEHIGH ou LOADHIGH. Elle peut être incorporée dans la gestion usuelle de la mémoire de sorte qu'un appel de la fonction 48h permet d'obtenir un bloc de mémoire supérieure. Ces blocs sont reconnaissables à leur adresse de segment qui est supérieure à A000h (équivalent de 640 K en hexadécimal.

La fonction 58h offre quatre options pour piloter l'inclusion des blocs de mémoire supérieure dans la gestion usuelle de la mémoire. Les deux premières options numérotées 00h et 01h existent déjà depuis la version 2.0 de DOS, tandis que le options 02h et 03h n'ont été introduites qu'à partir de la version 5.0.

Option	Signification
00h	Lit la stratégie d'allocation
01h	Fixe la stratégie d'allocation
02h	Teste si les blocs de mémoire supérieure sont incorporés
03h	Demande l'incorporation des blocs de mémoire supérieure

L'option 03h permet de demander à DOS d'inclure les UMB lorsqu'une allocation est demandée par la fonction 48h. Cette mesure est d'autant plus recommandée que la mémoire requise par un programme est plus importante.

Pour appeler l'option 03h de la fonction 58h, chargez la valeur 5802h dans le resgitre AX et la valeur 0 ou 1 dans BX. 1 veut dire que vous souhaitez incorporer les UMB, tandis que 0 restreint l'usage de la mémoire à la partie conventionnelle située en-dessous des 640 Ko.

Mais avant de déclencher cet appel, il est sage de lire l'état présent de l'allocation de la mémoire supérieure, pour pouvoir la rétablir à la fin de votre programme. Vous vous servirez à cet effet de l'option 02h de la fonction 58h. Il faut charger la valeur 5802h dans AX, il n'y a pas d'autre argument. Une fois la fonction exécutée, vous trouverez dans le registre AL une valeur de 0 ou de 1 qui s'interprète comme dans le cas de l'option 03H.

Après avoir appelé l'une ou l'autre des deux options citées, il est recommandé de tester l'indicateur de retenue. N'oubliez pas que ces nouveautés n'existent que depuis DOS 5.0 et qu'elles nécessitent la présence dans le fichier de configuration CONFIG.SYS d'une ligne spéciale.

S'il se trouve que l'indicateur de retenue est à 1, c'est que l'une des deux conditions mentionnées n'est pas satisfaite : les blocs de mémoire supérieure ne sont donc pas gérés.

Pour exploiter la mémoire supérieure de votre ordinateur, il est préférable de mettre en service des petits blocs mémoire. Si vous réclamez plusieurs blocs contigüs avec des centaines de Ko, vous ne les trouverez que rarement en mémoire supérieure. Par contre vous pourrez y exploiter avec profit une multitude de petits blocs. Au lieu de demander un bloc de 50 Ko, essayez plutôt de vous restreindre à 5 blocs de 10 Ko si votre algorithme le permet.

La stratégie d'allocation

Depuis la version 2.0 DOS est capable de suivre une stratégie d'allocation paramétrable par l'option 01h de la fonction 58h. La stratégie dont il est question ici concerne la recherche des blocs libres. Il existe trois codes possibles :

Code	Signification
00h	Recherche par le bas
01h	Recherche du meilleur ajustement
02h	Recherche par le haut

Les codes 00h et 02h sont faciles à comprendre. Lorsqu'il part en quête d'un bloc libre, DOS commence sa recherche tout en haut ou tout en bas de la mémoire vive. Il réserve le nombre demandé de paragraphes dans le premier bloc qui se présente avec une taille suffisante.

Le code 01h permet d'optimiser l'allocation. DOS parcourt l'intégralité de la mémoire et y recherche un bloc libre dont la taille soit exactement celle qui est requise, ou à défaut légèrement supérieure, mais le moins possible. Cette stratégie sous-entend que la mémoire est souvent fragmentée, notamment lorsque des programmes résidents restituent de la mémoire avant de s'installer définitivement. Il en résulte des morceaux de mémoire épars qui ne sont pas très importants.

Si un programme n'a besoin que de petits blocs, il est judicieux de les rechercher dans les parties fragmentées qui autrement ne seraient plus utiles, plutôt que d'entamer un grand bloc précieux.

Le code choisi pour la stratégie d'allocation est à mémoriser dans le registre BL. En plus des trois codes mentionnés plus haut, signalons que le bit 7 du registre BL, exploité indépendamment du code, joue également un rôle important. Il décide si la recherche des blocs libres doit commencer par les UMB ou à l'intérieur de la TPA. Si l'on opte pour la première alternative, il faut mettre le bit à 1, ce qui n'a vraiment de sens que si la mémoire supérieure est incluse dans la gestion générale de la mémoire par l'option 03h.

L'option 00h permet de prendre connaissance de la stratégie d'allocation courante. Le résultat, c'est-à-dire le code de la stratégie, est retourné dans le registre AL. A partir de DOS 5.0, il faut également tenir compte du bit 7 qui indique si la recherche des blocs libres commence ou non en mémoire supérieure.

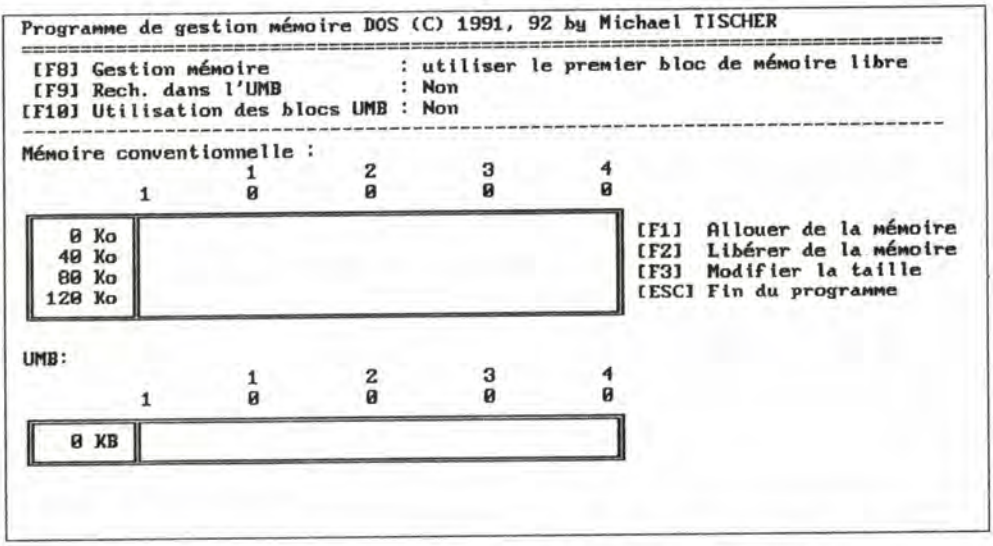
22.3. Visualisation de l'allocation mémoire

Pour vous permettre d'observer concrètement comment DOS alloue et libère la mémoire, j'ai mis au point deux programmes appelés MEMDEMOP.PAS et MEMDEMOC.PAS qui ont des fonctions et des structures pratiquement identiques.

L'un et l'autre ont pour préoccupation d'allouer, de libérer et de modifier des blocs mémoire à l'aide des fonctions DOS 48h, 49h et 4Ah, tout en affichant leur situation. Comme l'écran est trop petit pour donner une image complète de la TPA et des blocs de mémoire supérieure, le programme se consacre à deux zones bien précises : une zone de 160 Ko dans la TPA et une autre de 40 Ko dans la mémoire supérieure.

Cette restriction est rendue possible parce qu'en début de programme les deux zones sont allouées comme deux blocs contigus. Ensuite toute la mémoire restante est allouée sous forme de blocs de 1 Ko. Il n'existe alors plus aucun bloc libre qui atteigne 1 Ko, ni

dans la TPA ni dans la mémoire supérieure. Mais pourquoi cette valeur de 1 Ko ? Attendez, vous allez comprendre dans un moment.



Comment les programmes MEMDEMOP et MEMDEMOC visualisent l'allocation de la mémoire

L'étape suivante consiste à libérer à nouveau les deux grands blocs. Ils constituent alors les seules zones dans lesquelles DOS peut allouer de la mémoire, pour peu que la requête atteigne ou dépasse 1 Ko, à quoi s'appliquent précisément les deux programmes.

En réalité c'est l'utilisateur qui demande et libère de la mémoire. Les touches <F1> et <F3> sont préposées à cet usage, comme l'explique la légende de la figure. Quand on actionne ces touches le programme demande quel est le bloc concerné. Les blocs sont référencés par les lettres de A à Z, sans distinction entre majuscule et minuscule. Il existe donc 26 blocs à la disposition de l'utilisateur.

Indiquez la lettre souhaitée at tapez <Entrée>. Pour allouer de la mémoire, il faut indiquer une lettre correspondant à un bloc libre. Inversement, pour libérer ou modifier la taille d'un bloc il faut mentionner une lettre rattachée à un bloc alloué.

Lorsque vous allouez, agrandissez ou réduisez un bloc, le programme vous demande la taille souhaitée en Ko. Tapez le nombre que vous désirez et clôturez par <Entrée>.

Après ces opérations l'un et l'autre programme redessinent la fenêtre centrale de l'écran qui représente l'occupation de la mémoire. Chaque zone de la mémoire occupée par un bloc se signale par la lettre attachée à ce bloc. Les zones non allouée restent vierges. Chaque caractère dans les deux fenêtres correspond à 1 Ko de la TPA ou de la mémoire supérieure, ce que rappelle l'échelle affichée en bordure.

La fenêtre des UMB n'apparaît que si une zone de 40 Mo a réellement pu être réservée dans la mémoire supérieure. Si elle est absente, cela signifier que votre système ne gère pas les blocs de mémoire supérieure ou qu'il ne s'est trouvé aucun espace en un seul tenant de 40 Ko dans la mémoire supérieure.

Le mécanisme de l'allocation devient particulièrement intéresssant à observer lorsque vous jouez un peu avec les touches <F1>, <F2> et <F3>. Comme vous le voyez dans la partie supérieure de l'écran, vous pouvez choisir la stratégie de l'allocation et décider ou non de l'inclusion de la mémoire supérieure. A chaque fois que vous appuyez sur la touche concernée, l'état courant est inversé.

En changeant la stratégie d'allocation et la décision d'incluison, vous pouvez voir très distinctement comment les blocs sont établis dans la partie haute ou basse de la mémoire disponible et s'ils sont pris dans la TPA ou dans la mémoire supérieure.

Pour terminer l'un ou l'autre programme, appuyez simplement sur la touche <Esc>. Toute la mémoire précédemment allouée est alors libérée.

Listing : MEMDEMOP.PAS

```
{*********************************************************************}
{*                      MEMDEMOP.PAS                                 *}
{*-------------------------------------------------------------------*}
{*   Fonction       : Montre comment DOS gère la mémoire             *}
{*-------------------------------------------------------------------*}
{*   Auteur         : Michael TISCHER                                *}
{*   Développé le   : 08.10.1991                                     *}
{*   Dernière MAJ   : 20.03.1992                                     *}
{*********************************************************************}

{$M 8096, 0, 10240 }

program MEMDEMOP;

uses crt,                                       { Intègre l'unité CRT }
     dos;                                       { Intègre l'unité DOS }

{== Constantes ===================================================}

const
{-------- Numéros des fonctions de l'interruption 21h-----------------}

     GET_MEM      = $48;                  { Réserve de la mémoire vive }
     FREE_MEM     = $49;                   { Libère de la mémoire vive }
     CHANGE_MEM   = $4A;    { Modifie la taille d'une zone de mémoire }
     GET_STRATEGY = $5800;             { Lit la stratégie d'allocation }
     SET_STRATEGY = $5801;             { Fixe la stratégie d'allocation }
     GET_UMB      = $5802;     { Lit l'état d'inclusion des blocs UMB }
     SET_UMB      = $5803;             { Fixe l'inclusion des blocs UMB }

{-- Stratégies de recherche pour FixeStrategie ----------------------}

     CHERCHE_ENBAS    = $00;          { premier bloc de mémoire libre }
     CHERCHE_MEILLEUR = $01;          { meilleur bloc de mémoire libre }
     CHERCHE_ENHAUT   = $02;          { dernier bloc de mémoire libre }
     DABORD_UMB       = $80;                { Chercher dans la zone UMB }
                                      { utiliser avec CHERCHE_....) }

{-------- Constantes pour FixeUMB -----------------------------------}

     UMB_NON = $00;                { Ne tient pas compte des blocs UMB }
     UMB_OUI = $01;                             { Alloue des blocs UMB }

{------- Constantes pour Demo ---------------------------------------}

     TEST_TA        = 10240-1;           { 10239 paragraphes pour test }
     TEST_TA_UMB    = 2560-1;      { 2559 paragraphes d'UMB pour test }
     TEST_TA_KB     = 160;          { Environnement de test = 160 Ko }
     TEST_TA_UMB_KB = 40;          { Environnement de test UMB = 40 Ko }
     NBREBLOC       = 26;     { Nbre d'adresses pour aff. du résultat }

{-- Codes de touches pour saisie utilisateur ------------------------}

     ESC  = #27;                                  { Interruption par ESC }
     F1   = #59;                                  { Touche de fonction F1 }
     F2   = #60;                                  { Touche de fonction F2 }
     F3   = #61;                                  { Touche de fonction F3 }
     F8   = #66;                                  { Touche de fonction F8 }
     F9   = #67;                                  { Touche de fonction F9 }
     F10  = #68;                                  { Touche de fonction F10 }

{== Déclarations de types ========================================}

type  BlocTyp = record              { Gestion d'un bloc de mémoire alloué }
          Adresse,                                            { Segment }
          Taille : word;                          { Taille du bloc en Ko }
              end;

{== Typisierte Constantes ========================================}

const OuiNon     : array[ false..true ] of string =
          ( 'Non ', 'Oui ' );
      SText      : array[ CHERCHE_ENBAS..CHERCHE_ENHAUT ] of string =
          ( 'Utiliser le premier bloc de mémoire libre ',
            'Utiliser le meilleur bloc de mémoire libre',
            'Utiliser le dernier bloc de mémoire libre ' );
      ChampCouleur: array[ 0..1 ] of byte  =
          ( $07, $70 );
      TexteTouche : array[ 0..3 ] of string =
          ( ' [F1]  Allouer de la mémoire',
            ' [F2]  Libérer de la mémoire',
            ' [F3]  Modifier la taille ',
            ' [ESC] Fin du programme ' );

{== Variables globales ===========================================}

var Regs         : Registers;              { Registres d'interruptions }
    ChampAdresse : array[ 0..1000 ] of word;       { Champ de mémoires }
    NbAdresse    : word;                          { Nombre d'adresses }
    SegConv      : word;          { Adr. bloc de test dans mémoire conv. }
    UMBSeg       : word;            { Adr. bloc de test en mémoire sup. }
    BlocChamp    : array[ 0..NBREBLOC - 1 ] of BlocTyp;      { Mémoire }

{*********************************************************************
* DOS_GetMem : Réserve de la mémoire                                 *
* Entrée     : Taille mémoire souhaitée en paragraphes               *
* Sortie     : Segment du bloc de mémoire alloué ,                   *
*              Nombre de paragraphes alloués ou nombre maximal  des  *
*              paragraphes disponibles                               *
*********************************************************************}
```

```
{
procedure DOS_GetMem(     Ta  : word;
                      var Adr : word;
                      var Res : word );

begin
  with Regs do
    begin
      ah := GET_MEM;                              { Numéro de fonction }
      bx := Ta;                        { Nombre de paragraphes à réserver }
      msdos( Regs );                          { Déclenche l'interruption }
      if( flags and fcarry = 0 ) then                  { Appel réussi ? }
        begin
          Adr := regs.ax;       { Oui, restitue l'adresse et la taille }
          Res := Ta;
        end
      else                                                { Non ,erreur }
        begin
          Adr := 0;                              { pas de mémoire réservée }
          Res := bx;                          { Taille disponible maximale }
        end;
    end;
end;

{*****************************************************************
* DOS_FreeMem : Libère de la mémoire préalablement réservée      *
* Entrée      : Segment de la mémoire                            *
* Sortiee     : néant                                            *
*****************************************************************}

procedure DOS_FreeMem( Adr : word );

begin
  with Regs do
    begin
      ah := FREE_MEM;                             { Numéro de la fonction }
      es := Adr;                        { Adresse de la mémoire à libérer }
      msdos( Regs );                          { Déclenche l'interruption }
    end;
end;

{*****************************************************************
* DOS_ChangeMem : Change la taille d'un bloc réservé             *
* Entrée        : Ancien segment, nvelle taille souhaitée (paragraphes) *
* Sortie        : Segment du bloc alloué                         *
*                 Nombre de paragraphes alloués ou nombre maximal *
*                 de paragraphes disponibles                     *
*****************************************************************}

procedure DOS_ChangeMem(     Ta  : word;
                         var Adr : word;
                         var Res : word );
begin
  with Regs do
    begin
      ah := CHANGE_MEM;                           { Numéro de la fonction }
      bx := Ta;                        { Nombre de paragraphes à réserver }
      es := Adr;                  { Segment du bloc de mémoire à modifier }
      msdos( Regs );                          { Déclenche l'interruption }
      if( flags and fcarry = 0 ) then                  { Appel réussi ? }
        Res := Ta                                { Nouvelle taille mémoire }
      else                                                { Non, erreur }
        Res := bx;                           { Taille maximale disponible }
    end;
end;

{*****************************************************************
* LitStrategie : Lit la stratégie de gestion de la mémoire       *
* Entrée       : néant                                           *
* Sortie       : Type de stratégie                               *
*****************************************************************}

function LitStrategie : integer;

begin
  with Regs do
    begin
      ax := GET_STRATEGY;                 { Fixe le numéro de la fonction }
      msdos( Regs );                          { Déclenche l'interruption }
      LitStrategie := ax;                        { Retourne la stratégie }
    end;
end;

{*****************************************************************
* LitUMB       : Lit l'état d'inclusion des blocs UMB            *
* Entrée       : néant                                           *
* Sortie       : Indique si les blocs UMB sont pris en compte    *
* Info         : Disponible à partir de MS-DOS version 5.0       *
*****************************************************************}

function LitUMB : integer;

begin
  with Regs do
    begin
      ax := GET_UMB;                      { Fixe le numéro de la fonction }
      msdos( Regs );                          { Déclenche l'interruption }
      LitUMB := al;                                  { Retourne l'état }
    end;
end;

{*****************************************************************
* FixeStrategie: Fixe la stratégie d'allocation de la mémoire    *
* Entrée       : Nouveau type de stratégie souhaitée             *
* Sortie       : néant                                           *
*****************************************************************}

procedure FixeStrategie( Strategie : integer );

begin
  with Regs do
    begin
      ax := SET_STRATEGY;                 { Fixe le numéro de la fonction }
      bx := Strategie;
      msdos( Regs );                          { Déclenche l'interruption }
    end;
end;

{*****************************************************************
* FixeUMB      : Fixe l'état d'inclusion des blocs de mémoire UMB *
* Entrée       : Nouvel état d'inclusion souhaité                *
* Sortie       : néant                                           *
* Info         : Disponible à partir de la version 5.0 de MS-DOS *
*****************************************************************}

procedure FixeUMB( UMB : integer );

begin
  with Regs do
    begin
      ax := SET_UMB;                      { Fixe le numéro de la fonction }
      bx := UMB;
      msdos( Regs );                          { Déclenche l'interruption }
    end;
end;

{*****************************************************************
* AlloueMemoire  : Crée l'environnement du test                  *
* Entrée         : néant                                         *
* Sortie         : néant                                         *
*****************************************************************}

procedure AlloueMemoire;

var SegAdr    : word;                    { Segment de la mémoire allouée }
    Essai     : word;                     { Taille de mémoire souhaitée }
    Taille    : word;              { Taille de la zone de mémoire allouée }

begin
    (-- Alloue le  bloc de test-----------------------------------)

  FixeUMB( UMB_NON );                        { En mémoire conventionnelle }
  DOS_GetMem( TEST_TA, SegConv, Taille );                { Cherche le bloc}

  if ( SegConv = 0 ) then                                      { Erreur? }
    exit;                      { Pas de bloc de test, terminer la procédure }


  FixeUMB( UMB_OUI );                  { Bloc de test en mémoire supérieure }
  FixeStrategie( CHERCHE_ENHAUT or DABORD_UMB );
  DOS_GetMem( TEST_TA_UMB, UMBSeg, Taille );
  if( UMBSeg <> 0 ) and ( UMBSeg < $A000 ) then   { Pas de blocs UMB ? }
    begin
      DOS_FreeMem( UMBSeg );                              { Libère la mémoire }
      UMBSeg := 0;                                  { Absence de blocs UMB }
    end;

  (-- Alloue la mémoire conv. restante et la mémoire supérieure -------)
  (--    par blocs de 1 Ko                                          ---)

  Essai := 63;                  { Essaie d'abord d'allouer 15 paragraphes }
  NbAdresse := 0;          { Initialise le nombre de blocs de 1 Ko alloués }
  repeat
    DOS_GetMem( Essai, SegAdr, Taille );          { Réclame de la mémoire }
    if ( SegAdr <> 0 ) then                       { Est-elle disponible ? }
      begin
        ChampAdresse[ NbAdresse ] := SegAdr;        { Mémorise l'adresse }
        inc( NbAdresse );
      end;
  until( SegAdr = 0 );                                   { Tout est alloué }

    (-- 3ª Libère à nouveau les blocs de test--------------------------)

  if( SegConv > 0 ) then             { Mémoire conventionnelle allouée ? }
    begin                                                        { Oui }
      DOS_FreeMem( SegConv );
```

```
    dec( SegConv );                          { MCB également libéré }
  end;
  if( UMBSeg > 0 ) then                  { Mémoire supérieure allouée ? }
    begin                                                  { Oui }
      DOS_FreeMem( UMBSeg );
      dec( UMBSeg );                       { MCB également libéré }
    end;
end;

{*********************************************************************
* LibereMemoire   : Libère la mémoire allouée par AlloueMemoire      *
* Entrée          : néant                                            *
* Sortie          : néant                                            *
* Variables glob. : ChampAdresse/R                                   *
*********************************************************************}

procedure LibereMemoire;

var i : word;                                              { Compteur }

begin
  if( NbAdresse > 0 ) then           { A-t-on alloué de la mémoire ? }
    for i := 0 to NbAdresse - 1 do     { Oui, la libère bloc par bloc }
      DOS_FreeMem( ChampAdresse[ i ] );
end;

{*********************************************************************
* AfficheResultat : Affiche l'occupation de la mémoire               *
* Entrée          : AVECCADRE = TRUE, si le cadre doit également être *
*                               affiché                              *
* Sortie          : néant                                            *
*********************************************************************}

procedure AfficheResultat( AvecCadre : boolean );

var SChamp     : array[ 0..TEST_TA_KB - 1 ] of char;
    SChamp_UMB : array[ 0..TEST_TA_UMB_KB - 1 ] of char;
    i,j        : word;                                   { Compteurs }
    Position   : word;                        { Variable auxiliaire }
    Sdummy     : string;
    DerCara    : char;          { Mémorise le dernier caractère affiché }
    CoulCour   : byte;                { Couleur d'affichage courante }

begin
  fillchar( SChamp[ 0 ], TEST_TA_KB, #32 );       { Champs d'affichage }
  fillchar( SChamp_UMB[ 0 ], TEST_TA_UMB_KB, #32 );

  {-- Remplit le tableau de la mémoire ------------------------------}

  for i := 0 to NBREBLOC - 1 do
    begin
      if( BlocChamp[ i ].Adresse > $A000 ) then                { UMB? }
        begin
          Position := ( BlocChamp[ i ].Adresse - UMBSeg ) div 64;
          for j := 0 to BlocChamp[ i ].Taille div 64 do
            SChamp_UMB[ Position + j ] := chr( i + 65 );
        end
      else if( BlocChamp[ i ].Adresse > 0 ) then
        begin
          Position := ( BlocChamp[ i ].Adresse - SegConv ) div 64;
          for j := 0 to BlocChamp[ i ].Taille div 64 do
            SChamp[ Position + j ] := chr( i + 65 );
        end;
    end;

        {-- Dessine le cadre du tableau ----------------------------}

  if( AvecCadre ) then
    begin
      writeln( 'Mémoire conventionnelle :' );
      writeln( '          1         2         3         4 ' );
      writeln( '  1       0         0         0         0 ' );
      writeln( '' );
      for i := 0 to 3 do
        begin
          str( i * 40 : 3, Sdummy );
          writeln( ' ' + Sdummy +
                   ' Ko                                    ',
                   TexteTouche[ i ] );
        end;
      writeln( '' );
      if ( UMBSeg > 0 ) then
       begin
        writeln(#13#10'UMB:' );
        writeln('          1         2         3         4 ');
        writeln('  1       0         0         0         0 ');
        writeln('');
        writeln('   0 KB                                    ');
        writeln('');
       end
      else
        writeln( #13#10, 'Pas d''UMB disponible' );
    end;

  {-- Affiche le tableau de la mémoire conventionnelle ----------------}

  DerCara := #0;                                { Dernier caractère affiché }
  CoulCour := 0;                                { Dernière couleur affichée }
  for i := 0 to 3 do
    for j := 0 to 39 do
      begin
        if( DerCara <> SChamp[ i * 40 + j ] ) then  { Chang. couleur ? }
          begin
            CoulCour := ( CoulCour + 1 ) mod 2; { Nouveau code couleur }
            textcolor( ChampCouleur[ CoulCour ] shr 4 );
            textbackground( ChampCouleur[ CoulCour ] and $0F );
            DerCara := SChamp[ i * 40 + j ];        { Symbole d'égalité }
          end;
        gotoxy( j + 11, i + 11 );                { Affiche un caractère }
        write( SChamp[ i * 40 + j ] );
      end;

  {-- Affiche le tableau de la mémoire supérieure ---------------------}

  if ( UMBSeg > 0 ) then
    begin
      for j := 0 to 39 do
        begin
          if( DerCara <> SChamp_UMB[ j ] ) then     { Chang. couleur ? }
            begin
              CoulCour := ( CoulCour + 1 ) mod 2;        { code couleur }
              textcolor( ChampCouleur[ CoulCour ] shr 4 );
              textbackground( ChampCouleur[ CoulCour ] and $0F );
              DerCara := SChamp_UMB[ j ];          { Symbole d'égalité }
            end;
          gotoxy( j + 11, 21 );
          write( SChamp_UMB[ j ] );
        end;
    end;
  textcolor( $07 );
  textbackground( $00 );
end;

{*********************************************************************
* Demo        : Démonstration de la gestion de la mémoire            *
* Entrée      : Inclusion des UMB, recherche a priori dans les UMB ,  *
*               stratégie d'allocation de la mémoire                 *
* Sortie      : néant                                                *
* Info        : Les fonctions UMB sont disponibles qu'à partir de la *
*               version 5.0 de DOS                                   *
*********************************************************************}

procedure Demo( Avec_UMB    : boolean;
                UMB_dabord : boolean;
                Strategie  : integer );

var i         : integer;                          { Compteur d'itérations }
    Touche    : char;                                  { Touche frappée }
    Marqueur  : char;     { Marqueur (A-Z) souhaité pour la réservation }
    Essai     : word;                          { Taille de la réservation }
    Taille    : word;
    sdummy    : string;

begin
  {-- Initialise les champs adresse et taille ------------------------}

  for i := 0 to NBREBLOC - 1 do              { Pas encore de bloc alloué }
    with BlocChamp[ i ] do
      begin
        Adresse := 0;                      { Segment du bloc de mémoire }
        Taille := 0;                                  { Taille du bloc }
      end;

  gotoxy( 1, 7 );
  AfficheResultat( TRUE );                          { Affiche le tableau }

  {-- Boucle de démonstration ----------------------------------------}

  repeat
                        {-- Fixe la stratégie souhaitée --------------}

    if ( Avec_UMB ) then             { Exploiter la mémoire supérieure ? }
      FixeUMB( UMB_OUI )
    else
      FixeUMB( UMB_NON );

    if ( UMB_dabord ) then
      FixeStrategie( Strategie or DABORD_UMB )
    else
      FixeStrategie( Strategie );

    {-- Affiche la stratégie actuelle ------------------------------}

    gotoxy( 1, 3 );
    writeln( ' [F8] Gestion mémoire          : ', SText[ Strategie ]);
```

```
  writeln( ' [F9] Recherche dans l''UMB      : ', OuiNon[UMB_dabord]);
  writeln( '[F10] Utilisation des blocs UMB : ', OuiNon[Avec_UMB ]);
  writeln( '-------------------------------------------',
           '---------------------------' );

  {-- Saisie et traitement  ------------------------------------}

  repeat until keypressed;                  { Attend une frappe de touche }
  Touche := readkey;                                 { Lit la touche }
  if( ( Touche = #0 ) and ( keypressed ) ) then      { T. de fonction }
   Touche := readkey;                                { Lit le 2me code }

  case Touche of
    F1 :                                    { Alloue le bloc de mémoire }
      begin
        i := -1;                            { pas encore de bloc valide }
        repeat
          gotoxy( 1, 23 );
          write( 'Quel bloc faut-il réserver [ A-Z ]   : ' );
          readln( Marqueur );
          Marqueur := upcase( Marqueur );
          if( Marqueur >= 'A' ) and ( Marqueur <= 'Z' ) then
            if( BlocChamp[ ord( Marqueur ) - 65 ].Adresse = 0 ) then
              i := ord( Marqueur ) - 65;
        until( i <> - 1 );
        write( 'Combien de Ko faut-il réserver       : ' );
        readln( Essai );

        Essai := Essai * 64 - 1;          { Conversion en paragraphes }
        DOS_GetMem( Essai, BlocChamp[ i ].Adresse,
                    BlocChamp[ i ].Taille );
        if( BlocChamp[ i ].Taille <> Essai ) then          { Erreur ? }
          begin
            str( ( BlocChamp[ i ].Taille + 1 ) div 64, sdummy );
            write( 'Il ne reste plus que ' + sdummy + 'Ko ' );
            repeat
            until keypressed;

            while keypressed do
              Touche := readkey;
            Touche := #0;
          end;
        gotoxy( 1, 23 );
        writeln( '                                                ' );
        writeln( '                                                ' );
        write  ( '                                                ' );
        gotoxy( 1, 7 );
        AfficheResultat( FALSE );                { Affiche le tableau }
      end;

    F2 :                                                { Libère le bloc}
      begin
        i := -1;                            { Pas encore de bloc valable }
        repeat
          gotoxy( 1, 23 );
          write( 'Quel bloc faut-il libérer [ A-Z ]    : ' );
          readln( Marqueur );
          Marqueur := upcase( Marqueur );
          if( Marqueur >= 'A' ) and ( Marqueur <= 'Z' ) then
            if( BlocChamp[ ord( Marqueur ) - 65 ].Adresse <> 0 ) then
              i := ord( Marqueur ) - 65;
        until ( i <> - 1 );
        DOS_FreeMem( BlocChamp[ i ].Adresse );
        BlocChamp[ i ].Adresse := 0;
        BlocChamp[ i ].Taille := 0;
        gotoxy( 1, 23 );
        writeln( '                                                ' );
        gotoxy( 1, 7 );
        AfficheResultat( FALSE );                { Affiche le tableau }
      end;

    F3 :                                  { Modifie la taille d'un bloc }
      begin
        i := -1;                            { Pas encore de bloc valide }
        repeat
          gotoxy( 1, 23 );
          write( 'De quel bloc faut-il changer la taille [ A-Z ]: ' );
          readln( Marqueur );
          Marqueur := upcase( Marqueur );
          if ( Marqueur >= 'A' ) and ( Marqueur <= 'Z' ) then
            if ( BlocChamp[ ord( Marqueur ) - 65 ].Adresse <> 0 ) then
              i := ord( Marqueur ) - 65;
        until ( i <> - 1 );
        write( 'Combien de Ko faut-il réserver :     ' );
        readln( Essai );
        Essai := Essai * 64 - 1;          { Conversion en paragraphes }
        DOS_ChangeMem( Essai, BlocChamp[ i ].Adresse, Taille );
        if( Taille <> Essai ) then                         { Erreur ? }
          begin
            str( ( Taille + 1 ) div 64, sdummy );
            write( 'Il ne reste plus que ' + sdummy + 'Ko ' );
            repeat until keypressed;
            while keypressed do
              Touche := readkey;
            Touche := #0;
          end
        else
          BlocChamp[ i ].Taille := Taille;  { Fixe une nvelle taille }
        gotoxy( 1, 23 );
        writeln( '                                                ' );
        writeln( '                                                ' );
        write  ( '                                                ' );
        gotoxy( 1, 7 );
        AfficheResultat( FALSE );                { Affiche le tableau }
      end;

    F8 :
      Strategie := ( Strategie + 1 ) mod 3;   { Change de stratégie }

    F9 :                                    { Commutation : UMB d'abord }
      UMB_dabord := not UMB_dabord;

    F10:                                 { Commutation : Inclure les UMB }
      avec_UMB := not avec_UMB;
  end;
until( Touche = ESC );
end;

{*****************************************************************}
{***                    Programme principal                    ***}
{*****************************************************************}

var StartStrategie : integer;       { Stratégie d'allocation au départ }
    StartUMB       : integer;           { Etat des blocs UMB au départ }
    ActUMB_oui     : boolean;       { Utiliser les blocs UMB (oui/non) }
    ActUMB_dabord  : boolean;          { Mémoire supérieure prioritaire }
    ActStrategie   : integer;         { Stratégie d'allocation actuelle }

begin
      {-- Sauvegarde les valeurs courantes ---------------------------}

  StartStrategie := LitStrategie;              { Stratégie d'allocation }
  StartUMB := LitUMB;        { Prise en compte de la mémoire supérieure }
  AllogueMemoire;                          { Crée l'environnement du test }
  FixeStrategie( Startstrategie );      { Restaure l'ancienne stratégie }
  FixeUMB( StartUMB );

  if ( SegConv = 0 ) then           { Mémoire conventionnelle allouée ? }
    begin                                  { Non, on arrête avec erreur }
      clrscr;
      writeln('MEMDEMOP : Mémoire insuffisante !');
      exit;                                                { C'est fini }
    end;

      {-- Valeurs de départ pour la gestion de la mémoire ------------}

  ActUMB_oui := ( StartUMB = UMB_OUI );
  ActUMB_dabord := ( ( StartStrategie and DABORD_UMB ) = DABORD_UMB );
  ActStrategie := StartStrategie and ( $FF xor DABORD_UMB );

      {-- Démonstration de la gestion de la mémoire ------------------}

  clrscr;
  writeln( 'Programme de gestion mémoire DOS                 ',
           '(C) 1991, 92 by Michael TISCHER' );
  writeln( '-------------------------------------------------',
           '-----------------------' );
  Demo( ACTUMB_oui, ACTUMB_dabord, ActStrategie );

      {-- Restaure les anciennes valeurs de DOS----------------------}

  LibereMemoire:                                { Libère la mémoire allouée }
  FixeStrategie( StartStrategie );
  FixeUMB( StartUMB );
  clrscr;
end.
```

Listing : MEMDEMOC.C

```
/*****************************************************************
*                        M E M D E M O C . C                     *
******************************************************************
*   Fonction       : Montre comment DOS gère la mémoire          *
******************************************************************
*   Modèle mémoire : SMALL                                       *
******************************************************************
*   Auteur         : Michael TISCHER                             *
*   Développé le   : 08.10.1991                                  *
*   Dernière MAJ   : 20.03.1992                                  *
*****************************************************************/

/*-- Fichiers d'inclusion ---------------------------------------*/

#include <dos.h>
#include <conio.h>
#include <stdio.h>
#include <stdarg.h>
#include <string.h>
#include <ctype.h>
#include <stdlib.h>

/*-- Constantes -------------------------------------------------*/

#define TRUE  ( 0 == 0 )
#define FALSE ( 0 == 1 )

/*-- Numéros des fonctions de l'interruption 0x21 ---------------*/

#define GET_MEM      0x48            /* Réserve de la mémoire vive */
#define FREE_MEM     0x49            /* Libère de la mémoire vive */
#define CHANGE_MEM   0x4A  /* Change la taille d'une zone de mémoire */
#define GET_STRATEGY 0x5800          /* Lit la stratégie d'allocation */
#define SET_STRATEGY 0x5801         /* Fixe la stratégie d'allocation */
#define GET_UMB      0x5802  /* Lit l'état d'inclusion des blocs UMB */
#define SET_UMB      0x5803         /* Fixe l'inclusion des blocs UMB */

/*-- Stratégies de recherche pour FixeStrategie -----------------*/

#define CHERCHE_ENBAS    0x00           /* 1er bloc de mémoire libre */
#define CHERCHE_MEILLEUR 0x01           /* Meilleur bloc de mémoire */
#define CHERCHE_OBEN     0x02      /* Dernier bloc de mémoire libéré */
#define DABORD_UMB       0x80       /* Chercher dans la zone des UMB */
                                    /* (à utiliser avec CHERCHE_ ): */

/*-- Constantes pour FixeUMB ------------------------------------*/

#define UMB_NON      0x00               /* Ignore la mémoire supérieure */
#define UMB_OUI      0x01                     /* Alloue des blocs UMB */

/*-- Constantes pour Demo ---------------------------------------*/

#define TEST_TA        (10240-1)       /* 10239 paragraphes pour test*/
#define TEST_TA_UMB    (2560-1)     /* 2559 paragraphes UMB pour test */
#define TEST_TA_KB     160                /* Envir. de test = 160 Ko */
#define TEST_TA_UMB_KB 40               /* Envir. de test UMB = 40 Ko */
#define NBREBLOC       26    /* Nbr d'adr. pour affichage du résultat */

/*-- Codes de touches pour saisie utilisateur -------------------*/

#define ESC  27                                 /* Interruption par ESC */
#define F1   59                                 /* Touche de fonction F1 */
#define F2   60                                 /* Touche de fonction F2 */
#define F3   61                                 /* Touche de fonction F3 */
#define F8   66                                 /* Touche de fonction F8 */
#define F9   67                                 /* Touche de fonction F9 */
#define F10  68                                /* Touche de fonction F10 */

/*-- Déclarations de types --------------------------------------*/

typedef struct
        {
          unsigned int Adresse,                          /* Segment */
                       Taille;                  /* Taille en mémoire */
        } BlocTyp;

typedef unsigned char BYTE;

/*-- Macros -----------------------------------------------------*/

#ifdef MK_FP                          /* Macro MK_FP déjà définie ? */
  #undef MK_FP                          /* Oui, alors on l'efface */
#endif

#define MK_FP(s,o) ((void far *)(((unsigned long) (s)<<16)|(o)))

/*-- Constantes typées ------------------------------------------*/

char *OuiNon[]  =
          { "Non ", "Oui " };
char *SText[]   =
          {
            "Utilise le premier bloc de mémoire libre ",
            "Utilise le meilleur bloc de mémoire libre",
            "Utilise le dernier bloc de mémoire libre "};
BYTE ChampCouleur[] =
          { 0x07, 0x70 };
char *TexteTouche[] =
          {
            " [F1] Allouer de la mémoire",
            " [F2] Libérer de la mémoire",
            " [F3] Modifier la taille",
            " [ESC] Fin du programme"};

/*-- Variables globales -----------------------------------------*/

union REGS   regs;          /* Registres pour gérer les interruptions */
struct SREGS sregs;        /* Registres de segment pour int. étendues */
unsigned int ChampAdresse[ 1000 ];              /* Champs de mémoire */
unsigned int NbAdresse;                         /* Nombre d'adresses */
unsigned int SegConv;              /* Adr. bloc test en mémoire conv. */
unsigned int UMBSeg;               /* Adr. bloc test en mémoire sup. */
BlocTyp      BlocChamp[ NBREBLOC ];                      /* Mémoire */

/*-- Routines d'écran pour Microsoft C --------------------------*/

#ifndef __TURBOC__                              /* Microsoft C? */

  #define textcolor( Couleur )
  #define textbackground( Couleur )

  /*****************************************************************
  * Gotoxy       : Positionne le curseur                          *
  * Entrées      : Coordonnées du curseur                         *
  * Sortie       : néant                                          *
  *****************************************************************/

  void gotoxy( int x, int y )
  {
    regs.h.ah = 0x02;          /* Numéro de la fonction d'interruption*/
    regs.h.bh = 0;                                      /* Couleur */
    regs.h.dh = y - 1;
    regs.h.dl = x - 1;
    int86( 0x10, &regs, &regs );          /* Déclenche l'interruption */
  }

  /*****************************************************************
  * clrscr       : Efface l'écran                                 *
  * Entrées      : cf infra                                       *
  * Sortie       : néant                                          *
  *****************************************************************/

  void clrscr( void )
  {
    regs.h.ah = 0x07;          /* Numéro de la fonction d'interruption*/
    regs.h.al = 0x00;
    regs.h.ch = 0;
    regs.h.cl = 0;
    regs.h.dh = 24;
    regs.h.dl = 79;
    int86( 0x10, &regs, &regs );          /* Déclenche l'interruption */
    gotoxy( 1, 1 );                        /* Positionne le curseur */
  }

#endif

/*****************************************************************
* PRINT : Comme RINTF, mais écrit directement la chaîne dans la  *
*         mémoire d'écran                                        *
* Entrées : COLONNE = Colonne d'affichage                        *
*           LIGNE   = Ligne d'affichage                          *
*           COULEUR = Couleur d'affichage                        *
*           STRING  = Pointeur sur la chaîne PRINTF              *
*           ...     = autres arguments                           *
* Sortie : néant                                                 *
*****************************************************************/

void Print( int Colonne, int Ligne, BYTE Couleur, char * String, ...)
{
  struct vr {                       /* Une position à l'écran = 2 octets */
         BYTE caractere,                                /* Code ASCII */
              attribut;                                /* Attribu associé */
       } far *lptr;          /* Pointeur courant sur la mémoire d'écran */
  va_list     parameter;     /* Liste de paramètres pour macros VA_... */
  char        affichage[255],        /* Buffer pour chaîne de formatage */
              *aptr = affichage;       /* Pour parcourir ladite chaîne */
```

```
static unsigned int vioseg = 0;
union REGS   Register;     /* Registres pour gérer les interruptions */

if( vioseg == 0 )                                  /* Premier appel  ?*/
{                  /* Oui, cherche le segment de la mémoire d'écran */
  Register.h.ah = 0x0F;
  int86(0x10, &Register, &Register);
  vioseg = ( Register.h.al == 7 ? 0xb000 : 0xb800 );
}

va_start( parameter, String );             /* Convertit les paramètres */
vsprintf( affichage, String, parameter );                /* Formate */
lptr = (struct vr far *)
        MK_FP( vioseg, ( (Ligne-1) * 80 + (Colonne-1) ) << 1 );

for( ; *aptr ; )                                   /* Parcourt la chaîne */
{
  lptr->caractere = *aptr++;          /* Caractère en mémoire d'écran */
  lptr++->attribut = Couleur;         /* Fixe l'attribut du caractère */
  ++Colonne;
}
gotoxy( Colonne, Ligne );                          /* Déplace le curseur */
}

/*****************************************************************
* DOS_GetMem : Réserve de la mémoire                             *
* Entrée     : Taille mémoire souhaitée en paragraphes           *
* Sortie     : Segment du bloc de mémoire alloué,                *
*              Nombre de paragraphes alloués ou nombre maximum   *
*              de paragraphes disponibles                        *
*****************************************************************/

void DOS_GetMem( unsigned int Ta,
                 unsigned int *Adr,
                 unsigned int *Res )
{
regs.h.ah = GET_MEM;                           /* Numéro de la fonction */
regs.x.bx = Ta;                    /* Nombre de paragraphes à réserver */
intdos( &regs, &regs );                          /* Interruption de DOS */

if( !regs.x.cflag )                                  /* Appel réussi ? */
{
  *Adr = regs.x.ax;             /* Oui, restitue l'adresse et la taille */
  *Res = Ta;
}
else                                                    /* Non, erreur*/
{
  *Adr = 0;                                  /* Pas de mémoire réservée */
  *Res = regs.x.bx;                         /* Taille disponible maximale */
}
}

/*****************************************************************
* DOS_FreeMem: Libère de la mémoire préalablement réservée       *
* Entrée     : Segment de la mémoire                             *
* Sortie     : néant                                             *
*****************************************************************/

void DOS_FreeMem( unsigned int Adr )
{
regs.h.ah = FREE_MEM;                          /* Numéro de la fonction */
sregs.es = Adr;                                    /* Segment du bloc */
intdosx( &regs, &regs, &sregs );                 /* Interruption de DOS */
}

/*****************************************************************
* DOS_ChangeMem : Modifie la taille d'un bloc réservé            *
* Entrées       : Ancien segment et nouvelle taille              *
* Sortie        : Segment du nouveau bloc alloué,                *
*                 nombre de paragraphes alloués ou disponibles   *
*****************************************************************/

void DOS_ChangeMem( unsigned int Ta,
                    unsigned int *Adr,
                    unsigned int *Res )
{
regs.h.ah = CHANGE_MEM;                        /* Numéro de la fonction */
regs.x.bx = Ta;                    /* Nombre de paragraphes à réserver */
sregs.es = *Adr;                          /* Segment du bloc à modifier */
intdosx( &regs, &regs, &sregs );                 /* Interruption de DOS */
if( !regs.x.cflag )                                  /* Appel réussi? */
  *Res = Ta;                          /* Oui, indique la nouvelle taille */
else                                                    /* Non, erreur */
  *Res = regs.x.bx;                          /* Mémoire maximale disponible */
}

/*****************************************************************
* LitStrategie : Lit la stratégie d'allocation en vigueur        *
* Entrée       : néant                                           *
* Sortie       : Type de stratégie                               *
*****************************************************************/

int LitStrategie( void )
{
regs.x.ax = GET_STRATEGY;                  /* Fixe le numéro de la fonction */
intdos( &regs, &regs );
return regs.x.ax;                          /* Retourne le type de stratégie */
}

/*****************************************************************
* LitUMB       : Lit l'état d'inclusion des blocs UMB            *
* Entrée       : néant                                           *
* Sortie       : Indique sir les blocs UMB sont pris en compte   *
* Info         : à partir de la version 5.0 de DOS uniquement    *
*****************************************************************/

int LitUMB( void )
{
regs.x.ax = GET_UMB;                       /* Fixe le numéro de la fonction */
intdos( &regs, &regs );
return regs.h.al;                                  /* Indique l'état */
}

/*****************************************************************
* FixeStrategie: Fixe la stratégie d'allocation de la mémoire    *
* Entrée       : Nouveau type de stratégie souhaitée             *
* Sortie       : néant                                           *
*****************************************************************/

void FixeStrategie( unsigned int Strategie )
{
regs.x.ax = SET_STRATEGY;                  /* Fixe le numéro de la fonction*/
regs.x.bx = Strategie;
intdos( &regs, &regs );
}

/*****************************************************************
* FixeUMB      : Fixe l'état d'inclusion des blocs de mémoire UMB *
* Entrée       : Nouvel état d'inclusion souhaité                *
* Sortie       : néant                                           *
* Info         : Disponible à partir de la version 5.0 de DOS    *
*****************************************************************/

void FixeUMB( unsigned int UMB )
{
regs.x.ax = SET_UMB;                       /* Fixe le numéro de la fonction */
regs.x.bx = UMB;
intdos( &regs, &regs );
}

/*****************************************************************
* AllOueMemoire  : Crée l'environnement du test                  *
* Entrée         : néant                                         *
* Sortie         : néant                                         *
*****************************************************************/

void AllOueMemoire( void )
{
unsigned int SegAdr;                       /* Segment de la mémoire allouée */
unsigned int Essai;                         /* Taille de mémoire souhaitée */
unsigned int Taille;                       /* Taille de la mémoire allouée */

/*-- 1º Alloue les blocs de test ---------------------------------*/

FixeUMB( UMB_NON );
DOS_GetMem( TEST_TA, &SegConv, &Taille );          /* Cherche le bloc*/
if( SegConv == 0 )                                        /* Erreur? */
  return;                                  /* Oui, retourne à l'appelant*/

FixeUMB( UMB_OUI );
FixeStrategie( CHERCHE_OBEN | DABORD_UMB );
DOS_GetMem( TEST_TA_UMB, &UMBSeg, &Taille );
if( UMBSeg != 0  &&  UMBSeg < 0xA000 )         /* Pas de blocs UMB ? */
{
  DOS_FreeMem( UMBSeg );                          /* Libère la mémoire */
  UMBSeg = 0;                                    /* Absence de blocs UMB */
}

/*-- 2º Alloue la mémoire restante par blocs de 1 Ko ---------------*/

Essai = 63;                    /* Essaie d'abord d'allouer 14 paragraphes */
NbAdresse = 0;
do
{
  DOS_GetMem( Essai, &SegAdr, &Taille );     /* Réclame de la mémoire */
  if( SegAdr != 0 )                             /* Mémoire accordée ? */
    ChampAdresse[ NbAdresse++ ] = SegAdr; /* Oui, mémorise l'adresse */
}
while( SegAdr != 0 );                                 /* Tout est alloué */

/*-- 3º Libère à nouveau les blocs de test -----------------------*/

if( SegConv > 0 )
  DOS_FreeMem( SegConv-- );                        /* MCB également libéré */
if( UMBSeg > 0 )
  DOS_FreeMem( UMBSeg-- );                         /* MCB également libéré */
```

```
!)
!
!/*********************************************************************
!* LibereMemoire   : Libère la mémoire allouée par ALlOueMemoire      *
!* Entrée          : néant                                            *
!* Sortie          : néant                                            *
!* Variable globale: ChampAdresse/R                                   *
!*********************************************************************/
!
!void LibereMemoire( void )
!
!{
! unsigned int i;                                          /* Compteur */
!
! if( NbAdresse > 0 )              /* Libère un à un les blocs de 1 Ko */
!   for( i = 0; i < NbAdresse; i++ )
!     DOS_FreeMem( ChampAdresse[ i ] );
!}
!
!/*********************************************************************
!* AfficheResultat : Affiche l'occupation de la mémoire               *
!* Entrées         : AVECCADRE = TRUE, si le cadre doit aussi être -  *
!*                   affiché                                          *
!* Sortie          : néant                                            *
!*********************************************************************/
!
!void AfficheResultat( BYTE AvecCadre )
!{
!
! char          SChamp[ TEST_TA_KB ];
! char          SChamp_UMB[ TEST_TA_UMB_KB ];
! unsigned int  i, j;                                  /* Compteurs */
! unsigned int  Position;                     /* Variable auxiliaire */
! char          DerCara;      /* Mémorise le dernier caractère affiché */
! int           CoulCour;              /* Couleur d'affichage courante */
!
! memset( SChamp, 32, TEST_TA_KB );        /* Initialise champs d'aff. */
! memset( SChamp_UMB, 32, TEST_TA_UMB_KB );
!
! /*-- Remplit le tableau de la mémoire ------------------------------*/
!
! for( i = 0; i < NBREBLOC; i++ )
! {
!   if( BlocChamp[ i ].Adresse > 0xA000 )                    /* UMB? */
!   {
!     Position = ( BlocChamp[ i ].Adresse - UMBSeg ) / 64;
!     for( j = 0; j <= BlocChamp[ i ].Taille / 64; j++ )
!       SChamp_UMB[ Position + j ] = i + 65;
!   }
!   else if ( BlocChamp[ i ].Adresse > 0 )
!   {
!     Position = ( BlocChamp[ i ].Adresse - SegConv ) / 64;
!     for( j = 0; j <= BlocChamp[ i ].Taille / 64; j++ )
!       SChamp[ Position + j ] = i + 65;
!   }
! }
!
! /*-- Trace le cadre du tableau -------------------------------------*/
!
! if ( AvecCadre )
! {
!   Print( 1, 7, 0x07, "Mémoire conventionnelle :" );
!   Print( 1, 8, 0x07,
!          "                  1       2       3       4" );
!   Print( 1, 9, 0x07,
!          "          1       0       0       0       0" );
!   Print( 1, 10, 0x07,
!          "" );
!   for ( i = 0; i < 4; i++ )
!    Print( 1, 11 + i, 0x07, " %3i %s%s", i * 40,
!           "Ko                                    ",
!           TexteTouche[ i ] );
!   Print( 1, 15, 0x07,
!          "" );
!   if ( UMBSeg > 0 )
!   {
!    Print( 1, 17, 0x07, "UMB :" );
!    Print( 1, 18, 0x07,
!           "                  1       2       3       4" );
!    Print( 1, 19, 0x07,
!           "          1       0       0       0       0" );
!    Print( 1, 20, 0x07,
!           "" );
!    Print( 1, 21, 0x07,
!           "   0 KB                                     " );
!    Print( 1, 22, 0x07,
!           "" );
!   }
!   else
!    Print( 1, 17, 0x07, "Pas d'UMB disponible");
! }
!
!  /*-- Affiche le tableau de la mémoire conventionnelle -------------*/
!
! DerCara = 0;                                /* Dernier caractère affiché */
! CoulCour = 1;                               /* Dernière couleur affichée */
! for( i = 0; i < 4; i++ )
!  for( j = 0; j < 40; j++ )
!  {
!    if( DerCara != SChamp[ i * 40 + j ] ) /* Changement de couleur ? */
!    {
!      CoulCour = ( CoulCour + 1 ) % 2;    /* Nouveau code de couleur */
!      DerCara = SChamp[ i * 40 + j ];     /* Caractère de comparaison */
!    }
!    Print( j + 11, i + 11, ChampCouleur[ CoulCour ],
!           "%c", SChamp[ i * 40 + j ] );
!  }
!
! /*-- Affiche le tableau des UMB ------------------------------------*/
!
! if( UMBSeg > 0 )
! {
!   for( j = 0; j < 40; j++ )
!   {
!     if( DerCara != SChamp_UMB[ j ] )        /* Changement de couleur */
!     {
!       CoulCour = ( CoulCour + 1 ) % 2;    /* Nouveau code de couleur */
!       DerCara = SChamp_UMB[ j ];          /* Caractère de comparaison */
!     }
!     Print( j + 11, 21, ChampCouleur[ CoulCour ],
!            "%c", SChamp_UMB[ j ] );
!   }
! }
!}
!
!/*********************************************************************
!* Demo     : Démonstration de la gestion de la mémoire               *
!* Entrées  : Inclusion des UMB, recherche prioritaire dans les UMB   *
!*            Stratégie d'allocation de la mémoire                    *
!* Sortie   : néant                                                   *
!*********************************************************************/
!
!void Demo( int Avec_UMB,
!           int UMB_dabord,
!           int Strategie )
!{
! int           i;                                        /* Compteur */
! int           Touche;                          /* Touche frappée */
! char          Marqueur[ 5 ];          /* Marqueur (A-Z) de réservation */
! unsigned int Essai;                                /* Taille réservée */
! unsigned int Taille;
!
!  /*-- Initialise les champs adresse et taille ----------------------*/
!
! for ( i = 0; i < NBREBLOC; i++ )                /* Toutes les adresses */
! {
!   BlocChamp[ i ].Adresse = 0;                     /* Segment du bloc */
!   BlocChamp[ i ].Taille = 0;                       /* Taille du bloc */
! }
!
! AfficheResultat( TRUE );                          /* Affiche le tableau */
!
!  /*-- Boucle de démonstration ---------------------------------------*/
!
! do
! {
!  /*-- Fixe la stratégie souhaitée ----------------------------------*/
!  if( Avec_UMB )                                  /* Utilise les UMB ? */
!    FixeUMB( UMB_OUI );
!  else
!    FixeUMB( UMB_NON );
!
!  if( UMB_dabord )
!    FixeStrategie( Strategie | DABORD_UMB );
!  else
!    FixeStrategie( Strategie );
!
!  /*-- Affiche la stratégie actuelle --------------------------------*/
!
!  Print( 1, 3, 0x07,
!    "  [F8] Stratégie de gestion mémoire : %s", STexte[ Strategie ]  );
!  Print( 1, 4, 0x07,
!    "  [F9] Recherche dans les blocs UMB : %s", OuiNon[ UMB_dabord ]);
!  Print( 1, 5, 0x07,
!    "  [F10] Exploitation des blocs UMB  : %s", OuiNon[ Avec_UMB ]  );
!  Print( 1, 6, 0x07, "-----------------------------------" );
!  Print( 40, 6, 0x07,"-----------------------------------" );
!
!  /*-- Saisie et traitement -----------------------------------------*/
!
!  while( !kbhit() );                  /* Attend une frappe de touche */
!  Touche = getch();                                  /* Lit la touche */
!  if( ( Touche == 0 ) && ( kbhit() ) )          /* Touche de fonction */
!    Touche = getch();                          /* Cherche le 2me code */
!
!  switch(  Touche )
```

```
{
  case F1 :                              /* Alloue le bloc de mémoire */
    i = -1;                              /* Pas encore de bloc valide */
    do                                   /* Saisie jusqu'à bloc valide */
    {
     Print( 1, 23, 0x07, "Quel bloc faut-il réserver [ A-Z ]: " );
     scanf( "%s", Marqueur );
     Marqueur[0] = toupper( Marqueur[ 0 ] );
     if(( Marqueur[0] >= 'A' ) && ( Marqueur[0] <= 'Z' ))
       if( BlocChamp[ (int) Marqueur[0] - 65 ].Adresse == 0 )
         i = (int) Marqueur[0] - 65;
    }
    while ( i == -1 );
    Print( 1, 24, 0x07,
      "Combien de Ko faut-il réserver : " );
    scanf( "%i", &Essai );
    Essai = Essai * 64 - 1;              /* Conversion en paragraphes */
    DOS_GetMem( Essai, &BlocChamp[ i ].Adresse,
                       &BlocChamp[ i ].Taille );
    if( BlocChamp[ i ].Taille != Essai )                 /* Erreur ? */
    {
     Print( 1, 25, 0x07, "Il ne reste plus que %Ad Ko !",
                         ( BlocChamp[ i ].Taille + 1 ) / 64 );
     while( !kbhit() );
     while( kbhit() )
       Touche = getch();
     Touche = 0;
    }
    Print(1,23, 0x07, "                                                ");
    Print(1,24, 0x07, "                                                ");
    Print(1,25, 0x07, "                                                ");
    AfficheResultat( FALSE );                  /* Affiche le tableau */
    break;

  case F2 :                                /* Libère le bloc alloué */
    i = -1;                                /* Pas encore de bloc valide */
    do                                     /* Saisie jusqu'à bloc valide */
    {
      Print( 1, 23, 0x07, "Quel bloc faut-il libérer [ A-Z ] : " );
      scanf( "%s", Marqueur );
      Marqueur[0] = toupper( Marqueur[ 0 ] );
      if(( Marqueur[0] >= 'A' ) && ( Marqueur[0] <= 'Z' ))
        if( BlocChamp[ (int) Marqueur[0] - 65 ].Adresse != 0 )
          i = (int) Marqueur[0] - 65;
    }
    while( i == -1 );
    DOS_FreeMem( BlocChamp[ i ].Adresse );
    BlocChamp[ i ].Adresse = 0;
    BlocChamp[ i ].Taille = 0;
    Print(1. 23, 0x07, "                                               ");
    AfficheResultat( FALSE );                  /* Affiche le tableau */
    break;

  case F3 :                              /* Modifie la taille d'un bloc */
    i = -1;                                /* Pas encore de bloc valide */
    do                                     /* Saisie jusqu'à bloc valide */
    {
      Print( 1. 23, 0x07, "Quel bloc faut-il libérer [ A-Z ] : " );
      scanf( "%s", Marqueur );
      Marqueur[0] = toupper( Marqueur[ 0 ] );
      if(( Marqueur[0] >= 'A' ) && ( Marqueur[0] <= 'Z' ))
        if( BlocChamp[ (int) Marqueur[0] - 65 ].Adresse != 0 )
          i = (int) Marqueur[0] - 65;
    }
    while ( i == -1 );
    Print( 1, 24, 0x07, "Combien de Ko faut-il réserver : " );
    scanf( "%i", &Essai );
    Essai = Essai * 64 - 1;              /* Conversion en paragraphes */
    DOS_ChangeMem( Essai, &BlocChamp[ i ].Adresse, &Taille );
    if( Taille != Essai )                                /* Erreur ?*/
    {
      Print( 1, 23, 0x07, "Il ne reste plus que %4i Ko !",
                          ( Taille + 1 ) / 64 );
      while( !kbhit() );
      while( kbhit() )
        Touche = getch();
      Touche = 0;
    }
    else
      BlocChamp[ i ].Taille = Taille;            /* Nouvelle taille */
    Print(1. 23, 0x07,"                                              ");
    Print(1. 24, 0x07,"                                              ");
    Print(1. 25, 0x07,"                                              " );
    AfficheResultat( FALSE );                  /* Affiche le tableau */
    break;

  case F8 :                                    /* Change de stratégie */
    Strategie = ( Strategie + 1 ) % 3;
    break;

  case F9 :                        /* Commutation : Recherche prioritaire */
                                   /*  dans les blocs UMB */
    UMB_dabord = ! UMB_dabord;
    break;

  case F10:                  /* Commutation : Inclusion des blocs UMB */
    Avec_UMB = ! Avec_UMB;
    break;
  }
}
while( Touche != ESC );
}

/*****************************************************************
**                    Programme principal                       **
*****************************************************************/

void main( void )
{
 int StartStrategie;                 /* Stratégie d'allocation au départ */
 int StartUMB;          /* Décision d'inclusion des blics UMB au départ */
 int Act_UMB_oui;                /* Exploitation des blocs UMB (Oui/Non) */
 int Act_UMB_dabord;                /* Recherche prioritaire des UMB */
 int Act_Strategie;                 /* Stratégie d'allocation actuelle */

 /*-- Dessine l'écran  ---------------------------------------------*/

 clrscr();
 Print( 1, 1, 0x07,
        "Démonstration de gestion mémoire sous DOS " );
 Print( 51, 1, 0x07,
          " (C) 1991, 92 by Michael TISCHER" );
 Print( 1, 2, 0x07, "ÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄ" );
 Print( 40, 2, 0x07, "ÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄ" );
 Print( 25, 5, 0x07, "Initialisation");

     /*-- Sauvegarde les valeurs courantes ----------------------------*/

 StartStrategie = LitStrategie();            /* Strategie d'allocation */
 StartUMB = LitUMB();                        /* Prise en compte des UMB */
 AllogueMemoire();                          /* Crée lenvironnement de test*/
 FixeStrategie( StartStrategie );     /* Restaure l'ancienne stratégie */
 FixeUMB( StartUMB );

 if ( SegConv == 0 )                 /* Mémoire conventionnelle allouée ? */
 {                                             /* Non, termine avec erreur */
  clrscr();
  printf( "MEMDEMOC : Mémoire insuffisante \n" );
  exit(1);
 }

                    /*-- Valeurs de départ ---------------------------*/

 Act_UMB_oui = ( StartUMB == UMB_OUI );
 Act_UMB_dabord = ( ( StartStrategie & DABORD_UMB ) == DABORD_UMB );
 Act_Strategie = ( StartStrategie & ( 0xFF ^ DABORD_UMB ) );

 /*-- Démonstration de la gestion de la mémoire ----------------------*/

 clrscr();
 Print( 1, 1, 0x07,
        "Démonstration de gestion mémoire sous DOS");
 Print( 51, 1, 0x07,
          "(C) 1991, 92 Michael Tischer" );
 Print( 1, 2, 0x07, "ÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄ" );
 Print( 40, 2, 0x07, "ÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄ" );
 Demo( Act_UMB_oui, Act_UMB_dabord, Act_Strategie );

 /*-- Restaure les anciennes valeurs ---------------------------------*/

 LibereMemoire();                          /* Libère l'espace réservé */
 FixeStrategie( StartStrategie );
 FixeUMB( StartUMB );
 clrscr();
}
```

22.4. Les coulisses de la gestion mémoire

Si vous jetez un coup d'oeil attentif sur la fin des listings précédents, vous y verrez apparaître l'acronyme MCB qui veut dire "Memory Control Bloc" (Bloc de contrôle de la mémoire). Ces MCB jouent un rôle central dans la gestion interne de DOS. Nous allons étudier ce point dans le présent chapitre, ce qui revient à pénétrer véritablement dans les coulisses de l'activité de DOS.

La gestion des zones de mémoire allouées par les MCB

Pour gérer les zones de mémoire allouées par la fonction 48h de DOS, le système met un MCB à la tête de chaque bloc. Il occupe 16 octets, débute toujours à une adresse d'offset divisible par 16 et précède immédiatement la zone allouée. Les fonctions DOS travaillent toujours avec l'adresse de segment des zones allouées, mais il est facile de déterminer le segment du MCB en diminuant cette adresse de 1.

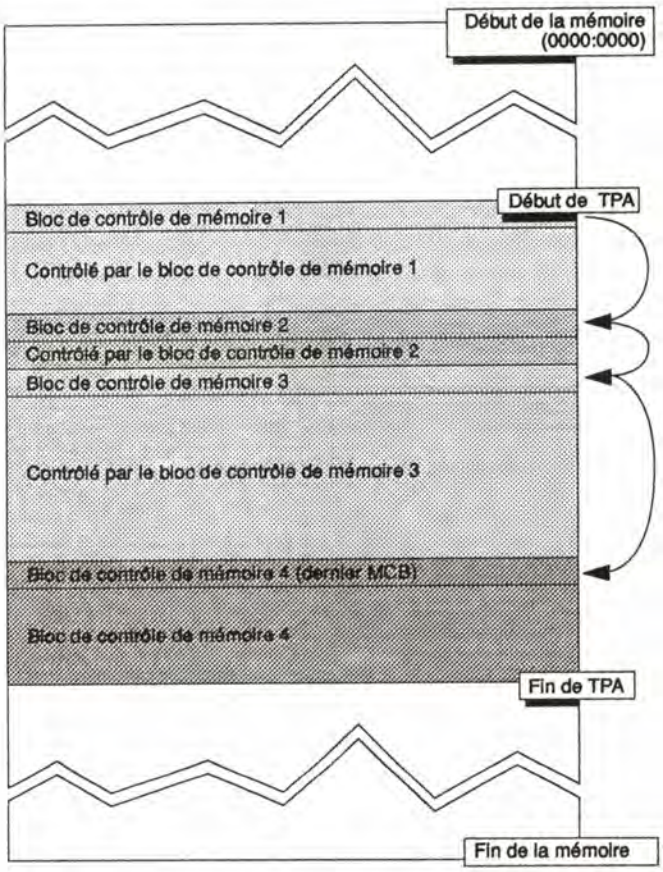
Organisation d'un MCB en mémoire		
Adresse	**Fonction**	**Type**
+00h	ID ("Z" = dernier MCB, "M" = à suivre)	1 BYTE
+01h	adresse de segment PSP	1 WORD
+03h	taille des paragraphes en mémoire	1 WORD
+05h	non utilisé	11 BYTE
+10h	espace mémoire allouée	x PARAG
Longueur : 16 + taille de l'espace mémoire allouée		

Comme le montre la figure ci-dessus, le MCB se compose de trois champs. Le premier d'entre eux contient l'une des initiales du dénommé Mark Zbikowski, un développeur de MS-DOS qui s'est amusé à immortaliser son nom. "M" indique que le présent MCB est suivi d'autres MCB, alors que "Z" signale qu'il s'agit du dernier MCB dans la mémoire.

Le deuxième champ est le segment du PSP du programme propriétaire. Ce champ n'est significatif que si la zone allouée fait partie de l'environnement d'un programme référencé par son PSP. Car avant de charger un programme la fonction EXEC alloue dans tous les cas une zone de mémoire séparée pour y stocker le bloc d'environnement du programme.

Si la zone de mémoire est un PSP, le deuxième champ d'un MCB pointera la plupart du temps sur la zone elle-même.

Le troisième champ du MCB est beaucoup plus intéressant. Il indique la taille en paragraphes de la zone de mémoire introduite. Comme le prochain MCB suit immédiatement la zone allouée, le nombre contenu dans ce champ est égal à la distance jusqu'au prochain MCB diminuée de 1. Chaque MCB pointe donc sur le suivant : il en résulte une liste chaînée qu'il suffit de parcourir pour détecter tous les MCB existants.



Gestion de la mémoire par les MCB

Entrée dans la liste des MCB

Pour pouvoir se déplacer dans la liste des MCB, il faut d'abord en trouver un point d'entrée. L'adresse du premier MCB en mémoire est tenue à jour par DOS dans une structure interne appelée DIB (DOS Information Bloc) ou bloc d'information DOS. Normalement ce DIB n'est pas accessibles aux programmes. Mais l'adresse de la structure peut être recherchée par le moyen de la fonction non documentée 52h qui renvoie un pointeur sur le DIB dans les registres appariés ES:BX.

Curieusement le pointeur ainsi obtenu ne référence pas le premier champ du DIB mais le second ! Mais le renseignement qui nous intéresse, à savoir l'adresse du premier MCB, se trouve justement dans le premier champ du DIB. Le pointeur se compose fort naturellement d'un segment et d'un offset, ce qui fait 4 octets. Il se trouve donc à l'adresse ES:[BX-4].

Cette indication est à exploiter avec beaucoup de précautions. On pourrait croire qu'il suffit de retrancher 4 du contenu du registre BX pour obtenir en ES:BX l'adresse effective de l'information recherchée. Mais tel n'est le cas que si l'offset en BX est supérieur ou égal à 4. Si l'offset est inférieur, la soustraction de 4 donne un nombre négatif, ce qui cause une erreur d'adressage. En effet il n'existe pas d'adresse négative !

Voici un exemple pour mettre en évidence le problème. Si le registre BX contient le nombre 0 comme offset du DIB, la soustraction de 4 donne 0FFFCh. Dans le cadre d'une opération arithmétique, ce nombre est interprété comme -4. Mais utilisé comme adresse, 0FFFCh est considéré comme un nombre non signé, représentant une mémoire tout au bout du segment du DIB. Il est clair qu'on ne trouvera pas l'information escomptée à cet endroit et qu'on ne pourra pas entrer ainsi dans la liste des MCB.

Programmes d'exemple

Les trois programmes suivants écrits en BASIC, Pascal et C suivent la liste chaînée des MCB et affichent leur contenu. Leur logique est suffisamment raffinée pour leur permettre de distinguer les zones qui mémorisent l'environnement d'un programme de celles qui contiennent un PSP ou d'autres informations.

La technique utilisée consiste à progresser à travers la mémoire de MCB en MCB pour analyser les zones rencontrées. Pour passer d'un MCB au suivant, il suffit de se servir du troisième champ d'un MCB qui permet de construire un pointeur sur le MCB suivant. Il en résulte une boucle qui est parcourue jusqu'au dernier MCB dont le champ d'identification contient le caractère "Z".

Chaque MCB ainsi exploré donne quelques informations d'état sur lui-même et sur la zone de mémoire qu'il gère. Ce sont plus précisément :

- ✓ le numéro du MCB
- ✓ son adresse en mémoire
- ✓ l'adresse de la zone de mémoire génére par le MCB
- ✓ le contenu du champ d'identification ("M" ou "Z")
- ✓ l'adresse du PSP associé (indépendamment de son existence effective)
- ✓ la taille de la zone de mémoire associée en paragraphes et octets

Affichage d'un bloc d'environnement

Une fois les informations recueillies, la zone de mémoire associée est examinée. Le programme teste d'abord s'il s'agit d'un bloc d'environnement. Tel est le cas si au début de la zone de mémoire se trouve la chaîne COMSPEC= qui introduit tout bloc de ce type.

Si la chaîne en question est découvertes le programme se met à lire les différentes chaînes d'environnement. La tâche est facile car chacune de ces chaînes se présente en format ASCII et se termine par un caractère nul. Si à la suite d'une chaîne on retrouve un autre caractère nul, c'est qu'on a atteint la fin de la chaîne d'environnement.

Jusqu'à la version 3.0 de DOS, c'étaient là les seules informations à tirer du bloc d'environnement. Mais depuis la version 3.1 la dernière chaîne du bloc est suivie d'une information très intéressante, à savoir le nom du programme auquel appartient le bloc. Ce nom se présente avec le chemin d'accès complet au répertoire d'où a été lancé le programme.

Notez cependant qu'entre le caractère nul qui termine le bloc et le début de la châine du nom du programme se trouve un mot. Ce mot, terminé par un caractère nul, doit contenir la valeur 0001h pour que le nom du programme soit valable.

PSP ou non ?

Si la zone de mémoire n'est pas reconnue comme un bloc d'environnement, il peut s'agir d'un PSP, d'un programme transitoire ou résident. Le test consiste à lire les deux premières mémoires de la zone : en tête de chaque PSP se présentent les instructions machine INT 20h (code 0CDh, 020h).

Si ce test ne donne rien, il n'est pas possible de savoir si la zone de mémoire contient des instructions, des données ou quoi que ce soit d'autre. Pour vous permettre de vous faire votre idée personnelle à ce sujet, le programme affiche les premiers 80 octets de la zone sous forme de "dump" hexadécimal et ASCII.

Une fois que l'utilisateur a répondu à l'invitation qui lui est faite de taper une touche, le MCB suivant est examiné et le programme se termine lorsque le dernier MCB a été atteint.

Exemple d'affichage du programme

Pour vous aider à interpréter les affichages obtenus, les pages suivantes vous en donnent un exemple obtenu avec le programme en C sur l'ordinateur de l'auteur. Vous trouverez un peu plus loin une explication individuelle des différents MCB.

```
MCBC (c) 1988 by Michael Tischer

Numéro de MCB - 1
Adresse MCB   - 09C8:0000
Adr. mémoire  - 09C9:0000
ID            - M
Adresse PSP   - 0008:0000
Taille        - 1554 paragraphes ( 24864 octets )
Contenu       - non identifiable (programme ou données)

DUMP | 0123456789ABCDEF     00 01 02 03 04 05 06 07 08 09 0A 0B 0C 0D 0E 0F
-----|-------------------------------------------------------------------
0000 | n p  Ç! , $CLOCK     6E 01 70 00 08 80 21 00 2C 00 24 43 4C 4F 43 4B
0010 |    Ç                 20 06 03 80 02 1F 1D 1F 1E 1F 1E 1F 1F 1E 1F 1E
0020 |  .ë   .↑    PSQR     1F 2E 89 1E 11 00 2E 8C 06 13 00 CB 50 53 51 52
0030 | WVU  £η  -  &è]     57 56 55 1E 06 9C FC 0E 1F C4 3E 11 00 26 8A 5D
0040 |  Ç√ t!Ç√ tJ ■u Θ     02 80 FB 04 74 21 80 FB 08 74 4A 0A DB 75 03 E9
---------------------------------------- Veuillez frapper une touche ---
Numéro de MCB - 2
Adresse MCB   - 0FDB:0000
Adr. mémoire  - 0FDC:0000
ID            - M
Adresse PSP   - 0FDC:0000
Taille        - 231 paragraphes ( 3696 octets )
Contenu       - PSP (suivi d'un programme)
---------------------------------------- Veuillez frapper une touche ---
Numéro de MCB - 3
Adresse MCB   - 10C3:0000
Adr. mémoire  - 10C4:0000
ID            - M
Adresse PSP   - 0000:0000
Taille        - 3 paragraphes ( 48 octets )
Contenu       - non identifiable (programme ou données)

DUMP | 0123456789ABCDEF     00 01 02 03 04 05 06 07 08 09 0A 0B 0C 0D 0E 0F
-----|-------------------------------------------------------------------
0000 |                      00 01 00 00 00 00 00 CB 00 00 00 FF FF FF FF FF
0010 |                C     FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF 43

0020 | :\AUTOEXEC.BAT       3A 5C 41 55 54 4F 45 58 45 43 2E 42 41 54 00 00
0030 | M ■                  4D DC 0F 0A 00 00 00 00 00 00 00 00 00 00 00 00
0040 | COMSPEC=C:\COMMA     43 4F 4D 53 50 45 43 3D 43 3A 5C 43 4F 4D 4D 41
---------------------------------------- Veuillez frapper une touche ---
Numéro de MCB - 4
Adresse MCB   - 10C7:0000
Adr. mémoire  - 10C8:0000
ID            - M
Adresse PSP   - 0FDC:0000
Taille        - 10 paragraphes ( 160 octets )
Contenu       - environnement
Nom du progr. - inconnu
Chaînes d'environnement
          COMSPEC=C:\COMMAND.COM
          PATH=C:\;C:\DOS;C:\FONREV;E:\;D:\MSC\BIN
          INCLUDE=d:\msc\include
          LIB=d:\msc\lib
          TMP=d:\msc\tmp
          PROMPT=[$p]
---------------------------------------- Veuillez frapper une touche ---
```

```
Numéro de MCB - 5
Adresse MCB   - 10D2:0000
Adr. mémoire  - 10D3:0000
ID            - M
Adresse PSP   - 10DD:0000
Taille        - 9 paragraphes ( 144 octets )
Contenu       - environnement
Nom du progr. - C:\DOS\KEYB.COM
Chaînes d'environnement
          COMSPEC=C:\COMMAND.COM
          PATH=C:\;C:\DOS;C:\FONREV;E:\;D:\MSC\BIN
          INCLUDE=d:\msc\include
          LIB=d:\msc\lib
          TMP=d:\msc\tmp
---------------------------------------- Veuillez frapper une touche ---
Numéro de MCB - 6
Adresse MCB   - 10DC:0000
Adr. mémoire  - 10DD:0000
ID            - M
Adresse PSP   - 10DD:0000
Taille        - 341 paragraphes ( 5456 octets )
Contenu       - PSP (suivi d'un programme)
---------------------------------------- Veuillez frapper une touche ---
Numéro de MCB - 7
Adresse MCB   - 1232:0000
Adr. mémoire  - 1233:0000
ID            - M
Adresse PSP   - 123D:0000
Taille        - 9 paragraphes ( 144 octets )
Contenu       - environnement

Nom du progr. - C:\DOS\CED.COM
Chaînes d'environnement
          COMSPEC=C:\COMMAND.COM
          PATH=C:\;C:\DOS;C:\FONREV;E:\;D:\MSC\BIN
          INCLUDE=d:\msc\include
          LIB=d:\msc\lib
          TMP=d:\msc\tmp
---------------------------------------- Veuillez frapper une touche ---
Numéro de MCB - 8
Adresse MCB   - 123C:0000
Adr. mémoire  - 123D:0000
ID            - M
Adresse PSP   - 123D:0000
Taille        - 1030 paragraphes ( 16480 octets )
Contenu       - PSP (suivi d'un programme)
---------------------------------------- Veuillez frapper une touche ---
Numéro de MCB - 9
Adresse MCB   - 1643:0000
Adr. mémoire  - 1644:0000
ID            - M
Adresse PSP   - 164E:0000
Taille        - 9 paragraphes ( 144 octets )
Contenu       - environnement
Nom du progr. - C:\DOS\CACHE-AT.COM
Chaînes d'environnement
          COMSPEC=C:\COMMAND.COM
          PATH=C:\;C:\DOS;C:\FONREV;E:\;D:\MSC\BIN
          INCLUDE=d:\msc\include
```

```
          LIB=d:\msc\lib
          TMP=d:\msc\tmp
---------------------------------------- Veuillez frapper une touche ---
Numéro de MCB = 10
Adresse MCB   = 164D:0000
Adr. mémoire  = 164E:0000
ID            = M
Adresse PSP   = 164E:0000
Taille        = 1922 paragraphes ( 30752 octets )
Contenu       = PSP (suivi d'un programme)
---------------------------------------- Veuillez frapper une touche ---
Numéro de MCB = 11
Adresse MCB   = 1DD0:0000
Adr. mémoire  = 1DD1:0000
ID            = M
Adresse PSP   = 1DDC:0000
Taille        = 10 paragraphes ( 160 octets )
Contenu       = environnement
Nom du progr. = C:\DOS\KEYBUF.COM
Chaînes d'environnement

          COMSPEC=C:\COMMAND.COM
          PATH=C:\;C:\DOS;C:\FONREV;E:\;D:\MSC\BIN
          INCLUDE=d:\msc\include
          LIB=d:\msc\lib
          TMP=d:\msc\tmp
          PROMPT=[$p]
---------------------------------------- Veuillez frapper une touche ---
Numéro de MCB = 12
Adresse MCB   = 1DDB:0000
Adr. mémoire  = 1DDC:0000
ID            = M
Adresse PSP   = 1DDC:0000
Taille        = 27 paragraphes ( 432 octets )
Contenu       = non identifiable (programme ou données)

DUMP | 0123456789ABCDEF     00 01 02 03 04 05 06 07 08 09 0A 0B 0C 0D 0E 0F
-----|-------------------------------------------------------------------------
0000 |  M M M M M M M M     00 4D 00 4D 00 4D 00 4D 00 4D 00 4D 00 4D 00 4D
0010 |  M M M M M M M M     00 4D 00 4D 00 4D 00 4D 00 4D 00 4D 00 4D 00 4D
0020 | + 1  K K K K K K     2B 1B 31 02 00 4B 00 4B 00 4B 00 4B 00 4B 00 4B
0030 |  K K K K K K K K     00 4B 00 4B 00 4B 00 4B 00 4B 00 4B 00 4B 00 4B
0040 |  K K K K K K K K     00 4B 00 4B 00 4B 00 4B 00 4B 00 4B 00 4B 00 4B
---------------------------------------- Veuillez frapper une touche ---
Numéro de MCB = 13
Adresse MCB   = 1DF7:0000
Adr. mémoire  = 1DF8:0000
ID            = M
Adresse PSP   = 0FDC:0000
Taille        = 4 paragraphes ( 64 octets )
Contenu       = PSP (suivi d'un programme)
---------------------------------------- Veuillez frapper une touche ---
Numéro de MCB = 14
Adresse MCB   = 1DFC:0000
Adr. mémoire  = 1DFD:0000
ID            = M
Adresse PSP   = 1E08:0000
Taille        = 10 paragraphes ( 160 octets )
Contenu       = environnement
```

```
Nom du progr. = D:\PCI\C\TC.EXE
Chaînes d'environnement
          COMSPEC=C:\COMMAND.COM
          PATH=C:\;C:\DOS;C:\FONREV;E:\;D:\MSC\BIN
          INCLUDE=d:\msc\include
          LIB=d:\msc\lib
          TMP=d:\msc\tmp
          PROMPT=[$p]
---------------------------------------- Veuillez frapper une touche ---

Numéro de MCB = 15
Adresse MCB   = 1E07:0000
Adr. mémoire  = 1E08:0000
ID            = M
Adresse PSP   = 1E08:0000
Taille        = 16200 paragraphes ( 259200 octets )
Contenu       = PSP (suivi d'un programme)
---------------------------------------- Veuillez frapper une touche ---
Numéro de MCB = 16
Adresse MCB   = 5D50:0000
Adr. mémoire  = 5D51:0000
ID            = M
Adresse PSP   = 5D5C:0000
Taille        = 10 paragraphes ( 160 octets )
Contenu       = environnement
Nom du progr. = C:\TC\OBEX\MEMDEMO.EXE
Chaînes d'environnement
          COMSPEC=C:\COMMAND.COM
          PATH=C:\;C:\DOS;C:\FONREV;E:\;D:\MSC\BIN
          INCLUDE=d:\msc\include
          LIB=d:\msc\lib
          TMP=d:\msc\tmp
          PROMPT=[$p]
---------------------------------------- Veuillez frapper une touche ---
Numéro de MCB = 17
Adresse MCB   = 5D5B:0000
Adr. mémoire  = 5D5C:0000
ID            = M
Adresse PSP   = 5D5C:0000
Taille        = 4512 paragraphes ( 72192 octets )
Contenu       = PSP (suivi d'un programme)
---------------------------------------- Veuillez frapper une touche ---
Numéro de MCB = 18
Adresse MCB   = 6EFC:0000
Adr. mémoire  = 6EFD:0000
ID            = Z
Adresse PSP   = 0000:0000
Taille        = 12547 paragraphes ( 200752 octets )
Contenu       = non identifiable (programme ou données)

DUMP | 0123456789ABCDEF    00 01 02 03 04 05 06 07 08 09 0A 0B 0C 0D 0E 0F
-----|-------------------------------------------------------------------
0000 | H   6   % ê   ¿     48 00 00 00 36 00 08 00 25 00 88 08 00 00 A8 00
0010 | (   v     Ω   ·     28 04 1E 00 76 00 17 00 FF FF 96 08 00 00 27 0C
0020 | H   Ω     Ω É .     48 1F 00 00 51 00 1E 00 FF FF 96 08 90 01 2E 05
0030 | HB  6   % Ω ∞ B     48 42 00 00 36 00 08 00 25 00 96 08 EC 01 42 00
0040 | ( π v     ñ   E     28 0D E3 00 76 00 17 00 FF FF A4 08 00 00 45 0C
---------------------------------------- Veuillez frapper une touche ---
```

1 Le premier MCB n'a pu être identifié par le programme (l'adresse de PSP indiquée ne présente donc aucun intérêt) mais l'extrait de mémoire affiché permet de reconnaître son contenu. Dans la première ligne du dump ASCII figure le mot $CLOCK, qui est le nom sous lequel le DOS désigne le driver de l'horloge interne. Il semble effectivement qu'il s'agisse bien de la mémoire occupée par un driver de périphérique car la structure des 18 premiers octets correspond exactement à celle de l'en-tête d'un driver de périphérique. Il ne peut toutefois s'agir ici de l'un des drivers de DOS installés à demeure car ces derniers sont installés en dessous de la TPA (Transient Program Area), de sorte qu'il n'est pas nécessaire de leur allouer de la mémoire. Nous avons donc probablement un driver installé lors du lancement du système par le moyen d'une instruction DEVICE située à l'intérieur du fichier de configuration CONFIG.SYS. Il se trouve effectivement que le premier driver que j'ai installé dans ce fichier est le driver HEUREAT.SYS dont le nom de périphérique est justement $CLOCK. Ce driver n'occupe cependant que quelques Ko, alors que la zone de mémoire allouée s'étend bien au-delà. Il doit donc y avoir un autre code de programme ou d'autres données à la suite de ce driver. En examinant les 5 lignes du dump affichées par le programme, on s'aperçoit que ce driver est suivi de tous les autres drivers intégrés avec l'instruction DEVICE. La première zone de mémoire allouée est donc réservée par DOS dès le lancement du système (opération de Boot) pour recevoir les drivers de périphérique dans l'ordre où ils sont indiqués dans le fichier de configuration.

2 Cette zone de mémoire contient de toute évidence un programme. Comme elle n'est précédée d'aucun PSP qui pourrait en indiquer le nom, ce nom restera inconnu. La situation de ce programme dans la chaîne des MCB et dans la mémoire permet cependant de constater qu'il y a été placé peu après l'opération de lancement du système comme programme résident.

3 Il est impossible de formuler des affirmations sur le contenu de cette zone de mémoire. Soit elle a été réservée par un programme qui avait l'intention d'y placer des données par la suite, soit elle est tout simplement restée "en rade" lors de la libération de parcelles de la mémoire.

4 Il s'agit sans hésitation possible d'un environnement, mais le nom du programme correspondant fait défaut. Comme adresse du PSP on trouve 0FDC:0000. Il s'y trouve la zone de mémoire qui a été allouée à travers le MCB 2. Comme le MCB 2 est un PSP et le MCB 4 un environnement, on a tout lieu de supposer qu'on a ici affaire à un programme et à l'environnement correspondant. Le bloc d'environnement ne contenant pas de nom de programme, on peut supposer que le programme du MCB 2 n'a pas été chargé ni lancé oar le moyen de l'interface utilisateur de DOS ni avec une instruction d'un fichier BATCH. Il semble plutôt que le MCB 2 représente la partie résidente de l'interpréteur de commandes COMMAND.COM qui aurait placé son environnement dans le MCB 4. En examinant le code programme du MCB 2 à l'aide d'un débogueur, on peut confirmer pleinement cette hypothèse.

5 On trouve ici l'environnement du programme KEYB.COM, qui permet de travailler avec un clavier français et qui est lancé dans le cadre de mon AUTOEXEC.BAT par l'instruction KEYB FR. Comme ce programme reste toujours en mémoire, c'est qu'il s'agit d'un programme résident.

6 Alors que le MCB 5 contient l'environnement du programme KEYB.COM, nous rencontrons ici sa mémoire (qui contient le PSP, les instructions et les données). La preuve en est que l'adresse du PSP dans le bloc d'environnement (MCB 5) désigne la zone de mémoire qui est gérée par ce MCB.

7, 8 Ces deux zones de mémoire contiennent l'environnement et le PSP (code programme) du programme CED.COM invoqué dans mon fichier AUTOEXEC.BAT. Comme il se trouve toujours en mémoire, c'est qu'il s'agit visiblement d'un programme résident.

9, 10 Mêmes explications que pour 7 et 8 mais concerne le programme CACHEAT.COM.

11, 12 Nous retrouvons encore des zones de mémoire occupées par un programme (KEYBUF.COM du chapitre xx) et son environnement. L'environnement peut être identifié clairement mais pas le PSP. En effet ce programme détourne son PSP de sa fonction originelle en l'employant comme buffer du clavier, écrasant ainsi l'appel d'interruption qui figurait au début du PSP. Or c'est justement cet appel qui nous sert à reconnaître un PSP. En conséquence, la zone de mémoire ne peut être identifiée comme PSP.

13 Le programme nous indique bien que nous avons affaire à un PSP mais il ne peut s'agir ici que du début d'un PSP, cette zone de mémoire comportant seulement 64 octets, alors qu'un PSP en occupe 256. Apparemment un PSP n'a pas été totalement libéré, de sorte qu'il en reste une partie en mémoire.

14, 15 Le programme d'affichage des MCB ayant été écrit en C, je me trouvais à l'intérieur de l'environnement de Turbo C lorsque je l'ai lancé pour produire l'extrait décrit ici. C'est pourquoi l'environnement et les instructions du programme Turbo C figurent dans les MCB 14 et 15.

16, 17 Pour obtenir un dump des MCB ou de la mémoire, le programme MEMDEMO a été compilé, linké et enfin exécuté à l'intérieur de l'environnement Turbo C. Turbo C a lancé l'exécution en appelant ce programme comme un processus subordonné, à l'aide du chargeur EXEC. Les zones de mémoire 16 et 17 ont donc été allouées par le chargeur EXEC pour l'exécution du programme. Une fois le programme terminé, elles ont dû être libérées.

18 Le dernier bloc de mémoire représente le reste de la mémoire non encore allouée. Dans ce cas précis, sa taille est de 200 Ko.

Les trois programmes qui suivent, écrits en Basic, Pascal et C, affichent le dump des MCB comme il a été présenté plus haut. Ces programmes se ressemblant beaucoup, tant dans leur logique que par les fonctions et variables utilisées, nous allons les décrire ensemble.

Seul la version en Basic présente quelques différences car la lecture de la mémoire ne peut pas s'y faire par des pointeurs FAR : il faut employer les instructions PEEK et DEF SEG.

Les programmes en C et Pascal accèdent à la mémoire à l'aide de pointeurs FAR car les zones de mémoire à adresser se situent en dehors de leur segment de données. Si les pointeurs sont automatiquement du type FAR sous Turbo Pascal, ce n'est le cas en C que si vous sélectionnez un modèle de mémoire approprié (Compact, Huge ou Large) ou si vous utilisez un opérateur de transtypage pour définir explicitement un pointeur FAR. C'est la seconde méthode que nous avons employée dans ce programme pour permettre une compilation même avec un modèle de mémoire travaillant de façon standard avec des pointeurs NEAR (Tiny, Small et Medium).

Notez qu'en C un problème se pose pour convertir en un pointeur FAR un segment et un offset obtenus séparément. Il existe heureusement une macro prédéfinie dans le fichier d'inclusion DOS.H de Turbo C, mais cette macro manque en Microsoft C. C'est pourquoi elle fait l'objet d'une définition spéciale pour prévenir son absence éventuelle. En Pascal, la conversion des adresses en pointeur FAR est effectuée à l'aide de la fonction prédéfinie PTR

Les deux listings sont commentés de façon suffisamment explicite pour que nous arrêtions là ces observations préliminaires.

Listing : MCBP.PAS

```
{*********************************************************************}
{*                          M C B P . P A S                          *}
{*-------------------------------------------------------------------*}
{*   Fonction       : Permet de suivre le chaînage des blocs de      *}
{*                    mémoire alloués par DOS                        *}
{*-------------------------------------------------------------------*}
{*   Auteur         : Michael TISCHER                                *}
{*   Développé le   : 22.08.1988                                     *}
{*   Dernière MAJ   : 20.03.1992                                     *}
{*********************************************************************}

program MCBP;

uses DOS, CRT;                          { Intègre les unités DOS et CRT }

type BytePtr = ^byte;                              { Pointeur sur octet }
     Zone    = array[0..1000] of byte;       { Zone qlcq de la mémoire }
     ZonePtr = ^Zone;                             { Pointeur sur zone }
     MCB     = record                          { Memory-Control-Bloc }
         IdCode   : char;     { "M": existe un bloc qui suit, "Z": Fin }
         PSP      : word;                  { Segment du PSP associé }
         Distance : word;               { Nombre de paragraphes - 1 }
               end;
     MCBPtr  = ^MCB;                                { Pointeur sur MCB }
     MCBPtr2 = ^MCBPtr;              { Pointeur sur un pointeur de MCB }
     HexStr  = string[4];          { Mémorise chaîne hexa. à 4 chiffres }

var CvHStr  : HexStr;              { Mémorise la chaîne hexa convertie }

{*********************************************************************}
{* HexString: Transforme un nombre en chaîne hexadécimale            *}
{*            à 4 positions                                          *}
{* Entrée : - HexVal = Valeur à convertir                            *}
{* Sortie : La chaîne hexa .                                         *}
{*********************************************************************}

function HexString( HexVal : word ) : HexStr;

var compteur,                                  { Compteur d'itérations }
    Nibble  : byte;                      { Quartet inférieur d'un mot }

begin
  CvHStr := 'xxxx';                                  { Crée la chaîne }
  for compteur:=4 downto 1 do                 { Parcourt ses 4 positions }
    begin
      Nibble := HexVal and $000f;     { Ne prend en cmpt que 4 bits sup }
      if( Nibble > 9 ) then                        { Chiffre = lettre ? }
        CvHStr[ compteur ] := chr(Nibble - 10 + ord('A'))       { Oui }
      else                                        { Chiffre entre 0 et 9 }
        CvHStr[ compteur ] := chr(Nibble + ord('0'));
      HexVal := HexVal shr 4;
    end;
  HexString := CvHStr;                       { Transmet la chaîne trouvée }
end;

{*********************************************************************}
{* FirstMCB: Retourne un pointeur sur le premier MCB.                *}
{* Entrée  : néant                                                   *}
{* Sortie : Pointeur sur le premier MCB                              *}
{*********************************************************************}
```

```
!
!function FirstMCB : MCBPtr;
!
!var Regs : Registers;          { Registres pour gérer les interruptions }
!
!begin
!  Regs.ah := $52;              { Fonction: Lit l'adresse du DOS-Info-Bloc }
!  MsDos( Regs );                 { Déclenche l'interruption $21 de DOS }
!
!{*-- ES:(BX-4) pointe sur le premier  MCB, forme le pointeur ---------*}
!
!  FirstMCB := MCBPtr2( ptr( Regs.ES-1, Regs.BX+12 ) )^;
!end;
!
!{*****************************************************************}
!{* Dump: Affiche le dump hexa et ASCII d'une zone de mémoire     *}
!{* Entrée : - DPtr  = Pointeur sur la zone à dumper.             *}
!{*          - Nbr   = Nombre de lignes de 16 octets à dumper     *}
!{* Sortie : néant                                                *}
!{*****************************************************************}
!
!procedure Dump( DPtr : ZonePtr; Nbr : byte);
!
!type HBStr = string[2];         { Mémorise un nombre hexa à 2 chiffres }
!
!var  Offset,                          { Offset dans la zone de mémoire }
!     Z       : integer;                                  { Compteur }
!     HexStr  : HBStr;          { Mémorise un nombre hexa pour le dump }
!
!procedure HexByte( HByte : byte );
!
!begin
!  HexStr[1] := chr( (HByte shr 4) + ord('0') );      { Premier chiffre }
!  if HexStr[1] > '9' then                        { Est-ce une lettre ? }
!    HexStr[1] := chr( ord(HexStr[1]) + 7 );                  { Oui }
!  HexStr[2] := chr( (HByte and 15) + ord('0') );   { Deuxième chiffre }
!  if HexStr[2] > '9' then                        { Est-ce une lettre ? }
!    HexStr[2] := chr( ord(HexStr[2]) + 7 );                  { Oui }
!end;
!
!begin
!  HexStr := 'zz';                                  { Crée la chaîne hexa }
!  writeln;
!  write('DUMP  0123456789ABCDEF       00 01 02 03 04 05 06 07 08');
!  writeln(' 09 0A 0B 0C 0D 0E 0F');
!  write('');
!  writeln('');
!  Offset := 0;              { Commence par le premier octet de la zone }
!  while Nbr>0 do                          { Parcourt la boucle Nbr fois }
!    begin
!      write(HexString(Offset), '  ');
!      for Z:=0 to 15 do                    { Traite 16 octets à la fois }
!        if(Dptr^[Offset+Z] >= 32) then          { ASCII ordinaire ? }
!          write( chr(Dptr^[Offset+Z]) )                { Oui, l'affiche }
!        else                                                 { Non }
!          write(' ');      { A la place du caractère affiche un espace }
!      write('      '); { Place le curseur sur la partie hexadécimale }
!      for Z:=0 to 15 do                              { Traite 16 octets }
!        begin
!          HexByte( Dptr^[Offset+Z] );              { Conversion en hexa }
!          write(HexStr, ' ');                  { Affiche la chaîne hexa }
!        end;
!      writeln;
!      Offset := Offset + 16;                { Offset sur ligne suivante }
!      Dec( Nbr );             { Décrémente le nombre de lignes restantes }
!    end;
!  writeln;
!end;
!
!{*****************************************************************}
!{* TraceMCB: Suit la chaîne des blocs MCB.                       *}
!{* Entrée  : néant                                               *}
!{* Sortie  : néant                                               *}
!{*****************************************************************}
!
!procedure TraceMCB;
!
!const ComSpec : array[0..7] of char = 'COMSPEC=';
!
!var  ActMCB   : MCBPtr;
!     Fin      : boolean;
!     Touche   : char;
!     NrMCB,                                  { Numéro du MCB examiné }
!     Z        : integer;                                 { Compteur }
!     MemPtr   : ZonePtr;
!
!begin
!  Fin    := false;
!  NrMCB  := 1;                    { Le premier MCB porte le numero 1 }
!  ActMCB := FirstMCB;              { Lit le pointeur sur premier MCB }
!  repeat                                              { Suit la chaîne }
!    if ActMCB^.IdCode = 'Z' then                { Dernier MCB atteint ? }
!      Fin := true;                                             { Oui }
!    writeln('Numéro du MCB = ', NrMCB);
!    writeln('Adresse du MCB= ', HexString(seg(ActMCB^)), ':',
!                                HexString(ofs(ActMCB^)) );
!    writeln('Adr. Mémoire  = ', HexString(succ(seg(ActMCB^))), ':',
!                                HexString(ofs(ActMCB^)) );
!    writeln('ID            = ', ActMCB^.IdCode);
!    writeln('Adresse du PSP= ', HexString(ActMCB^.PSP), ':0000');
!    writeln('Taille        = ', ActMCB^.Distance, ' paragraphes ',
!        '( ', longint(ActMCB^.Distance) shl 4, ' octets)');
!    write  ('Contenu       = ');
!
!            {*-- Est-ce un environnement  ? ------------------------*}
!
!    Z      := 0;                { Commence l'examen par le premier octet }
!    MemPtr := ZonePtr(ptr(Seg(ActMCB^)+1, 0));     { Ptr sur mém. vive }
!    while( (Z<=7) and (ord(ComSpec[Z]) = MemPtr^[Z]) ) do
!      Inc(Z);                                  { Z sur caractère suivant }
!    if Z>7 then                              { A-t-on découvert la chaîne ? }
!      begin                                { Oui, il s'agit d'un environnement }
!        writeln('Environnement');
!        MemPtr := ZonePtr(ptr(Seg(ActMCB^)+1, 0));
!        if Lo(DosVersion) >= 3 then          { DOS 3.0 ou postérieure ? }
!        begin                                { Oui, donne le nom du programme }
!          write('Nom du progr  = ');
!          Z := 0;                            { Commence par le premier octet}
!          while not( (MemPtr^[Z]=0) and (MemPtr^[Z+1]=0) ) do
!          Inc( Z );                                { Cherche chaîne vide }
!          if( MemPtr^[Z+2]=1 ) and ( MemPtr^[Z+3]=0 ) then
!            begin                                  { Voici un nom de programme }
!              Z := Z + 4;                          { Z sur le début du nom }
!              repeat                               { Parcourt le nom du programme }
!                write( chr(MemPtr^[Z]) );          { Affiche un caractère }
!                Inc( Z );                          { Traite le caractère suivant }
!              until ( MemPtr^[Z]=0 );              { Jusqu'à la fin de la chaîne }
!              writeln;
!            end
!          else                          { Nom du programme impossible à trouver }
!            writeln('inconnu');
!        end;
!
!        {*-- Affiche les chaînes de l'environnement ------------------*}
!
!        writeln(#13,#10, 'Chaînes de l''environnement');
!        Z := 0;     { Commence par le premier octet de la zone allouée }
!        while MemPtr^[Z]<>0 do                { répète jusqu'à chaîne vide }
!        begin
!          write('      ');
!          repeat                                          { Affiche un chaîne }
!            write( chr(MemPtr^[Z]) );                  { Affiche un caractère}
!            Inc( Z );                              { Traite le caractère suivant}
!          until MemPtr^[Z]=0;                              { Fin de la chaîne }
!          Inc( Z );                { Passe au début de la chaîne suivante }
!          writeln;                                      { Termine la ligne }
!        end
!      end
!    else                                              { Pas d'environnement }
!      begin
!        {*-- S'agit-il d'un PSP ? ---------------------------------*}
!        {*-- (introduit par la commande INT 20 (Code=$CD $20) ) ------*}
!
!        MemPtr := ZonePtr(ptr(seg(ActMCB^)+1, 0));          { Init. ptr }
!        if( (MemPtr^[0]=$CD) and (MemPtr^[1]=$20) ) then
!                                                        { c'est bien un PSP }
!          writeln('PSP (suivi d''un programme)')
!        else                { La commande INT 20 n'a pas pu être détectée }
!          begin
!            writeln('non identifiable (Programme ou données)');
!            Dump( MemPtr, 5);          { Effectue un dump de 5*16 octets }
!          end;
!      end;
!
!    write('');
!    writeln(' Appuyez sur une touche');
!    if( not Fin ) then
!      begin                                  { Pointeur sur MCB suivant }
!        ActMCB := MCBPtr(ptr(seg(ActMCB^) + ActMCB^.Distance + 1, 0));
!        Inc(NrMCB);                          { Incrémente le numéro du MCB }
!        Touche := ReadKey;
!      end;
!  until( Fin )                  { répète opération jusqu'au dernier MCB }
!end;
!
!{*****************************************************************}
!{**                       PROGRAMME PRINCIPAL                   **}
!{*****************************************************************}
!
!begin
!  ClrScr;                                             { Efface l'écran }
!  writeln( 'MCBP  -  (c) 1988, 92 by Michael TISCHER' );
!  writeln;
!  writeln;
!  TraceMCB;                                  { Parcourt la chaîne des MCB }
!end.
```

Listing : MCBC.C

```
/*****************************************************************
*                          M C B C . C                           *
*****************************************************************
*  Fonction      : Permet de suivre le chainage des blocs de     *
*                  mémoire alloués par DOS                       *
*****************************************************************
*  Auteur        : MICHAEL TISCHER                               *
*  Développé le  : 23.08.1988                                    *
*  Dernière MAJ  : 20.03.1982                                    *
*****************************************************************
*  Modèle mémoire : SMALL                                       *
*****************************************************************/

/*--- Fichiers d'inclusion ---------------------------------------*/

#include <stdio.h>
#include <dos.h>
#include <stdlib.h>
#include <conio.h>

/*--- Typedef ----------------------------------------------------*/

typedef unsigned char BYTE;             /* Bricolage d'un type Byte */
typedef unsigned int  WORD;
typedef BYTE          BOOLEAN;
typedef BYTE far      *FB;              /* Pointeur FAR sur un octet */

/*--- Constantes -------------------------------------------------*/

#define TRUE  ( 0 == 0 )                /* Utilisé avec le type BOOLEAN */
#define FALSE ( 1 == 0 )

/*--- Structures et unions ---------------------------------------*/

struct MCB
     {                                  /* Décrit un bloc MCB en mémoire */
       BYTE id_code; /* 'M' = il existe un bloc qui suit , 'Z' = Fin */
       WORD psp;                        /* Segment du PSP associé */
       WORD distance;                   /* Nombre de paragraphes réservés */
     };

typedef struct MCB far *MCBPtr;         /* Pointeur FAR sur un MCB */

/*--- Macros -----------------------------------------------------*/

#ifdef MK_FP                            /* MK_FP déjà défini ? */
  #undef MK_FP
#endif

#define MK_FP(s, o) ((void far *) (((unsigned long) (s)<<16)|(o)))

/*****************************************************************
*  Fonction      : F I R S T _ M C B                            *
*****************************************************************
*                  Retourne un pointeur sur le premier MCB.     *
*  Entrée : néant                                               *
*  Sortie : Pointeur sur le premier MCB                         *
*****************************************************************/

MCBPtr first_mcb( void )
{
  union REGS  regs;             /* Registres pour gérer les interruptions */
  struct SREGS sregs;           /* Mémorise les registres de segment */

  regs.h.ah = 0x52;     /* Fonction: "Lire l'adresse du DOS-Info-Bloc" */
  intdosx( &regs, &regs, &sregs );          /* interruption DOS 0x21 */

    /*-- ES:(BX-4) pointe sur le premier MCB, forme le pointeur ------*/

  return( *((MCBPtr far *) MK_FP( sregs.es-1, regs.x.bx+11 )) );
}

/*****************************************************************
*  Foction       : D U M P                                      *
*****************************************************************
*            Affiche le dump hexa et ASCII d'une zone de mémoire *
*  Entrées : BPTR = Pointeur sur la zone de mémoire              *
*            Nbr  = Nombre de lignes du dump (par 16 octets)     *
*  Valeur de retour : néant                                     *
*****************************************************************/

void dump( FB bptr, BYTE nbr)
{
  FB   lptr;                    /* Pointeur courant sur ligne de dump */
  WORD offset;                  /* Offset par rapport à BPTR */
  BYTE i;                       /* Compteur */

  printf("\nDUMP  0123456789ABCDEF        00 01 02 03 04 05 06 07 08");
  printf(" 09 0A 0B 0C 0D 0E 0F\n");
  printf("");
  printf("\n");

  for (offset=0;  nbr-- ; offset += 16, bptr += 16)
  {                                     /* Parcourt la boucle NBR fois */
    printf("%04x  ", offset);
    for (lptr=bptr, i=16; i-- ; ++lptr)     /* Affiche caractère ASCII */
      printf("%c", (*lptr<32) ? ' ' : *lptr);
    printf("      ");
    for (lptr=bptr, i=16; i-- ; )           /* Affiche équivalent hexa */
      printf("%02X ", *lptr++);
    printf("\n");                           /* Passe à la ligne suivante */
  }
}

/*****************************************************************
*  Fonction      : T R A C E _ M C B                            *
*****************************************************************
*                  Suit la chaîne des blocs MCB.                *
*  Entrée : néant                                               *
*  Valeur de retour : néant                                     *
*****************************************************************/

void trace_mcb( void )
{
  static char fenv[] =
              {                         /* Première chaîne d'environnement */
                'C', 'O', 'M', 'S', 'P', 'E', 'C', '='
              };

  MCBPtr   act_mcb;                     /* Pointeur sur MCB actuel */
  BOOLEAN  fin;                         /* TRUE si le dernier MCB est trouvé */
  BYTE     nr_mcb,                      /* Numéro du MCB présentement traité */
           i;                           /* Compteur */
  FB       lptr;                        /* Pointeur sur l'environnement */

  fin    = FALSE;                       /* On y va */
  nr_mcb = 1;                           /* Le premier MCB porte le numéro 1 */
  act_mcb = first_mcb();                /* Cherche un pointeur sur le premier MCB */

  do                                    /* Traite les différents MCB */
  {
    if( act_mcb->id_code == 'Z' )       /* Dernier MCB atteint ? */
      fin = TRUE;                       /* Oui */
    printf("Numéro du MCB = %d\n", nr_mcb++);
    printf("Adresse du MCB= %Fp\n", act_mcb);
    printf("Adr. Mémoire  = %Np:0000\n", FP_SEG(act_mcb)+1);
    printf("ID            = %c\n", act_mcb->id_code);
    printf("Adresse du PSP= %Fp\n", (FB) MK_FP(act_mcb->psp, 0) );
    printf("Taille        = %u paragraphes ( %lu octets)\n",
           act_mcb->distance, (unsigned long) act_mcb->distance << 4);
    printf("Contenu       = ");

              /*-- Est-ce un environnement ?-------------------------*/

    for (i=0, lptr=(FB)act_mcb+16;   /* Compare première chaîne à FENV */
         ( i<sizeof fenv ) && ( *(lptr++) == fenv[i++] ) ; );

    if( i == sizeof fenv )                  /* Chaîne détectée  ? */
    {                                   /* Oui, il s'agit d'un environnement */
      printf("Environnement\n");
      if( _osmajor >= 3 )               /* Version 3.0 de DOS ou ultérieure ? */
      {                                 /* Oui, donne le nom du programme */
        printf("Nom du progr = ");
        for( ; !(*(lptr++)==0 && *lptr==0) ; );
                                        /* Cherche la dernière chaîne de l'envir. */
        if( *(int far *)(lptr + 1) == 1 )
                                        /* Voici un nom de programme */
          for( lptr += 3; *lptr ; )             /* On le parcourt */
            printf( "%c", *(lptr++) );  /* car, par car, pour afficher */
        else                            /* Pas de programme détecté */
          printf("inconnu ");
        printf("\n");                   /* Passe à la ligne suivante */
      }

      /*-- Affiche les chaînes de l'environnement ------------------*/

      printf("Chaînes de l'environnement\n");
      for (lptr=(FB) act_mcb +16; *lptr ; ++lptr)
      {                                         /* Affiche une chaîne */
        printf("      ");
        for( ; *lptr ; )  /* Parcourt la chaîne jusqu'au caractère NUL */
          printf( "%c", *(lptr++) );  /* Affiche un caractère à la fois*/
        printf("\n");                   /* Passe à la ligne suivante */
      }
```

```
  }
  else                                          /* Pas d'environnement */
  {

    /*-- S'agit-il d'un PSP? ------------------------------------------*/
    /*-- (introduit par la commande INT 20 (Code=0xCD 0x20) ) -------*/

    if (*((unsigned far *) MK_FP( act_mcb->psp, 0 )) == 0x20cd)
      printf("PSP (suivi d'un programme)\n");            /* Oui */
    else          /* La commande INT 0x20 n'a pas pu être détectée */
    {
      printf("non identifiable (Programme ou données)\n");
      dump( (FB) act_mcb + 16, 5);  /* dump des 5*16 premiers octets */
    }
  }

  printf("");
  printf(" Appuyez sur une touche\n");
  if( ifin )                                          /* Autre MCB ? */
  {                                  /* Oui, fixe un pointeur dessus */
    act_mcb = (MCBPtr)
              MK_FP( FP_SEG(act_mcb) + act_mcb->distance + 1, 0 );
    getch();                          /* Attend une frappe de touche */
  }
 }
 while( ifin );                       /* Répète jusqu'au dernier MCB */
}

/*****************************************************************/
/**                      PROGRAMME PRINCIPAL                    **/
/*****************************************************************/

void main( void )
{
 printf("\nMCBC (c) 1988, 92 by Michael TISCHER\n\n");
 trace_mcb();                         /* Parcourt la chaîne des MCB */
}
```

Listing : MCBB.BAS

```
'*****************************************************************
'*                          M C B B . B A S                      *
'*---------------------------------------------------------------*
'*  Fonction      : Permet de suivre le chaînage des blocs de    *
'*                  mémoire alloués par DOS                      *
'*---------------------------------------------------------------*
'*  Auteur        : Michael TISCHER                              *
'*  Développé le  : 16.05.1991                                   *
'*  Dernière MAJ  : 20.10.1991                                   *
'*****************************************************************

DECLARE SUB TraceMCB ()
DECLARE SUB FirstMCB (Adr AS ANY)
DECLARE SUB Dump (Adr AS ANY, Nbre%)
DECLARE FUNCTION HexByte$ (HByte%)
DECLARE FUNCTION GetDosVer% ()
DECLARE FUNCTION GetWord& (SegAdr AS LONG, OfsAdr AS LONG)
DECLARE FUNCTION Hexstring$ (HexVal&)

'$INCLUDE: 'qb.bi'                    'Intègre un  fichier d'inclusion

CONST TRUE = -1                       'Définit des valeurs booléennes
CONST FALSE = NOT TRUE

TYPE AdrType                                      'Pointeur sur adresse
   OfsAdr AS LONG                                          'Offset
   SegAdr AS LONG                                          'Segment
END TYPE

CLS                                                  'Efface l'écran
PRINT "MCBB  -  (c) 1988, 92 by Michael Tischer": PRINT : PRINT
CALL TraceMCB                          'Parcourt la chaîne des MCB
END

'*****************************************************************
'* DUMP   : Affiche le dump hexadécimale et ASCII d'une zone de mémoire*
'* Entrée : SegAdr = Segment de la zone à soumettre au dump      *
'*          Nbre   = Nombre de lignes de 16 octets à afficher    *
'* Sortie : néant                                                *
'*****************************************************************

SUB Dump (SegAdr AS LONG, Nbre AS INTEGER)

DIM HexStr AS STRING * 2       'mémorise un nombre hexa à deux positions
DIM Offset AS LONG                      'Offset dans zone de mémoire

HexStr = "zz"                                       'Crée la chaîne hexa
PRINT
PRINT "DUMP  0123456789ABCDEF       00 01 02 03 ";
PRINT "04 05 06 07 08 09 0A 0B 0C 0D 0E 0F"
PRINT "";
PRINT ""
Offset = 0                                  'Commence par le premier octet
DEF SEG = SegAdr                                  'définir le segment
WHILE Nbre > 0                          'Parcourt la boucle Nbre fois
  PRINT Hexstring$(Offset); "  ";
  FOR z = 0 TO 15                          'Traite 16 octets à la fois
    IF PEEK(Offset + z) >= 32 THEN        'Caractère ASCII ordinaire ?
       PRINT CHR$(PEEK(Offset + z));                'Oui ,on l'affiche
    ELSE
       PRINT " ";                          'Non, on affiche un espace
    END IF
  NEXT
  PRINT "  ";                          'Place le curseur sur la partie hexa
  FOR z = 0 TO 15                        'Traite 16 caractères à la fois
    PRINT HexByte$(PEEK(Offset + z)); " ";          'Aff. octet en hexa
  NEXT
  PRINT                                                   'A la ligne
  Offset = Offset + 16                          'Offset sur ligne suivante
  Nbre = Nbre - 1                 'Décrémente le nombre de lignes restantes
WEND
PRINT
END SUB

'*****************************************************************
'*  FirstMCB : Renvoie un pointeur sur le premier MCB            *
'*  Entrée   : néant                                             *
'*  Sortie   : Pointeur sur le premier MCB dans la variable MCBAdr*
'*****************************************************************

SUB FirstMCB (MCBAdr AS AdrType)

DIM Register AS RegTypeX          'Registres pour gérer les interruptions

Register.ax = &H52 * 256      'Fonction: "Lire l'adresse du DOS-Info-Bloc
CALL INTERRUPTX(&H21, Register, Register)                   ' int. DOS
'-- (ES:BX-&H4)=ES-1:12 donne l'adresse du ptr sur le premier  MCB -----
DEF SEG = Register.es - 1                          'définit le segment
MCBAdr.OfsAdr = PEEK(Register.bx + 13) * 256& + PEEK(Register.bx + 12)
MCBAdr.SegAdr = PEEK(Register.bx + 15) * 256& + PEEK(Register.bx + 14)
END SUB

'*****************************************************************
'*  GetDosVer : Détermine la version de DOS                      *
'*  Entrée    : néant                                            *
'*  Sortie    : numéro de version (30 pour DOS 3.0, 33 pour DOS 3.3 etc)*
'*****************************************************************

FUNCTION GetDosVer%

DIM Register AS RegType                'Registres pour gérer l'interruption

Register.ax = &H30 * 256       'AH = N° de fonc "déterminer version DOS"
CALL INTERRUPT(&H21, Register, Register)                    'int. &H21
GetDosVer = INT(Register.ax \ 256) + (Register.ax MOD 256) * 10
END FUNCTION

'*****************************************************************
'* Getword : Lit un mot dans une adr. mémoire sous forme d'entier long *
'* Entrée  : SegAdr = Segment du mot                             *
'*           OfsAdr = Offset du mot                              *
'* Sortie  : Contenu du mot sous forme d'entier long             *
'*****************************************************************

FUNCTION GetWord& (SegAdr AS LONG, OfsAdr AS LONG)

DEF SEG = SegAdr                                    'Fixe le segment
GetWord& = PEEK(OfsAdr + 1) * 256& + PEEK(OfsAdr)      'Lit la mémoire
END FUNCTION

'*****************************************************************
'* Hexbyte : Convertit un octet en chaîne de caractères hexadécimale *
'* Entrée  : Hbyte = l'octet à convertir                         *
'* Sortie  : Chaîne de caractères obtenue                        *
'* Info    : La fonction QBASIC appelée HEX$ ne retourne pas toujours *
'*           des chaînes de deux caractères comme on le souhaite ici *
```

```
'*****************************************************************
'
FUNCTION HexByte$ (HByte AS INTEGER)
'
DIM HexSt AS STRING * 2                          'mémorise la chaîne hexa
'
MID$(HexSt, 1, 1) = HEX$(HByte \ 16)                    'Premier chiffre
MID$(HexSt, 2, 1) = HEX$(HByte MOD 16)                  'Deuxième chiffre
HexByte$ = HexSt                                 ' Transmet le résultat
END FUNCTION
'
'*****************************************************************
'* HexString : Convertit un entier long en chaîne de car. hexa.  *
'* Entrée    : Valeur à convertir                                *
'* Sortie    : Chaîne hexa obtenue                               *
'* Info      : La fonction de QBASIC appelée HEX$ ne fournit pas *
'*             toujours des chaînes de quatre position comme     *
'*             on le souhaite ici                                *
'*****************************************************************
'
FUNCTION Hexstring$ (HexValVar AS LONG)
'
DIM Nibble AS INTEGER                        'Quartet inférieur d'un mot
DIM HexVal AS LONG           ' Il faut mémoriser l'argument car Q-Basic
                                        'ne travaille pas par référence
DIM HStr AS STRING * 4                          'Chaîne hexa convertie
'
HexVal = HexValVar               ' Mémorise l'argument pour le modifier
HStr = "xxxx"                                       'Crée la chaîne
FOR compteur = 0 TO 3                            'Parcourt les 4 chiffres
   Nibble = HexVal AND &HF              'Ne garde que les 4 bits inférieurs
   MID$(HStr, 4 - compteur, 1) = HEX$(Nibble)     'Conv. quartet en hexa
   HexVal = HexVal \ 16     'Décale HexVal de 4 positions vers la droite
NEXT
Hexstring$ = HStr                                'Transmet la chaîne trouvée
END FUNCTION
'
'*****************************************************************
'* TraceMCB : suit la liste des blocs MCB                        *
'* Entrée   : néant                                              *
'* Sortie   : néant                                              *
'*****************************************************************
'
SUB TraceMCB
'
CONST kom = "COMSPEC="                '"COMSPEC=" déclaré comme constante
'
DIM ActMCB  AS AdrType                                  'Pointeur sur MCB
DIM ID      AS STRING * 1     '"M" il existe un bloc suivant , "Z" = Fin
DIM PSP     AS LONG                                  'Segment du PSP associé
DIM Distance AS LONG                          ' Nombre de paragraphes - 1
DIM MemPtr  AS LONG                                'Pointeur sur mémoire
DIM NrMCB   AS INTEGER                          'Numéro du MCB courant
DIM z       AS INTEGER                             'Compteur de boucle
DIM Fin     AS INTEGER                              'Condition d'arrêt
DIM ActOfs  AS LONG
'
DosVer = GetDosVer                           'Détermine la version de DOS
NrMCB = 1                                    'Commence par le premier MCB
Fin = FALSE
CALL FirstMCB(ActMCB)                  'Lit le pointeur sur le premier MCB
DO
  ActOfs = ActMCB.OfsAdr                                'Charge l'offset
  DEF SEG = ActMCB.SegAdr                  'Fixe le segment pour Peek()
  ID = CHR$(PEEK(ActOfs))                               'Lit le premier MCB
  PSP = GetWord&(ActMCB.SegAdr, ActOfs + &H1)
  Distance = GetWord&(ActMCB.SegAdr, ActOfs + &H3)
'
  IF ID = "Z" THEN                                      'Dernier MCB ?
    Fin = TRUE                                     'Interrompre la boucle
  END IF
  PRINT "Numéro du MCB    ="; NrMCB
  PRINT "Adresse du MCB   = "; Hexstring$(ActMCB.SegAdr); ":";
  PRINT Hexstring$(ActOfs)
  PRINT "Adresse mémoire  = "; Hexstring$(ActMCB.SegAdr + 1); ":";
  PRINT Hexstring$(ActOfs)
  PRINT "ID               = "; ID
  PRINT "Adresse du PSP   = "; Hexstring$(PSP); ":0000"
  PRINT "Taille           = "; Hexstring$(Distance); " paragraphes ( ";
  PRINT Distance * 16; " octets)"
  PRINT "Inhalt           = ";
'
    '---- Est-ce un environnement ? ----------------------------
  z = 0                                          'Commence la comparaison
  MemPtr = ActMCB.SegAdr + 1                        'Pointeur sur mémoire
  DEF SEG = MemPtr                              'Fixe le segment pour Peek()
  WHILE (z <= 7) AND MID$(kom, z + 1, 1) = CHR$(PEEK(ActMCB.OfsAdr + z))
    z = z + 1                                          'Caractère suivant
  WEND
  IF z > 7 THEN                          'On a trouvé la chaîne COMSPEC =
    PRINT "Environnement "
    IF DosVer > 30 THEN                 'Version 3.0 de DOS ou ultérieure ?
      PRINT "Nom du progr.    = ";          'Oui, donne le nom du programme
      z = 0                                'en commençant par le premier octet
      DO
    z = z + 1                                          'Cherche chaîne vide
      LOOP UNTIL PEEK(ActOfs + z) = 0 AND PEEK(ActOfs + z + 1) = 0
      IF PEEK(ActOfs + z + 2) = 1 AND PEEK(ActOfs + z + 3) = 0 THEN
          '--- Nom du programme découvert ---------------------------
    z = z + 4                                  'z sur premier caractère du nom
    DO                                      'Parcourt le nom du programme
      PRINT CHR$(PEEK(ActOfs + z));                    'Affiche un caractère
      z = z + 1                                        'Caractère suivant
    LOOP UNTIL PEEK(ActOfs + z) = 0           'Jusqu'à la fin de la chaîne
    PRINT
      ELSE                                                 'Rien de trouvé
    PRINT "Inconnu"
      END IF
    END IF

    '---- Affiche les chaînes de l'environnement ----------------------

    PRINT "Chaînes de l                                    'environnement"
    z = 0                  'Commence par le premier octet de la zone allouée
    WHILE PEEK(ActOfs + z) <> 0                  'Répète jusuqu'à Chaîne vide
      PRINT "      ";                                          'A la ligne
      DO                                              'Affiche une chaîne
    PRINT CHR$(PEEK(ActOfs + z));                    'Affiche un caractère
    z = z + 1                                          'Caractère suivant
      LOOP UNTIL PEEK(ActOfs + z) = 0                'Jusqu'à fin de chaîne
      z = z + 1                                          'Chaîne suivante
      PRINT                                              'termine la ligne
    WEND
  ELSE
  '---- S'agit-il d'un PSP ?  ---------------------------------------
  '---- (Introduit par la commande (INT 20) (Code &HCD &H20) )----------

    MemPtr = ActMCB.SegAdr + 1                        'Pointeur sur mémoire
    IF PEEK(ActOfs) = &HCD AND PEEK(ActOfs + 1) = &H20 THEN
      PRINT "PSP (suivi d'un programme)"                 'C'est bien un PSP
    ELSE                                'La commande INT 20 n'a pas pu être détectée
      PRINT "Programme non identifiable "
      CALL Dump(MemPtr, 5)     'Effectue un dump des 5*16 premiers octets
    END IF
  END IF
'
  PRINT " Appuyez sur une touche";
  PRINT " "
  DO                                            'Attend une frappe de touche
  a$ = INKEY$
  LOOP UNTIL a$ <> ""
  IF NOT Fin THEN                                        'Il reste un MCB
    ActMCB.SegAdr = ActMCB.SegAdr + Distance + 1    'Ptr sur MCB suivant
    NrMCB = NrMCB + 1
  END IF
LOOP UNTIL Fin  'Répète l'opération jusqu'à ce qu'il n'y ait plus de MCB
END SUB
'
```

23. La fonction EXEC

La fonction utilisée par DOS pour charger et exécuter des programmes est également disponible par le programmeur. En effet, les DOS-Shells à la Norton ou PC Tools ne sont pas les seuls qui ont besoin d'appeler un programme à partir d'un autre.

Ce chapitre décrit l'environnement de la fonction EXEC 4Bh que DOS propose pour réaliser cette tâche. Les programmeurs en langage évolué doivent noter ici qu'il existe des fonctions prédéfinies en Pascal et C qui peuvent être utilisées pour appeler des programmes et se servent de la fonction EXEC sur le plan interne. En tant que programmeur en langage évolué, tenez-vous donc prêt à travailler avec les blocs d'environnement, les lignes de commande et la restauration des registres de processeurs. Les pages suivantes décrivent ces diverses notions.

23.1. Charger et lancer des programmes

Contrairement à la plupart des fonctions DOS toutes simples servant uniquement à sortir les caractères ou à lire un fichier, la fonction EXEC s'entourne d'une riche philosophie qu'il convient d'expliciter immédiatement.

La philosophie de la fonction EXEC

Etant donné que la fonction EXEC fait toujours intervenir deux programmes, on rencontre ici la terminologie des programmes père et enfant issue de l'anglais Father et Child. Le programme père représente le programme d'appel utilisant la fonction EXEC. Le programme d'appel s'appelle au contraire programme enfant lorsqu'il hérite d'une ou plusieurs routines similaires à celle de son père.

Sachant que DOS n'est pas un système multi-tâches et ne peut exécuter simultanément qu'un seul programme, l'exécution du programme père dépend donc d'un appel correct de la fonction EXEC. D'une manière très générale, cette fonction permet à un programme appelé père d'appeler un programme appelé enfant. Cet enfant sera chargé dans la mémoire à partir d'une mémoire de masse puis exécuté. Si ce programme ne s'installe pas lui-même comme résident, la place mémoire qu'il occupait est à nouveau libérée après son exécution. Le programme-enfant peut à son tour appeler un autre programme à l'aide de cette fonction, pour lequel il fera donc office de père. Ainsi peut se former une sorte de chaîne dont la longueur n'est limitée que par la taille de la mémoire RAM disponible.

L'interpréteur de commandes constitue un bon exemple d'application de cette fonction EXEC puisqu'il fait justement exécuter par celle-ci les programmes spécifiés par l'utilisateur, pour lesquels il représente le programme père. Certains programmes

(comme le WORD de Microsoft par exemple) permettent à l'utilisateur de faire exécuter des instructions du DOS à partir du programme, ce qui est également réalisé à l'aide de cette fonction.

Le programme-père peut aussi transmettre certains paramètres au programme-enfant. Cela peut être réalisé en transmettant ces paramètres dans la ligne d'instruction ou bien à l'aide du bloc d'environnement. Une troisième possibilité de transmission de paramètres consiste à transmettre au programme-enfant des informations à l'intérieur du PSP. Le programme-enfant est en effet précédé dans la mémoire d'un PSP, comme tous les autres programmes exécutables, et il est donc possible d'inscrire des informations dans les deux FCB figurant dans ce PSP. Ces informations seront ainsi rendues accessibles au programme-enfant. Nous reviendrons plus en détail sur ces trois possibilités dans le cadre de ce chapitre.

Hiérarchie du mode d'exploitation

Une fois que le contrôle a été transmis au programme-enfant, ce dernier peut accéder à tous les fichiers et périphériques qui ont été ouverts dans le programme-père ou plutôt dans un des programmes-pères à l'aide d'une fonction handle. Un programme-enfant peut ainsi lire des informations ou bien en écrire dans un fichier dont il ne connaît même pas le nom, pourvu qu'il en connaisse le handle. Or il ne peut connaître ce handle que si ce numéro lui a été communiqué auparavant par le programme-père de l'une des trois façons que nous avons décrites, à moins que le programme-enfant ne se réfère à l'un des 5 handles ouverts en permanence. Le pointeur de fichier est naturellement modifié par ces accès du programme-enfant. Cependant la valeur que revêtait ce pointeur de fichier lors de l'appel du programme-enfant n'est pas rétablie lorsque la main est rendue au programme-père et les accès au fichier sont donc aussi "visibles" pour ce dernier.

Transmission de codes d'erreur

Après exécution du programme-enfant, le contrôle est rendu au programme-père dont l'exécution se poursuit. Le programme-enfant peut transmettre une valeur numérique à la fin de son exécution pour communiquer certaines informations (notamment l'apparition d'une erreur au cours de l'exécution du programme-enfant) au programme-père. Cela peut être réalisé en appelant la fonction 4Ch qui permet de terminer un programme en transmettant en même temps un code au programme-père. Contrairement à l'appel de programme habituel, il ne s'agit pas du processeur de commande COMMAND.COM, mais la valeur de retour parvient toutefois au programme-père.

La communication entre les programmes père et enfant ne peut naturellement fonctionner que si tous deux parlent le même langage, c'est-à-dire s'ils ont la même compréhension de la signification de cette valeur. Une fois que le contrôle a été rendu

au programme-père, ce dernier peut tester le code transmis à l'aide de la fonction 4Dh de l'interruption 21h. Pour appeler cette fonction, il suffit de placer le numéro de la fonction dans le registre AH. Après retour de la fonction, le registre AL contiendra alors le code de retour transmis par le programme-enfant.

Le contenu du registre AH indiquera en outre de quelle façon le programme-enfant a été quitté. La valeur 0 signifie une fin normale, 1 indique que le programme-enfant a été treminé en actionnant la touche Control-C ou la touche Control-Break. Si une erreur est apparue au cours de l'exécution du programme-enfant à l'occasion d'un accès à une mémoire de masse et a entraîné l'arrêt du programme-enfant, le code 2 est transmis. La valeur 3 signale enfin que le programme-enfant a été terminé par un appel de la fonction 31h ou de l'interruption 27h et qu'il s'est donc installé lui-même comme résident.

Libération de la mémoire RAM

Comme nous l'avons indiqué plus haut, la fonction EXEC ne peut charger le programme-enfant que si la mémoire RAM disponible est suffisante. Avec les programmes EXE, le DOS parvient assez bien à évaluer la mémoire nécessaire. Par contre, avec les programmes COM, cela n'est pas possible. C'est pourquoi le DOS réserve pour tout programme COM la totalité de la mémoire RAM non encore occupée. En conséquence, un programme COM ne peut appeler un autre programme à l'aide de la fonction EXEC puisqu'il n'y a plus de place mémoire disponible. Il en va toutefois de même pour bon nombre de programmes EXE, bien que pour des raisons différentes.

Si l'on veut malgré tout avoir recours à cette méthode d'appel d'un programme-enfant, il est nécessaire de libérer à nouveau la place mémoire qui n'est pas utilisée avant d'appeler la fonction EXEC.

Appel de la fonction EXEC

Pour appeler la fonction EXEC, les différents paramètres doivent être chargés dans les registres appropriés avant appel de l'interruption 21h. Le numéro de fonction doit être transmis comme d'habitude à travers le registre AH. Le registre AL doit recevoir la valeur 0 ou 3 suivant le mode d'appel de la fonction EXEC. 0 indique en effet à la fonction EXEC que le programme spécifié doit être chargé puis immédiatement exécuté. Si c'est au contraire la valeur 3 qui est transmise dans le registre AL, la fonction EXEC chargera le programme indiqué simplement comme un overlay, en le plaçant dans un emplacement du programme père défini d'avance (nous y reviendrons à la fin de cette section). L'adresse de deux paramètres est attendue dans les paires de registres DS:DX et ES:BX. Le premier paramètre représente le nom du programme à exécuter ou à charger, nom qui doit figurer dans la mémoire sous forme d'une chaîne de caractères ASCII et qui doit être terminé par une marque de fin (code ASCII 0). Le second

paramètre est ce qu'on appelle un bloc de paramètres, dont la structure est définie par convention et qui contient toute une série d'informations.

Le nom du programme peut comprendre une désignation de périphérique ainsi qu'une spécification de chemin complète. Son dernier élément est le nom du programme qui doit comporter le nom proprement dit ainsi que l'extension ".COM" ou ".EXE". Si la désignation de périphérique ou la spécification de chemin sont omises, le programme sera recherché dans le répertoire actuel du périphérique actuel. Comme la fonction EXEC ne peut exécuter directement un fichier batch, le nom de programme indiqué ne peut comporter l'extension ".BAT".

Si l'on veut faire exécuter un fichier batch, on doit indiquer comme nom de programme le fichier COMMAND.COM, c'est-à-dire l'interpréteur de commandes. Pour indiquer à ce dernier qu'il devra exécuter un fichier batch, il faut lui transmettre dans la ligne d'instruction l'option /C suivie du nom du fichier batch à exécuter. L'appel de l'interpréteur de commandes avec l'option /C permet d'ailleurs non seulement de faire exécuter un fichier batch mais aussi de faire exécuter tout autre programme, y compris des instructions internes du DOS telles que DIR par exemple.

Contrairement à l'appel direct d'un programme, l'appel de l'interpréteur de commandes permet d'indiquer le nom du programme sans extension. L'interpréteur de commandes cherchera en effet tout d'abord, dans ce cas, un fichier correspondant avec l'extension EXE, puis avec COM et enfin avec BAT.

S'il ne trouve pas le fichier dans le répertoire indiqué, il cherchera également dans tous les répertoires définis avec l'instruction PATH, ce qui n'est pas non plus le cas lorsqu'un programme est appelé directement sans passer par l'interpréteur de commandes.

Si vous voulez donner la possibilité à l'utilisateur, à l'intérieur d'un programme, d'appeler n'importe quelle instruction du DOS, le seul moyen consistera également à appeler l'interpréteur de commandes avec l'option /C et l'instruction à appeler. Encore faut-il naturellement que vous sachiez dans quel répertoire figure l'interpréteur de commandes. Si votre PC dispose d'un disque dur, vous pouvez partir du principe qu'il figure dans le répertoire racine du disque dur. Sa situation précise est d'ailleurs également enregistrée dans le bloc d'environnement, où l'indication COMSPEC= est suivie de la spécification de chemin exacte, y compris le nom de l'interpréteur de commandes.

Il est naturellement possible de transmettre des paramètres non seulement à l'interpréteur de commandes mais aussi à l'instruction appelée, à la suite du nom du programme. L'interpréteur de commandes est en effet un programme tout à fait ordinaire pour la fonction EXEC. Ce ou ces paramètres sont identiques à ceux qui vous indiquez lorsque vous appelez certains programmes au clavier.

Le bloc de paramètres de la fonction EXEC

Nous allons donc examiner de plus près comment transmettre ces paramètres à la fonction EXEC. Voyons cependant tout d'abord comment se présente la structure du bloc de paramètres (lorsque le registre AL contient la valeur 0) dont l'adresse doit être transmise à la fonction EXEC dans la paire de registres ES:BX.

Structure du bloc de paramètres pour l'appel de la fonction EXEC		
Adresse	**Contenu**	**Type**
ES:BX	Adresse de segment du bloc d'environnement	1 WORD
+02h	Adresse de la ligne de commande	1 PTR
+06h	Adresse du premier FCB	1 PTR
+0Ah	Adresse du second FCB	1 PTR
Longueur : 14 octets		

Le champ 1 indique l'adresse de segment du bloc d'environnement du programme-enfant. Nul n'est besoin d'une adresse d'offset pour ce bloc puisqu'il doit toujours commencer à une adresse divisible par 16. Cette adresse peut donc être spécifiée avec une adresse d'offset valant toujours 0.

Bloc d'environnement

Il s'agit d'un groupe de chaînes de caractères ASCII dans lesquelles l'interpréteur de commandes et d'autres programmes peuvent puiser certaines informations comme par exemple le chemin (PATH) dans lequel doivent être recherchés les fichiers. Chacune de ces chaînes de caractères présente normalement la forme suivante :

```
Nom = Paramètre
```

et se termine par une marque de fin (code ASCII 0). Les différentes chaînes sont placées immédiatement à la suite l'une de l'autre. La marque de fin d'une chaîne doit donc être suivie immédiatement du premier caractère de la chaîne suivante. La fin du bloc d'environnement, dont la longueur ne doit pas excéder 32 Ko, est signalée par une marque de fin à la suite de la marque de fin de la dernière chaîne.

A partir de la version 3.0 du DOS, la dernière chaîne d'environnement peut encore être suivie de chaînes supplémentaires dont le nombre est indiqué dans un mot placé à la suite de la dernière chaîne d'environnement. Jusqu'à présent, cependant, le DOS ajoute ici seulement une chaîne supplémentaire (le mot à la suite de la dernière chaîne d'environnement porte donc la valeur 1), qui contient la désignation du périphérique et le nom de chemin complet du programme dont relève le bloc d'environnement. A l'aide

des informations ainsi fournies par le bloc d'environnement étendu, un programme peut par exemple déterminer à partir de quel répertoire il a été chargé, pour charger ensuite les overlays ou autres fichiers importants directement à partir de ce répertoire, sans avoir à demander à l'utilisateur d'indiquer ce répertoire.

La figure suivante montre l'environnement du programme DEBUG.COM après qu'il ait été lancé en mémoire.

```
1DFE:0000  43 4F 4D 53 50 45 43 3D-43 3A 5C 43 4F 4D 4D 41   COMSPEC=C:\COMMA
1DFE:0010  4E 44 2E 43 4F 4D 00 50-41 54 48 3D 43 3A 5C 3B   ND.COM.PATH=C:\;
1DFE:0020  43 3A 5C 44 4F 53 3B 43-3A 5C 42 41 54 43 48 45   C:\DOS;C:\FONRE
1DFE:0030  53 3B 45 3A 5C 3B 44 3A-5C 4D 53 43 5C 42 49 4E   V;E:\;D:\MSC\BIN
1DFE:0040  00 49 4E 43 4C 55 44 45-3D 64 3A 5C 6D 73 63 5C   .INCLUDE=d:\msc\
1DFE:0050  69 6E 63 6C 75 64 65 00-4C 49 42 3D 64 3A 5C 6D   include.LIB=d:\m
1DFE:0060  73 63 5C 6C 69 62 00 54-4D 50 3D 64 3A 5C 6D 73   sc\lib.TMP=d:\ms
1DFE:0070  63 5C 74 6D 70 00 50 52-4F 4D 50 54 3D 5B 24 70   c\tmp.PROMPT=[$p
1DFE:0080  5D 00 00 01 00 43 3A 5C-44 4F 53 5C 44 45 42 55   ]....C:\DOS\DEBU
1DFE:0090  47 2E 43 4F 4D 00 00 00-00 00 00 00 00 00 00 00   G.COM...........
```

A partir du niveau utilisateur, le bloc d'environnement peut être manipulé à l'aide des instructions SET et PATH du DOS. Les programmes qui sont chargés et restent résidents après leur exécution ne participent cependant en aucune manière à la modification du bloc d'environnement par ces deux instructions du DOS.

Si le programme-père veut transmettre certaines informations au programme-enfant à l'aide du bloc d'environnement, il peut mettre en place un nouveau bloc d'environnement ou ajouter ces informations dans son bloc d'environnement. Dans le premier cas, l'adresse de segment du nouveau bloc d'environnement devra alors être inscrite dans le premier champ du bloc de paramètres. Si le programme-enfant doit toutefois disposer du même bloc d'environnement que le programme-père, il suffit d'inscrire la valeur 0 dans ce champ. Avant que le contrôle ne soit transmis au programme-enfant, l'adresse de segment du bloc d'environnement sera copiée dans la cellule de mémoire figurant à l'adresse 2Ch du PSP-enfant.

Si un nouveau bloc d'environnement est transmis au programme-enfant, il doit contenir au moins 3 chaînes de caractères qui font (normalement) partie du bloc d'environnement du programme-père et qui sont très importantes pour l'interpréteur de commandes :

```
COMSPEC = Paramètre
PATH = Paramètre
PROMPT = Paramètre
```

Transmission des paramètres de commande

Si un programme-enfant modifie son bloc d'environnement, ces modifications ne sont pas transmises au bloc d'environnement du programme-père une fois terminée l'exécution du programme-enfant.

Le second champ indique l'adresse des paramètres d'instruction qui seront copiés dans le PSP du programme à exécuter à partir de l'adresse 80h. Ils doivent présenter dans la mémoire la même structure que celle attendue par le DOS dans le PSP.

Le premier octet indique donc le nombre de caractères d'instruction moins 1. Viennent ensuite les caractères d'instruction eux-mêmes, sous forme de codes ASCII normaux. Les paramètres d'instruction se terminent par un caractère de retour de chariot (code ASCII 13).

Nous évoquions plus haut la possibilité de faire exécuter un programme batch à l'aide de l'interpréteur de commandes (COMMAND.COM). Pour poursuivre avec cet exemple, voici comment devraient se présenter dans la mémoire les paramètres d'instruction pour un programme batch que nous appellerons simplement DO.BAT :

```
DB 9,"/C DO.BAT",13
```

La fonction EXEC ne copie cependant pas les paramètres d'instruction de façon incontrôlée dans le PSP. Elle élimine tout d'abord tous les paramètres qui entraîneraient un détournement de l'entrée ou de la sortie car un détournement des entrée et sortie standard ne peut être réalisé que par le programme-père. Le programme-enfant peut malgré tout tirer parti du détournement des entrée et sortie en ayant recours aux 5 handles prédéfinis pour les entrée et sortie standard.

Transmission du FCB à l'aide du bloc de paramètres

Les champs 3 et 4 se réfèrent aux deux FCB qui ont été mis en place dans le PSP à l'adresse 5Ch ou 6Ch. Si vous ne voulez pas employer cette possibilité de transmettre des informations à travers les deux FCB, vous devez inscrire -1 (FFFFh) dans chacun de ces champs. Si vous tenez cependant à émuler exactement l'opération d'appel d'un programme à partir du clavier, vous devez inscrire les deux premiers paramètres d'instruction dans les deux FCB à l'aide de la fonction 29h du DOS. Avant de trans-mettre le contrôle au programme-enfant, la fonction EXEC copiera alors ces deux FCB dans le PSP du programme-enfant.

Sauvegarde des registres

Bien que tous les registres ainsi que le bloc de paramètres aient maintenant reçu les valeurs nécessaires, nous ne pouvons encore appeler la fonction EXEC. Elle détruit en effet lors de son exécution le contenu de tous les registres à l'exception des registres CS et IP. Il nous faut donc sauver sur la pile le contenu de tous les registres. Il faut ensuite sauvegarder le contenu des registres SS et SP à l'intérieur du segment de code.

Ce n'est qu'alors que la fonction EXEC pourra être activée en appelant l'interruption 21h. Une fois terminée l'exécution de la fonction EXEC, le flag Carry indique comme

d'habitude si la fonction a été exécutée sans encombre. Le registre AL contient en outre le code d'erreur 2 "Fichier non trouvé" qui est le plus utilisé. Il signale que EXEC n'a pas trouvé le programme spécifié. Reportez-vous à l'annexe 59h pour obtenir la liste des autres codes d'erreur.

Avant que l'exécution du programme ne puisse se poursuivre, il convient toutefois d'abord de restaurer le contenu des registres SS et SP en les lisant dans le segment de code. On peut ensuite aller rechercher sur la pile le contenu des autres registres.

23.2. La fonction EXEC pour charger les overlays

La fonction EXEC acquiert une fonction toute différente lorsqu'elle est appelée avec la valeur 3 dans le registre AL. Sa tâche consiste en effet uniquement, dans ce cas, à charger un programme .COM ou .EXE dans la mémoire, mais sans l'exécuter. C'est pourquoi le contrôle est rendu au programme d'appel immédiatement après chargement du programme. Le programme d'appel peut alors appeler le programme chargé chaque fois que nécessaire. Contrairement à la sous-fonction 0, cette sous-fonction ne charge pas le programme appelé n'importe où mais à l'adresse prescrite par le programme d'appel.

Le bloc de paramètres destiné à charger les overlays

Comme aucun paramètre n'est transmis au programme chargé (à quoi cela servirait-il en effet ?), la structure du bloc de paramètres est avec la sous-fonction 3 très différente de ce qu'elle était pour l'appel de la sous-fonction 0 :

Structure du bloc de paramètres pour l'appel de la fonction Overlay (sous-fonction 03h de la fonction EXEC)		
Adresse	**Contenu**	**Type**
ES:BX	Adresse de segment à laquelle est chargé	1 WORD
+02h	Facteur de relogement	1 WORD
Longueur : 4 octets		

Avant d'appeler la fonction, il faut inscrire dans le premier champ du bloc de paramètres l'adresse de segment à laquelle doit être chargé le programme voulu. Si le programme d'appel n'offre pas suffisamment de place mémoire pour charger le programme, il devra réclamer de la place mémoire à l'aide d'une des fonctions du DOS pour la gestion de la mémoire puis mettre la place mémoire ainsi obtenue à la disposition du programme appelé. Le programme chargé n'est toutefois pas précédé d'un PSP, de sorte qu'il est chargé directement à l'adresse de segment indiquée avec une adresse d'offset de 0.

Relogement d'overlays

Le facteur de relogement sert à adapter les adresses de segment du programme appelé. Cela n'a toutefois d'intérêt que pour les programmes EXE (puisque les programmes COM ne doivent pas comporter de définitions de segment explicites) et ce facteur de relogement devra donc toujours être fixé sur 0 pour les programmes COM. Pour les programmes EXE, par contre, il devra indiquer l'adresse de segment à laquelle le programme est chargé pour que les définitions de segment dans le programme EXE puissent être adaptées à cette adresse de segment.

Une fois le programme voulu chargé, encore faut-il bien sûr l'appeler pour que ses fonctions soutiennent le programme d'appel. Les routines du programme chargé doivent toujours être considérées comme des sous-programmes. Elles doivent donc être appelées à l'aide de l'instruction CALL (en langage machine). Il faut toujours utiliser un FAR-Call car le programme chargé est placé immédiatement à la suite du programme d'appel mais il ne peut jamais avoir la même adresse de segment.

Problèmes lors du chargement des overlays COM

Pour un programme COM, l'adresse d'offset pour cette instruction CALL sera toujours 100h puisque l'exécution des programmes COM commence toujours, par définition, immédiatement à la suite du PSP, c'est-à-dire à l'adresse 100h. Cela pose cependant un problème étant donné que la sous-fonction 3 ne charge pas le PSP. Le code d'un programme COM ne commencera donc pas à l'adresse d'offset 100h par rapport au segment de chargement mais bien à l'adresse 0. Or toutes les instructions de saut et tous les accès de données à l'intérieur du programme COM se réfèrent à l'adresse 100h et non à l'adresse 0. Il ne suffit donc pas de faire exécuter un FAR-CALL avec l'adresse du segment de chargement comme adresse de segment et 0 comme adresse d'offset. C'est l'adresse de segment elle-même qui doit être rectifiée. On indiquera donc l'adresse de chargement moins 10h comme adresse de segment pour le FAR-CALL et l'adresse 100h comme adresse d'offset.

Si un programme COM a été développé spécialement comme un overlay pour un autre programme, d'autres adresses d'entrée que l'adresse 100h sont naturellement possibles également. Il suffit de spécifier dans le FAR-CALL l'adresse d'offset voulue mais l'adresse de segment devra tout de même être inférieure de 10h à l'adresse du segment de chargement.

Exécution d'overlays EXEC

Le problème se présente différemment pour les programmes EXE. S'ils sont chargés pour être exécutés (à travers la sous-fonction 0), la fonction EXEC fixe le segment de code et le pointeur d'instruction sur l'instruction qui a été déclarée dans le source

assembleur comme la première instruction. Cette adresse est toutefois inconnue du programme d'appel, c'est-à-dire du programme qui charge le programme EXE comme overlay. On peut toutefois aisément y remédier en plaçant tout simplement au début du programme EXE, c'est-à-dire à l'adresse 0, la première instruction à exécuter dans ce programme. Cela suppose cependant que le source assembleur ne présente pas la structure habituelle avec en premier le segment de pile, puis le segment de données et enfin seulement le segment de code dont la première instruction constitue l'instruction de lancement du programme. Il convient donc au contraire, dans ce cas, que le segment de code soit le premier segment dans le source pour qu'il figure en tout état de cause au début du programme EXE.

Le FAR-CALL utilise alors comme adresse de segment l'adresse du segment de chargement et comme adresse d'offset l'adresse 0.

23.3. Nouveautés du DOS 5.0

Outre les deux sous-fonctions 00h et 03h, la fonction EXEC contient une autre sous-fonction portant le numéro 05h depuis la version DOS 5.0. Elle doit être utilisée par tous les programmes qui chargent, par leurs propres soins, des programmes et overlays dans la mémoire. Ils ne se servent pas alors des sous-fonctions 00h ou 03h mais ouvrent ces programmes comme un fichier normal pour charger ensuite leur contenu dans la RAM.

La nouvelle sous-fonction 05h n'est toutefois pas conçue pour charger ces programmes mais pour lancer leur exécution. Cette opération s'effectue certes parfaitement sans l'aide de la sous-fonction 05h, mais dans ce cas, DOS ne peut pas se permettre de réaliser les préparatifs nécessaires avant le lancement d'un programme. Ainsi, DOS active la ligne d'adresse A20 au cas où une partie du code de programme se trouve dans la HMA pour autoriser au programme l'accès à cette cellule de mémoire.

Il rectifie en outre son numéro de version lorsque le programme SETVER a attribué un numéro de version DOS différent de 5.0 au programme à exécuter. Pour parfaire l'opération, toutes sortes de paths sont examinés à l'intérieur du programme. Ils s'avèrent en effet nécessaires si le programme est chargé au-dessous des premiers 64 Ko de la RAM. Comme les versions précédentes de DOS ne prévoyaient pas cela, de nombreux programmes ne sont pas en mesure de réaliser cette action et ne fonctionnent pas alors convenablement.

Il est donc utile de mettre DOS 5.0 au courant de l'exécution d'un tel programme ou overlay. A cet effet, la sous-fonction 05h doit transmettre un pointeur sur la structure de données dans la paire de registres DS:DX outre le numéro de fonction dans le registre AX (4B05h). La figure suivante décrit cette structure. Grâce à cette dernière, DOS obtient toutes les informations nécessaires pour exécuter les opérations évoquées plus haut.

Structure de ExecState		
Adresse	**Contenu**	**Type**
00h	Réservé, doit contenir 0	1 WORD
02h	1 = Programme EXE 2 = Overlay	1 WORD
04h	Pointeur sur une chaîne ASCIIZ contenant le nom du programme ou overlay (il est possible d'indiquer les chemins dans cette chaîne)	1 PTR
08h	Adresse de segment du PSP du programme ou overlay	1 WORD
0Ah	Point d'entrée dans le programme ou overlay	1 PTR
0Eh	Taille du programme ou overlay y compris PSP	1DWORD

Après appel de cette fonction, pensez à vérifier que le flag Carry donne les informations nécessaires concernant la bonne exécution de l'appel. Si tel est le cas, l'exécution du programme ou overlay peut être envisagée. Celle-ci doit avoir obligatoirement lieu car ni les fonctions DOS ou BIOS ni toute autre interruption logicielle ne peuvent être appelées avant l'introduction dans le programme/overlay.

23.4. Exemple de programme

Alors que les langages évolués comme le BASIC, le Pascal ou le C prévoient des instructions spéciales pour appeler un programme à partir d'un autre programme, la fonction 4Bh doit être utilisée directement en assembleur. Nous allons donc vous présenter un exemple de programme pour cette fonction, de façon à faciliter quelque peu l'appel assez complexe de cette fonction.

C'est la procédure EXEPRG qui se charge du travail véritable dans ce programme. Elle sert à appeler le programme voulu à travers la fonction 4Bh. Elle attend seulement deux chaînes dont la première contienne le nom du programme à appeler et la seconde les paramètres à transmettre. Ces deux chaînes doivent être terminées par une marque de fin (code ASCII 0). Toutes les variables dont EXEPRG a besoin pour son travail sont contenues dans le segment de code. L'avantage est pour vous qu'il vous suffit de reprendre dans un de vos programmes les lignes du segment de code pour pouvoir utiliser cette routine. Après appel de EXEPRG, le flag Carry indique si une erreur est apparue. Si c'est le cas (flag Carry = 1), le registre AX contient le code d'erreur renvoyé par la fonction EXEC du DOS. Si par contre le programme à appeler a pu être exécuté sans problème, le flag Carry est annulé et le code de fin du programme appelé, tel que renvoyé par la fonction 4Dh du DOS, figure dans le registre AX.

Dans notre programme d'exemple, EXEPRG est utilisé pour faire afficher le répertoire actuel à l'aide de l'interpréteur de commandes. Il faut que ce dernier figure dans le répertoire racine du périphérique actuel.

Listing : EXEC.ASM

```
;*****************************************************************;
;*                            E X E C                            *;
;*---------------------------------------------------------------*;
;*   Fonction      : Appelle un programme quelconque à l'aide de la *;
;*                   fonction EXEC du DOS. Ce programme d'exemple  *;
;*                   affiche le contenu du répertoire racine du    *;
;*                   périphérique actuel.                          *;
;*                                                                 *;
;*---------------------------------------------------------------*;
;*   Auteur        : MICHAEL TISCHER                               *;
;*   Développé le  :  1.08.87                                      *;
;*   Dernière modif. : 19.03.92                                    *;
;*---------------------------------------------------------------*;
;*   Pour assembler  : MASM EXEC                                   *;
;*                     LINK EXEC                                   *;
;*             ou                                                  *;
;*                     TASM EXEC                                   *;
;*                     TLINK EXEC                                  *;
;*---------------------------------------------------------------*;
;*   Appel         : EXEC                                          *;
;*****************************************************************;

;-- Données ------------------------------------------------------

!data    segment para 'DATA'          ;Définition du segment de données

!prgname db "c:\command.com",0              ;Nom du programme à appeler
!prgpara db "/c dir",0              ;Les paramètres sont transmis au Prg

!        ;-- Les divers messages du programme ---------------------

!startmes db " EXEC - (c) 1987, 92 by Michael Tischer "
!        db 13,10,"$"
!mesok   db "OK",13,10,"$"
!mesnotf db "Erreur : COMMAND.COM non trouvé!",13,10,"$"
!meselse db "Erreur : Code d'erreur DOS = "
!mescode db "   ",13,10,"$"

!data    ends                                ;Fin du segment de données

;-- Code ---------------------------------------------------------

!code    segment para 'CODE'          ;Définition du code de segment

!        assume cs:code, ds:data, ss:stackseg

!exec            proc far

!        mov  ax,data              ;Charge adresse segment du DATA seg.
!        mov  ds,ax                          ;dans le registre DS

!        mov  ah,09h                    ;Afficher le message d'invite
!        mov  dx,offset startmes
!        int  21h

!        call setfree                   ;Libérer la mémoire inutilisée

!        mov  dx,offset prgname              ;Offset nom de programme
!        mov  si,offset prgpara         ;Offset de ligne d'instruction
!        call exeprg                           ;Appeler le programme

!        mov  dx,offset mesok       ;Partir d'une exécution non erronée
!        jnc  ex_mes                ;Pas d'erreur --> Afficher message

!        mov  dx,offset mesnotf      ;Apparemment, programme non trouvé
!        cmp  al,2                                   ;Pas vrai?
!        je   ex_mes                          ;Oui ---> Afficher message

!        xor  ah,ah                        ;Hi-Byte du code d'erreur
!        mov  si,offset mescode+2          ;Code d'erreur vers ASCII
!        call toint

!        mov  dx,offset meselse
!ex_mes: mov  ah,09h                           ;Afficher chaîne via DOS
!        int  21h

!        mov  ax,4C00h                ; FIN programme par fonction DOS
!        int  21h                   ;en transmettant le code d'erreur 0

!exec            endp

!;-- SETFREE: Libérer la place mémoire inutilisée -----------------
!;-- Entrée  : ES = Adresse du PSP
!;-- Sortie  : Aucune
!;-- Registres : AX, BX, CL et FLAGS sont modifiés
!;-- Infos   : Comme le segment de pile est toujours le dernier segment
!;--           dans un fichier EXE, ES:0000 désigne le début et SS:SP la
!;--           fin du programme en mémoire. C'est ainsi qu'on peut
!;--           calculer la longueur du programme

!setfree         proc near

!        mov  bx,ss       ;Calculer d'abord la différence entre les deux
!        mov  ax,es        ;adresses de segment. Cela donne le nombre de
!        sub  bx,ax              ;paragraphes du PSP au début de la pile

!        mov  ax,sp       ;Comme le pointeur de pile se trouve à la fin
!        add  ax,15             ;du segment de pile, son contenu indique
!        mov  cl,4                              ;la longueur de la pile
!        shr  ax,cl
!        add  bx,ax                    ;Ajouter à la longueur antérieure

!        mov  ah,4ah              ;Transmettre la nouvelle taille au DOS
!        int  21h

!        ret                                          ;Retour à l'appelant

!setfree         endp

!;-- EXEPRG: Appeler un autre programme ---------------------------
!;-- Entrée  : DS:DX = Adresse du nom de programme
!;--           DS:SI = Adresse de la ligne d'instruction
!;-- Sortie  : Carry-Flag = 1 : Erreur (AX = code d'erreur)
!;-- Registres : Seuls AX et FLAGS sont modifiés
!;-- Infos   : Le nom de programme et la ligne d'instruction doivent
!;--           figurer sous forme de chaînes ASCII terminées par le
!;--           code ASCII 0

!exeprg          proc near

!        ;-- Transmettre la ligne d'instruction spécifiée ----------
!        ;-- dans un buffer en comptant les caractères --------------

!        push bx                              ;Sauvegarder tous les registres
!        push cx                                 ;détruits par l'appel d'une
!        push dx                                          ;fonction EXEC DOS
!        push di
!        push si
!        push bp
!        push ds
!        push es

!        mov  di,offset comline+1          ;Car. dans ligne d'instruction
!        push cs                                          ;CS sur la pile
!        pop  es                                  ;Retirer pour placer dans ES
!        xor  bl,bl                       ;Fixer le compteur de carac. sur 0
!copypara: lodsb                                         ;Lire un caractère
!        or   al,al                                  ;Est-ce le code NUL (Fin)
!        je   fincopie                                    ;OUI --> assez copié
!        stosb                            ;Sauvegarder dans le nouveau buffer
!        inc  bl                         ;Augmenter le compteur de caractères
!        cmp  bl,126                                      ;Maximum atteint?
!        jne  copypara                                    ;NON --> continuer

!fincopie: mov  cs:comline,bl          ;Sauvegarder le nombre de caractères
!        mov  byte ptr es:[di],13           ;Terminer ligne d'instruction

!        mov  cs:rangss,ss         ;SS et SP doivent être stockés dans le
!        mov  cs:rangsp,sp                              ;segment de code

!        mov  bx,offset parblock   ;ES:BX désigne le bloc de paramètres
!        mov  ax,4B00h          ;Numéro de fonction pour la fonction EXEC
!        int  21h                              ;Appeler fonction du DOS

!        cli                  ;Désactiver temporairement les interruptions
!        mov  ss,cs:rangss         ;Rétablir anciennes valeurs segment de
!        mov  sp,cs:rangsp                    ;pile et du pointeur de pile
!        sti                           ;Activer à nouveau les interruptions

!        pop  es                                  ;Retirer à nouveau tous les
!        pop  ds                                      ;registres de la pile
!        pop  bp
!        pop  si
!        pop  di
!        pop  dx
!        pop  cx
!        pop  bx

!        jc   exeend                                  ;Erreur? OUI --> Fin
!        mov  ah,4dh              ;Pas d'erreur, Exécuter le code de fin
!        int  21h                                         ;du programme

!exeend:  ret                                          ;Retour à l'appelant

!        ;-- Ces variables ne peuvent être appelés qu'à travers CS ----
```

```
;
rangss    dw (?)                ;Reçoit SS pendant l'appel du programme
rangsp    dw (?)                ;Reçoit SP pendant l'appel du programme
;
parblock  equ this word         ;Bloc de paramètres pour la fonction EXEC
          dw 0                              ;Même bloc d'environnement
          dw offset comline ;Adresses d'offset et de segment de la ligne
          dw seg code                                        ;convertie
          dd 0                              ;Pas de données dans PSP @1
          dd 0                              ;Pas de données dans PSP @2
;
comline   db 128 dup (?)        ;Reçoit la ligne d'instruction convertie
;
exeprg          endp
;
;-- TOINT: Convertit un nombre binaire en ASCII et le place ------------
;--  dans un buffer de l'appelant en le justifiant à droite
;-- Entrée : DS:SI = Ptr sur la position où doit apparaître le chiffre
;--                  de poids faible
;--          AX    = Nombre binaire à convertir
;-- Sortie : DS:SI = Pointeur sur le chiffre de poids fort
;-- Registres : AX, BX, DX, SI et FLAGS sont modifiés
;-- Infos : - Le buffer doit contenir de la place pour au moins cinq
;--           caractères.
;--         - Le nombre entré dans le buffer est justifié à droite.
;
toint           proc near
;
          ;-- Diviser toujours par 10 le nombre dans la boucle et ------
          ;-- convertir ainsi le LSB au format ASCII -------------------
          ;-- puis l'entrer dans le buffer -----------------------------
;
          mov  bx,10                        ;Le diviseur est toujours 10
          jmp  short ti2
;
ti1:      dec  si                                   ;SI sur caractère suivant
;
ti2:      xor  dx,dx                                 ;Le dividende est DX:AX
          div  bx                                      ;Diviser DX:AX par 10
          or   dl,'0'                          ;Convertir DL au format ASCII
          mov  [si],dl                               ;Placer dans le buffer
          or   ax,ax                                    ;Y a-t-il un reste?
          jne  ti1                                  ;OUI --> Chiffre suivant
;
          ret                                            ;Retour à l'appelant
;
toint           endp
;
;-- Stack ---------------------------------------------------------------
;
          ;--- Le segment de pile est à placer ici à la fin du fichier -
          ;--- pour qu'il se trouve derrière le segment de données -----
          ;--- et de code après chargement du programme ----------------
;
stackseg  segment para stack 'STACK'      ;Définition du segment de pile
;
          dw 256 dup (?)             ;La taille de la pile est de 256 mots
;
stackseg  ends                                    ;Fin du segment de pile
;
;-- Fin -----------------------------------------------------------------
;
code      ends                                        ;Fin du segment de code
          end  exec                             ;Commencer l'exécution à EXEC
;
```

24. Interruptions Ctrl-Break et Critical Error

Sous le système d'exploitation DOS, il existe deux situations dans lesquelles un programme peut être terminé pratiquement à n'importe quel moment, du fait de l'intervention d'événements échappant totalement au contrôle du programme. Ces situations surviennent lorsque l'utilisateur actionne la touche <Ctrl>-<Break> ou la touche <Ctrl>-<C> ou bien lorsqu'apparaît, lors de l'accès à un périphérique externe (imprimante, disque dur, lecteur de disquette etc.), un défaut particulièrement grave, qu'on appelle "critical error".

Il se peut naturellement que l'utilisateur, en actionnant la touche <Ctrl>-<Break>, ait bien eu l'intention de mettre fin au programme sans attendre qu'il se termine de façon "régulière". Cependant un arrêt aussi brusque du programme ne va pas sans poser de problème. Le programme se termine en effet simplement par le fait que le DOS reprend le contrôle de l'exécution, de sorte que les instructions normalement exécutées lors de la fin régulière du programme se trouvent superbement ignorées. Il en résulte que les fichiers ouverts ne sont pas refermés comme il conviendrait, que les vecteurs d'interruption redirigés ne sont pas restaurés et que la mémoire allouée n'est pas libérée. Toutes ces conséquences peuvent déboucher sur une perte de données ou même sur un plantage du système (et donc sur une perte de données encore plus importante).

L'interruption Ctrl-Break 23h

Pour éviter cela, le DOS ne met pas systématiquement fin au programme actuellement exécuté lorsque les touches <Ctrl>-<Break> ou <Ctrl>-<C> sont actionnées, mais appelle l'interruption 23h qui, pour cette raison, est appelée interruption Ctrl-Break. Cette interruption est bien branchée, lors du lancement d'un programme, sur une routine du DOS chargée de terminer le programme, mais un programme peut parfaitement rediriger cette interruption sur une routine à lui, de façon à garder le contrôle des événements même lorsque <Ctrl>-<Break> est actionnée.

L'appel de cette interruption ne dépend toutefois pas directement du fait d'actionner l'une des deux combinaisons de touches. Le moment où le DOS teste si l'une des deux combinaisons de touches a été actionnée dépend en effet du flag Break. Ce flag peut être mis ou annulé soit dans le cadre de l'environnement DOS avec l'instruction BREAK (ON/OFF), soit à l'aide de la fonction DOS 33h, sous-fonction 1. Si ce flag est mis, le contenu du buffer clavier sera examiné lors de chaque appel d'une fonction de l'interruption du DOS 21h pour voir si <Ctrl>-<Break> ou <Ctrl>-<C> a été actionnée. Si par contre le flag Break est annulé, ce test ne sera effectué que lors de l'appel des fonctions DOS servant à l'accès aux périphériques d'entrée et de sortie standard.

Si l'une des deux combinaisons de touches est identifiée au cours de ce test, on charge tout d'abord dans les registres du processeur les valeurs qu'ils contenaient lors de l'appel

de la fonction DOS à exécuter. Ensuite seulement intervient l'appel de l'interruption 23h.

Si un programme redirige cette interruption sur une routine à lui, on peut aboutir à différents types de réaction à cet appel. Le programme peut par exemple ouvrir une fenêtre dans laquelle il demandera à l'utilisateur s'il veut sortir du programme. Mais il peut aussi décider de lui-même que le programme doit se terminer ou, au contraire, que le souhait de l'utilisateur de mettre fin au programme doit être purement et simplement ignoré.

Si le programme décide de s'arrêter, il conviendra d'appeler tout d'abord une sorte de fonction de remise en ordre, qui fermera tous les fichiers ouverts, restaurera les vecteurs d'interruption détournés et libérera à nouveau la mémoire allouée. Le programme pourra alors se terminer tout à fait normalement à travers la fonction 4Ch, sans qu'il soit nécessaire de rendre le contrôle à celui qui avait appelé l'interruption 23h.

S'il s'agit au contraire d'ignorer le fait que la touche <Ctrl>-<Break> ou <Ctrl>-<C> ait été actionnée, il suffit de rendre le contrôle au DOS à l'aide de l'instruction IRET du langage machine. Il convient simplement dans ce cas que le programme veille à ce que tous les registres du processeur contiennent bien toujours les valeurs qu'ils contenaient lors de l'appel de l'interruption 23h. Dans le cas contraire, la fonction DOS qui avait été appelée à l'origine ne pourrait en effet être exécutée correctement jusqu'à son terme.

Les deux méthodes seront illustrées à la fin de ce chapitre dans le cadre d'un programme d'exemple.

L'interruption Critical Error 24h

Contrairement à l'interruption Ctrl-Break, l'appel de l'interruption Critical Error ne représente généralement pas une réaction à une intervention délibérée de l'utilisateur, mais bien plutôt à une erreur survenue lors de l'accès à un périphérique externe, tel qu'une imprimante, une unité de disquette ou un disque dur. Si l'utilisateur peut dans certains cas remédier de lui-même à cette erreur (par exemple en cas de "unité de disquette non fermée" ou "imprimante n'est pas en marche"), d'autres erreurs signalent un défaut de fonctionnement de l'électronique et indiquent qu'il est indispensable de réparer le périphérique concerné (par exemple en cas de "Erreur de lecture lors de l'accès au disque dur").

Pour tenir compte des différentes catégories d'erreur, l'interruption Critical Error 24h désigne normalement une routine du DOS qui sort sur l'écran le message bien connu "(A)bort, (R)etry, (I)gnore" ou "(A)bandon, (R)eprise, (E)chec ?" en français et attend une entrée de l'utilisateur. Cette routine présente cependant deux inconvénients. Le premier est que l'écran du programme interrompu est de toute façon détruit et le second est qu'en cas d'arrêt définitif (Abort), le programme ne sera pas terminé "proprement". On

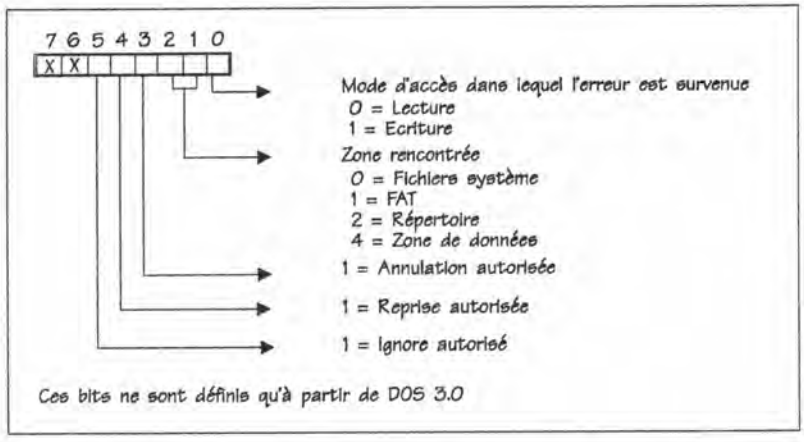
retrouve donc le même problème qu'avec <Ctrl>-<Break>, à savoir que les fichiers ouverts ne seront pas refermés, que la mémoire allouée ne sera pas restituée etc.

La solution consistera ici aussi à installer un gestionnaire d'interruption personnel à l'intérieur du programme, qui sera activé à la place du gestionnaire du DOS lorsque l'interruption Critical Error sera activée. Pour pouvoir localiser la source de l'erreur, le DOS transmet à ce gestionnaire différentes informations, à l'aide des registres du processeur.

Les autres bits du registre AH

Le bit 7 du registre AH indiquera ici s'il s'agit d'une erreur lors de l'accès à la disquette ou au disque dur (bit 7 annulé) ou bien d'une autre erreur (bit 7 mis). Si le bit 7 est supprimé, le numéro du périphérique est retourné en AL où 0 représente le lecteur A, 1 le B, 2 le C et ainsi de suite.

En outre, d'autres informations sont retournées dans le registre AH. Elles concernent la source de l'erreur et les solutions pour y remédier. La figure suivante montre l'aspect de ce registre dans une telle circonstance.



La paire de registres BP:SI désignera en outre l'en-tête du driver de périphérique lors de l'appel duquel l'erreur est survenue. Les 8 bits inférieurs du registre DI contiennent un code d'erreur détaillé, le contenu des 8 bits supérieurs restant indéfini. Les codes d'erreur suivants pourront être renvoyés :

Codes d'erreur transmis au gestionnaire de Critical Error	
Code	**Signification**
00h	Disquette protégée contre l'écriture.
01h	Accès à un périphérique inconnu.
02h	Unité de disquette n'est pas prête.
03h	Erreur inconnue.
04h	Erreur CRC.
05h	Longueur de données erronée.
06h	Erreur de recherche.
07h	Type de périphérique inconnu.
08h	Secteur non trouvé.
09h	Plus de papier dans l'imprimante.
0Ah	Erreur d'écriture.
0Bh	Erreur de lecture.
0Ch	Erreur générale.

Le gestionnaire de Critical Error peut réagir en ouvrant sur l'écran une fenêtre dans laquelle il invite l'utilisateur à choisir entre ignorer l'erreur (c'est-à-dire poursuivre le travail comme si de rien n'était !), répéter la tentative d'accès ou interrompre le programme. Si l'utilisateur opte pour la dernière possibilité, il ne sera toutefois pas possible de terminer proprement le programme, comme pour l'interruption <Ctrl>-<Break>, en refermant les fichiers ouverts et en prenant d'autres mesures nécessaires, car le gestionnaire ne peut appeler que les fonctions DOS 01h à 0Ch. Si d'autres fonctions DOS sont appelées à l'intérieur du gestionnaire de Critical Error, cela n'entraînera cependant pas directement une erreur, mais le retour au DOS qui s'ensuivra provoquera irrémédiablement un plantage du système. Un gestionnaire de ce type ne peut pas non plus terminer le programme à travers la fonction DOS 4Ch, mais doit absolument repasser le contrôle à celui qui l'a appelé à l'aide de l'instruction IRET. Le DOS attend à cette occasion que lui soit transmis dans le registre AL un code indiquant comment il doit réagir à l'erreur, d'après les conventions suivantes :

Codes renvoyés par un gestionnaire de Critical Error	
Code	**Signification**
00h	Ignorer l'erreur.
01h	Répéter l'opération.
02h	Terminer le programme à travers l'interruption 23h.
03h	Terminer la fonction appelée avec une erreur.

Le dernier code d'erreur n'est disponible qu'à partir de la version 3.0 du DOS, mais il représente pourtant la réaction la plus appropriée à une erreur qui n'a pu être éliminée par une répétition de l'accès. (Après par exemple que l'imprimante ait été mise en marche.) Le fait de transmettre ce code d'erreur entraîne une conclusion normale de la fonction lors de l'appel de laquelle l'erreur est survenue, si ce n'est que le flag Carry est mis pour signaler une erreur. Le programme ayant appelé la fonction ne peut cependant distinguer cette erreur "critique" d'une erreur "normale" mais cette distinction peut être rendue possible en fixant un autre flag à l'intérieur du gestionnaire de Critical Error.

Comme pour l'appel de l'interruption Critical Error, le programme peut alors réagir à une erreur de ce type en ouvrant une fenêtre et en demandant à l'utilisateur quelle attitude il doit adopter face à cette erreur. Si l'utilisateur se décide pour un arrêt du programme, il est maintenant possible de le terminer "proprement" car toutes les fonctions DOS, et non seulement les fonctions 01h à 0Ch, peuvent alors être appelées. Cette technique permet donc d'éviter une perte de données ou même le plantage du système lorsqu'une erreur critique est apparue, à moins naturellement que la nature même de cette erreur s'oppose à une sauvegarde des données ou aux autres mesures de remise en ordre.

Alors que les environnements de langages évolués tels que Turbo Pascal et Microsoft C disposent de fonctions pour intercepter les interruptions <Ctrl>-<Break> et Critical Error, le programmeur en assembleur doit avoir recours à des routines de son cru. Le listing assembleur ci-dessous devrait vous aider en cela, puisqu'il constitue l'armature de base pour un programme (EXE) dans lequel les deux gestionnaires sont redirigés sur des routines personnelles. Notez à cet égard que vous n'avez pas à réinstaller les deux gestionnaires une fois le programme terminé car le DOS réactive automatiquement les anciens gestionnaires.

Listing : CEBHAND.ASM

```
;*****************************************************************;
;*                          C E B H A N D                        *;
;*---------------------------------------------------------------*;
;*   Fonction  : Constitue l'outil de base d'un programme        *;
;*               Assembleur dans lequel il s'agit de détourner les *;
;*               interruptions Ctrl Break et Critical-Error du DOS. *;
;*---------------------------------------------------------------*;
;*   Auteur          : MICHAEL TISCHER                           *;
;*   Développé le    :  5.09.1988                                *;
;*   Dernière modif. : 19.03.1992                                *;
;*---------------------------------------------------------------*;
;*   Pour assembler  : MASM CEBHAND;                             *;
;*                     LINK CEBHAND;                             *;
;*               ou                                              *;
;*                     TASM CEBHAND                              *;
;*                     TLINK CEBHAND                             *;
;*---------------------------------------------------------------*;
;*   Appel           : CEBHAND                                   *;
;*                     (Laisser ouvert le lecteur de disquettes A *;
;*                      pour qu'une Critical-Error ait lieu.)    *;
;*****************************************************************;

;-- Constantes ---------------------------------------------------

;-- Stack --------------------------------------------------------

stackseg  segment para stack 'STACK'      ;Définition du segment de pile
          dw 256 dup (?)                  ;La taille de la pile est de 256 mots
stackseg  ends                            ;Fin du segment de pile

;-- Données ------------------------------------------------------

data      segment para 'DATA'             ;Définition du segment de données

cr_err    db 0                            ;vaut 1 si réglé sur 1 lors de l'accès
                                          ;Périphériques (Disquette, Disque dur
                                            ;ou imprimante) une erreur critique
                                                                ;est apparue.
cr_typ    db 0                            ;Numéro d'erreur de l'erreur critique

cr_mes    db "Erreur critique! (A)nnuler ou (R)eprise: $"
next_line db 13,10,"$"
end_mes   db "Programme terminé correctement.$"
brk_mes   db "Programme interrompu.$"
nom_fich  db "A:TEST.DAT",0                       ;Nom du fichier test

data      ends                            ;Fin du segment de données

;-- Code ---------------------------------------------------------

code      segment para 'CODE'             ;Définition du segment de code

          assume cs:code, ds:data, ss:stackseg

start     proc far
```

```
;-- Installer les deux handlers d'interruption ----------------

        push cs                                 ;CS sur la pile
        pop  ds                                 ;Retirer pour placer dans DS
        mov  ax,2523h                   ;n° fonc.: Régler Handler Ctrl-Break
        mov  dx,offset cbreak     ;DS:DX : maintenant l'adresse du H.
        int  21h                                ;Appeler l'interruption DOS

        mov  al,24h                             ;Régler l'interruption 24h
        mov  dx,offset cerror  ;DS:DX contient l'adresse du nouveau H.
        int  21h                                ;Appeler l'interruption DOS

        mov  ax,data    ;Charge adresse segment du segment de données
        mov  ds,ax                              ;dans le registre DS

        ;-- Ici vous pouvez ansiedeln votre programme----------------
        ;-------------------------------------------------------------

        ;-- En guise de démonstration, on essaye d'ouvrir un fichier -
        ;-- sur le lecteur de disquettes ouvert ----------------------

ouvrir_fich:
        mov  ah,3dh                             ;N°fonc.: Ouvrir fichier
        mov  al,0                               ;Mode fich.: Lecture seule
        mov  dx,offset nom_fich     ;DS:DX = Adresse du nom de fichier
        int  21h                                ;Appeler interruption DOS 21h
        jnc  fin                                ;Pas d'erreur? NON --> FIN

        cmp  cr_err,0                                   ;Critical Error?
        je   fin                                        ;NON --> FIN

        call crit_err                   ;Une erreur critique est apparue
        jmp  ouvrir_fich          ;CRIT_ERR ne revient que si l'opération
                                                ;doit être répétée
                                        ;(il est impossible d'ignorer)

        ;-- Les Handlers ne doivent pas être réinstallés avant -------
        ;-- la fin du programme car c'est DOS qui s'en charge --------

fin:    mov  ah,9                               ;N°fonc.: Afficher la chaîne
        mov  dx,offset end_mes                  ;DS:DX = Adresse du message
        int  21h                                ;Appeler interruption DOS

        mov  ax,4C00h       ;N°fonc.: Terminer le programme (ERRCODE=0)
        int  21h                                ;Appeler interruption DOS
                                                ;et terminer le programme

start           endp

;-- CRIT_ERR: Appelé dans le programme après découverte ----------------
;--           d'une erreur critique ----------------------------------

crit_err        proc near

        ;-- Afficher le message et réclamer l'entrée de l'utilisateur-

ask:    mov  ah,9                               ;N°fonc.: Sortir la chaîne
        mov  dx,offset cr_mes                   ;DS:DX = Adresse du message
        int  21h                                ;Appeler l'interruption DOS

        mov  ah,1                               ;N°fonc.: Entrer les caractères
        int  21h                                ;Appeler l'interruption DOS
        push ax                                         ;Apercevoir l'entrée

        mov  ah,9                               ;N°fonc.: Sortir la chaîne
        mov  dx,offset next_line                ;DS:DX = Adresse du message
        int  21h                                ;Appeler l'interruption DOS

        ;-- Evaluer l'entrée de l'utilisateur ----------------------

        pop  ax                                         ;Relire l'entrée
        cmp  al,"A"                                     ;Annuler?
        je   end_up                             ;zur "Aufräum"-Prozedur
        cmp  al,"a"                                     ;Annuler?
        je   end_up                             ;zur "Aufräum"-Prozedur
        cmp  al,"w"                                     ;Répéter?
        je   crend                      ;Oui, vers la fin de la procédure
        cmp  al,"W"                                     ;Répéter?
        jne  ask                                ;NON, nouvelle requête

crend:  ret                                     ;Retour à l'appelant

crit_err  endp

;-- END_UP: Exécute une "fin correcte" ----------------------------

end_up          proc near

        ;-- Tous les fichiers ouverts peuvent être fermés ici et -----
        ;-- la mémoire système allouée par le programme libérée ------

        mov  ah,9                               ;N°fonc.: Sortir la chaîne
        mov  dx,offset brk_mes                  ;DS:DX = Adresse der Meldung
        int  21h                                ;Appeler l'interruption DOS

        mov  ax,4C00h     ;Terminer normalement le programme à travers
        int  21h                                        ;la fonction 4Ch

end_up          endp

;-- CBREAK: Le nouveau handler Ctrl-Break -------------------------------

cbreak          proc far

        ;-- Tous les registres modifiés dans cette routine -----------
        ;-- doivent être sauvegardés sur la pile ---------------------
        ;-- (excepté le registre Flag). ------------------------------

        push ds

        mov  ax,data    ;Charge adresse segment du segment de données
        mov  ds,ax                                      ;dans le registre DS


        ;-- Ici, on peut par exemple ouvrir une fenêtre écran --------
        ;-- où on demande à l'utilisateur s'il souhaite --------------
        ;-- terminer réellement le programme. ------------------------

        jmp  go_on                              ;Ne pas terminer le programme


        ;-- Si l'utilisateur a décidé de terminer le programme -------
        ;-- on peut inclure ici une routine permettant ---------------
        ;-- de terminer le programme. ---------------------------------

        jmp  end_up                             ;Préparer la fin de programme


        ;-- Le programme n'est pas interrompu mais il est plutôt  ----
        ;-- traité tout à fait normalement ----------------------------

go_on:  pop  ds                                 ;Retirer les registres sauvegardés
        iret                                    ;Retour au DOS où traitement de
                                        ; la fonction interrompue reprend normalement
cbreak          endp

;-- CERROR: Le nouveau handler Critical-Error ---------------------------

cerror          proc far

        ;-- Tout registre SS, SP, DX, ES, DX, CX et BX modifié -------
        ;-- dans cette routine doit être sauvegardé sur la pile. -----

        sti                             ;Autoriser à nouveau les interruptions
        push ds

        mov  ax,data    ;Charge adresse segment du segment de données
        mov  ds,ax                                      ;dans le registre DS

        mov  cr_err,1                           ;Afficher l'erreur critique
        mov  ax,di                              ;Numéro d'erreur dans AX
        mov  cr_typ,al                  ;Apercevoir le numéro d'erreur
        mov  al,3               ;Terminer l'appel de fonction avec erreur

        pop  ds                                         ;Relire DS
        iret

cerror  endp

;-------------------------------------------------------------------------

code    ends                                    ;Fin du segment de code
        end  start                      ;Commencer l'exécution du programme
                                                ;avec la procédure START
```

25. Les drivers de périphériques

L'un des chapitres les plus fascinants mais aussi l'un des plus difficiles de la programmation de DOS concerne les drivers de périphériques. Ce sont des programmes qui permettent à DOS d'accéder aux différents périphériques : clavier, disque dur, CD-ROM, etc... Le côté fascinant est dû à la diversité des appareils parfois exotiques que l'on intègre dans l'environnement de DOS. La difficulté provient de la programmation en assembleur qui s'avère indispensable dans ce cas.

Ce chapitre vous apprendra tout sur les drivers de périphériques de DOS, leur organisation et leur développement. Vous y trouverez les programmes sources de plusieurs drivers pleinement fonctionnels que vous pourrez utiliser comme base de vos propres développements.

25.1. Les drivers de périphériques sous DOS

La notion de driver n'est pas apparue avec DOS. Elle existait déjà dans de nombreux autres systèmes d'exploitation. Nous commencerons par nous interroger sur la nature et les fonctions des drivers de périphériques.

Fonction des drivers de périphériques

Un driver de périphérique est un programme qui réalise une interface entre un niveau logiciel supérieur et un niveau matériel. Le driver isole le niveau supérieur des caractéristiques physiques du périphérique en prenant en charge les commandes matérielles. Le niveau supérieur n'est pas forcément celui du système d'exploitation. De nombreux logiciels et langages du commerce (par exemple AutoCad ou Turbo Pascal) disposent de drivers de périphériques pour piloter des matériels spécifiques.

Dans le cadre de ce chapitre nous nous intéresserons uniquement aux drivers de périphériques que DOS utilise pour communiquer avec le matériel. Si on considère un système d'exploitation comme une superposition de plusieurs couches, les drivers de périphériques constituent la couche la plus profonde qui permet à toutes les autres de fonctionner indépendamment du matériel. Pour adapter un système d'exploitation à divers environnements matériels, les drivers présentent des avantages considérables car ils seront seuls concernés par les ajustements.

Dans les systèmes d'exploitation anciens, les drivers de périphériques étaient inclus dans le code lui-même, de sorte qu'il était très difficile voire impossible de brancher des périphériques d'un type nouveau. La version 2.0 de DOS a introduit une souplesse nouvelle pour inclure des drivers dans DOS. L'utilisateur est à même de rendre disponible

sous DOS les disques les plus étranges, les cartes d'extension EMS les plus exotiques tout en conservant l'efficacité du système.

Pensez un moment à toutes les centaines (ou milliers ?) de disques durs qui sont connectables sur un PC. Il se différencient non seulement par leur capacité, leurs dimensions (nombre de têtes, de cylindres, de secteurs par piste), mais ils fonctionnent aussi avec des contrôleurs différents qui sont complètement incompatibles entre eux. Comment DOS pourrait-il s'en sortir tout seul avec une telle profusion de possibilités et de particularités, alors que chaque semaine voit apparaître de nouveaux modèles de disques (généralement plus puissants) ? Lorsque DOS accède aux répertoires et aux fichiers, il ne s'adresse jamais directement à un lecteur mais il passe par un driver de périphérique. Un nouveau disque peut être immédiatement installé sous DOS à partir du moment où le fabricant livre le driver associé.

Un driver se compose de quelques informations d'état qui donnent à DOS des indications sur son type et ses capacités, et d'une série de routines appelées "fonctions du driver". Ces dernières prennent en charge les différents services dont DOS a besoin pour accéder au périphérique commandé. C'est ainsi qu'un driver de disque dur devra disposer de fonctions de lecture, d'écriture et de vérification de secteurs.

Développement de drivers de périphériques

Comme la communication entre DOS et les drivers de périphériques repose sur un schéma simple d'appels de fonctions et de structures de données, un programmeur de niveau moyen est parfaitement en mesure de développer lui même des drivers. Dans le cadre de ce chapitre, nous présenterons un driver de disque virtuel qui n'a demandé que quelques heures de programmation.

Pourtant les drivers sous DOS ont une exigence ennuyeuse : ils ne peuvent être programmés qu'en assembleur. Il est pratiquement impossible de réaliser un driver en langage évolué. En effet les drivers doivent se présenter à DOS dans un format particulier qui n'est compatible avec aucun des types EXE ou COM et qui ne peut pas être généré directement par les compilateurs usuels. Grâce aux explications détaillées données dans ce chapitre, cet inconvénient ne devrait pas constituer un obstacle même si le codage en assembleur réclame plus de travail que la programmation en langage évolué.

Il ne faut pas oublier qu'en contrepartie les programmes écrits en assembleur offrent une vitesse d'exécution maximale, ce qui est important pour un driver. Supposez par exemple qu'un driver de disque dur soit mal programmé. A quoi servirait la vitesse d'accès du lecteur ou la mémoire cache, malgré ses qualités intrinsèques, le disque risque d'être surclassé par ses concurrents.

Intégration des drivers

Les drivers sont incorporés au système au moment du lancement de DOS, ils ne peuvent pas être chargés au niveau de la ligne de commande de DOS comme les programme COM et EXE ordinaires. Le "boot" commence par installer les drivers prédéfinis dans le noyau de DOS. Il s'agit des drivers appelés NUL, $CLOCK, CON, AUX et PRN. Il vient s'y ajouter un driver pour le lecteur de disquette et éventuellement un driver pour le disque dur s'il est disponible.

Nom	Driver
NUL	Avaleur de bits imaginaire
$CLOCK	Horloge
CON	Clavier et écran
AUX	Interface série
PRN	Interface parallèle

Les drivers fixes standard présentent la particularité de ne pas accéder directement aux périphériques mais de se servir des différentes fonctions du BIOS. Pour ce qui est du disque dur, le driver prédéfini ne peut donc entrer en action que si ledit disque est supporté par le BIOS, ce qui posait toujours problème dans le temps. Le BIOS n'est pas en mesure de commander n'importe quel disque, c'est là qu'interviennent les drivers fournis par le fabricant.

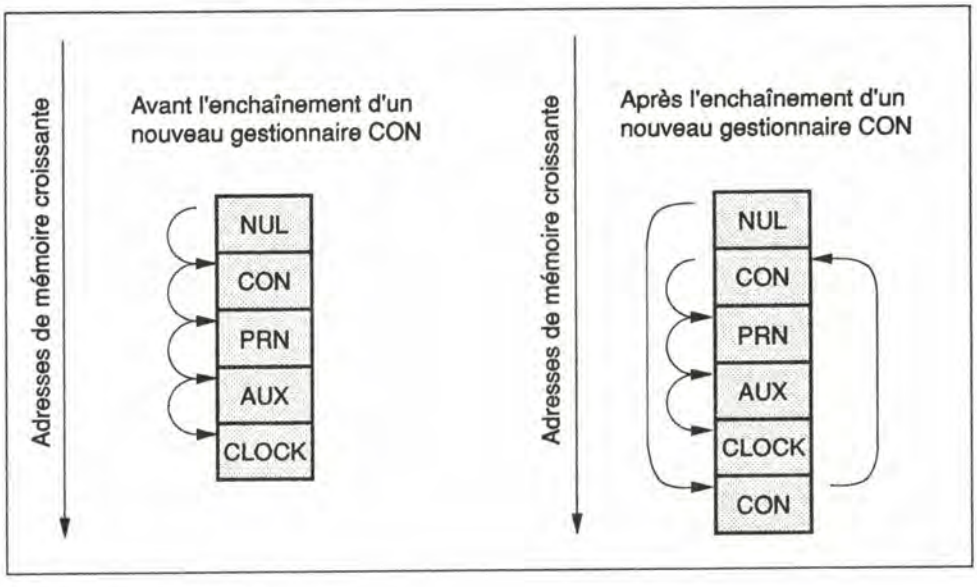
Les drivers standard sont rangés en mémoire directement à la suite les uns des autres et ils sont reliés par une sorte de chaîne. Si l'utilisateur veut installer un driver personnel, il doit l'indiquer à DOS dans le fichier CONFIG.SYS. Ce fichier est chargé et exploité au cours de l'opération de lancement du système, après intégration des drivers standard. Si DOS y rencontre l'instruction DEVICE=, il sait qu'un nouveau driver doit être intégré. Le nom du driver ainsi qu'une éventuelle désignation de lecteur et de chemin d'accès doivent être indiqués à la suite du signe =. Voici comment se présente par exemple l'instruction d'intégration de "ANSI.SYS", un driver qui offre des possibilités étendues d'affichage des caractères et qui fait partie des drivers fournis avec DOS :

```
DEVICE=ANSI.SYS
```

Le nouveau driver est alors intégré dans la chaîne : il vient se placer derrière le premier membre de la chaîne, le driver NUL, mais avant tous les autres. Il en est ainsi pour que les drivers installés de cette façon puissent remplacer l'un des drivers standard. En fait le driver ANSI.SYS que nous avons cité plus haut remplace le driver CON qui était en vigueur auparavant.

Le nouveau driver CON (ANSI.SYS) se trouve placé avant l'ancien pour que les appels des fonctions de clavier-écran passent désormais par lui. Si une opération fait appel au

driver CON , DOS recherchera le premier qui se présente dans la liste chaînée. La recherche commence par le driver NUL et suit les éléments successifs de la chaîne : le premier driver CON trouvé sera bien le driver ANSI.SYS. L'ancien driver CON ne peut plus être adressé, il reste ignoré en mémoire.



La chaîne des drivers

Mais tous les drivers ne peuvent être remplacés par de nouveaux. Le driver NUL reste toujours le premier dans la chaîne : un nouveau driver de ce nom serait de toutes façons inséré après lui et ne pourrait donc jamais être appelé. Par ailleurs les drivers pour les mémoires de masse (disquette et disque dur) ne peuvent être remplacés aussi facilement. Contrairement en effet aux drivers que nous avons cités jusqu'ici, ils ne sont pas associés à un nom (comme CON ou AUX), mais à une désignation de périphérique (A, B, etc...). Or cette désignation de périphérique est attribuée d'office par DOS sans qu'il soit possible de la changer. Par conséquent, si un nouveau driver de disquette est intégré dans le système, DOS l'appellera probablement D et il sera ignoré des programmes qui accèdent au lecteur par sa désignation habituelle A. Ce problème peut toutefois être résolu si tous les accès à un périphérique déterminé sont détournés sur un autre. L'instruction appropriée de DOS est ici ASSIGN.

Deux types de drivers

Vous voyez donc qu'il est nécessaire de traiter différemment les drivers de mémoires de masse et les drivers tels que CON, PRN, AUX etc... EN fait DOS distingue deux types de drivers de périphériques :

- ✓ les drivers de caractères (character device drivers)
- ✓ les drivers de blocs (block device drivers)

Les premiers communiquent en effet avec le matériel assujetti caractère par caractère (plus exactement : octet par octet) . Parmi eux on trouve : l'écran, le clavier, l'imprimante, etc.... Les drivers de blocs, au contraire, peuvent transférer toute une série de caractères (bloc) en un seul appel de fonction (disquettes, disque dur, streamer, etc...). Ces deux groupes de drivers diffèrent donc quelque peu, ce qui se traduit par exemple par le fait qu'ils offrent au système des fonctions parfois différentes.

Penchons-nous tout d'abord sur les drivers de caractères dont la structure est un peu plus simple que celle des drivers de blocs.

25.1.1. Les drivers de caractères

Comme nous l'avons déjà indiqué, les drivers de caractères transfèrent un octet et un seul lors de chaque appel de fonction. Ils conviennent donc à la communication avec des périphériques du genre clavier, écran, imprimante ou modem. Chaque driver de périphérique ne peut toutefois desservir qu'un seul périphérique.

Ces drivers peuvent opérer sous deux modes, le mode dit Cooked et le mode dit Raw. La différence ne tient pas aux types de fonctions mises à la disposition de DOS. Elle se révèle du côté de l'appelant des différentes fonctions de DOS qui se terminent par un appel au driver. Le chapitre xx au cours duquel ont été étudiés les services rendus par DOS dans le domaine des entrées-sorties de caractères a déjà mis l'accent sur ce point.

Différences entre modes Cooked et Raw

Lorsque des caractères sont lus par le driver en mode Cooked, DOS les transfère tout d'abord dans un buffer interne. Le système examine alors chaque caractère pour voir s'il s'agit d'un caractère de commande du genre <ENTREE>, <CTRL>-<P>, <CTRL>-<S> ou <CTRL>-<C>. Dès qu'il rencontre un caractère <ENTREE>, il ne réclame plus rien au driver, même si le nombre attendu de caractères n'a pas encore été lu. C'est alors seulement qu'il copie les caractères lus depuis son buffer interne dans le buffer du programme appelant.

Lorsque des caractères sont affichés en mode Cooked, DOS teste après chaque affichage si l'une des touches <CTRL>-<C> ou <CTRL>-<BREAK> a été actionnée, auquel cas le programme en cours est interrompu. Si la touche <CTRL>-<S> est pressée, le programme est arrêté jusqu'à ce qu'une touche quelconque soit à nouveau frappée. <CTRL>-<P>, enfin, sert à détourner l'affichage de l'écran vers l'imprimante (PRN). Ce détournement ne prend fin qu'après une nouvelle pression sur la même touche.

Tous ces tests disparaissent en mode Raw. Si un programme veut lire 10 caractères, ce seront effectivement 10 caractères qui seront lus, même si la touche <ENTREE> est déjà actionnée en guise de second caractère. Le détour par le buffer interne du DOS devient de ce fait inutile et les caractères sont transférés directement dans le buffer du programme appelant. En ce qui concerne l'affichage de caractères, il n'y a pas non plus de test des touches <CTRL>-<C> ou <CTRL>-<BREAK>.

Pour définir ou interroger le mode de fonctionnement d'un driver de caractères, vous pouvez utiliser la fonction 44h de l'interruption 21h de DOS que nous décrirons plus en détail à la fin de ce chapitre.

25.1.2. Drivers de blocs

Les drivers de blocs servent à communiquer avec les mémoires de masse comme le disque dur, la disquette ou le streamer. C'est pourquoi ils travaillent non pas au niveau des caractères mais au niveau des blocs. Un bloc est généralement un secteur physique d'environ 512 octets. Dans certains cas, un unique appel de fonction a pour effet de transférer plusieurs blocs ensemble. La taille de ces blocs peut varier d'un modèle de périphérique à l'autre.

Lorsque DOS désire accéder à une mémoire de masse par le moyen d'un driver de blocs, il lui communique le numéro du premier secteur concerné. Le driver convertit ce numéro de secteur logique, décompté à partir de 0, en une adresse physique (tête, cylindre, piste) avant de procéder à l'accès. La correspondance entre adresse logique et adresse physique du secteur est l'affaire exclusive du driver. La seule chose importante, c'est que la correspondance soit unique : chaque secteur logique doit être associé à un seul secteur physique.

Contrairement aux drivers de caractères, les drivers de blocs peuvent gérer simultanément plusieurs périphériques. Ainsi un driver de disque dur pourra faire fonctionner en même temps deux disques et même trois ou quatre. Un driver de blocs est aussi capable de partitionner un périphérique en plusieurs unités logiques (volumes) : c'est ce qui se passe souvent pour les disques durs.

Désignation des unités gérées

Les périphériques gérés ne portent pas de nom particulier ou de nom de fichier, on les désigne par des lettres : A, B, C. Ces désignations ne sont pas librement choisies par les drivers, elles sont imposées par DOS. Le système lui-même les attribue en tenant compte de l'emplacement du driver dans la chaîne des drivers. Le premier périphérique géré par un driver de blocs est baptisé A, le second B, le troisième C et ainsi de suite.

Si un driver commande plusieurs unités logiques, elles portent nécessairement des désignations consécutives. Si la lettre C a été attribuée à un driver de disque dur qui gère

trois unité logiques, il accapare aussi les lettres D et E. Le driver de blocs suivant sera appelé F.

Chacune des unités logiques commandées par un driver de périphérique doit disposer d'une table d'allocation des fichiers (file allocation table = FAT) ainsi que d'un répertoire racine. Contrairement aux drivers de caractères, les drivers de blocs ne connaissent pas la distinction entre modes Cooked et Raw. Les lectures et écritures qu'ils effectuent portent toujours sur le nombre exact de blocs qui leur a été commandé (à moins qu'une erreur ne se produise) et ils ne touchent pas aux données transférées.

25.1.3. Accès aux drivers

Il existe plusieurs possibilités pour accéder à un driver de périphérique. Les fonctions habituelles gérant les FCB ou les handles peuvent parfaitement exploiter les drivers de caractères lorsqu'on y remplace un nom de fichier par un nom de driver, par exemple 'CON'. On peut aussi accéder aux drivers de blocs par le moyen des fonctions usuelles de DOS (gestion des fichiers, des répertoires, etc...) en spécifiant simplement la désignation du driver au lieu de l'ancienne désignation.

L'accès évoqué n'est pas un accès direct car on se sert de différentes fonctions de DOS qui font appel au driver.

Pour effectuer des opérations de lecture ou écriture sur un driver de caractères vous pouvez bien sûr utiliser les fonctions gérant les FCB et les handles mais vous disposez aussi des fonctions 1h à Ch de l'interruption 21h. Depuis la version 2.0 de DOS, celles-ci accèdent en effet aux drivers pour toutes les entrées-sorties de caractères.

Notez cependant qu'il existe encore deux autres moyens d'entrer en contct avec les drivers. C'est surtout la fonction 44h de DOS, appelée IOCTL (I/O Control), qui joue un grand rôle dans ce domaine en offrant de nombreuses options. Nous en apprendrons davantage sur cette fonction un peu plus loin.

25.2. Structure d'un driver de périphérique

Bien que les drivers de blocs et les drivers de caractères diffèrent par certains aspects importants, ils présentent néammoins une structure uniforme.

Cette structure comporte un en-tête (header) qui donne des informations générales et deux routines qui gèrent toute la communication entre DOS et le driver. Les deux routines s'appellent routine de stratégie et routine d'interruption. Mais l'interruption dont il est question ici n'a rien à voir avec les interruptions matérielles et logicielles rencontrées jusqu'ici. Elles doivent être de type FAR pour que DOS puisse aussi les appeler de l'extérieur du driver.

Structure de l'en-tête

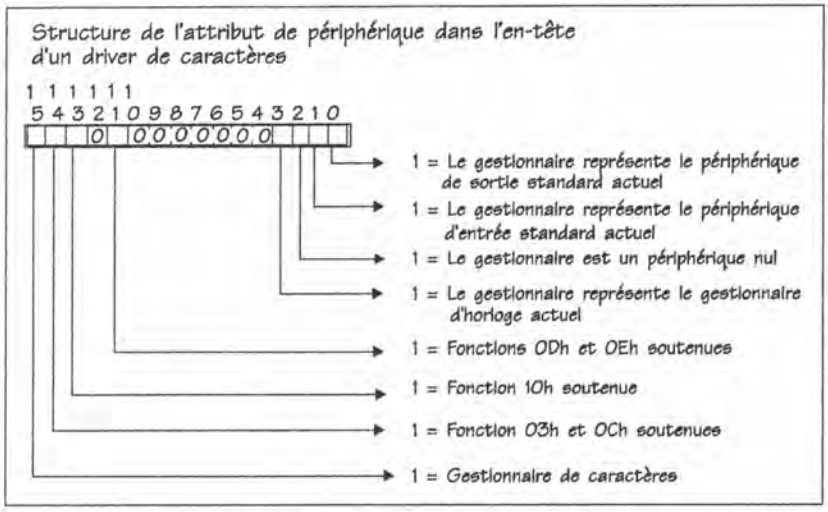
L'en-tête du driver contient certaines informations capitales pour permettre son exploitation par DOS. Par définition l'en-tête est situé tout au début du driver.

Structure de l'en-tête d'un driver de périphérique		
Adresse	**Contenu**	**Type**
+00h	Adresse du prochain driver	1 PTR
+04h	Attribut de périphérique	1 WORD
+06h	Offset de la routine de stratégie	1 WORD
+08h	Offset de la routine d'interruption	1 WORD
+0Ah	Nom du driver pour les drivers de caractères (complété avec des espaces) ou nombre de périphériques gérés pour les drivers de blocs	8 BYTES
Longueur : 18 octets		

Le premier champ établit la liaison avec le driver suivant dans la chaîne des drivers. Il doit être initialisé avec la valeur -1 pour que DOS reconnaisse l'en-tête.

Attribut de périphérique

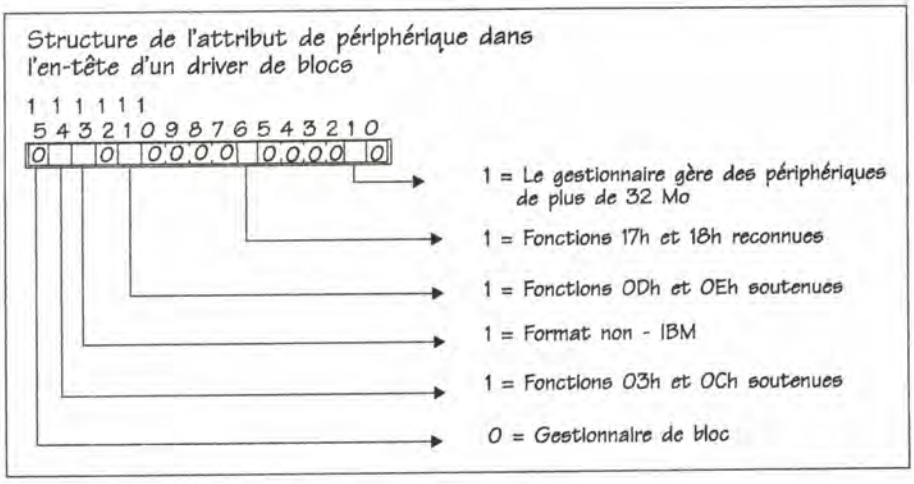
Le second champ est un champ de bits qui sert à décrire le driver de périphérique. Il indique, entre autres, à DOS s'il s'agit d'un driver de blocs ou d'un driver de caractères. Cette indication figure dans le bit 15 : 0 = driver de caractères, 1=driver de blocs. Le reste du champ doit être interprété en fonction de ce renseignement. Voici d'abord la structure de l'attribut pour un driver de caractères :



Comme le montre la figure précédente, les bits 0 à 3 servent à identifier le driver de caractères. On peut ainsi savoir s'il s'agit d'un nouveau driver standard pour les entrées-sorties par le périphérique CON, d'un autre driver NUL ou d'un nouveau driver d'horloge. Si aucun de ces bits n'est à 1, le driver en question ne remplace aucun driver prédéfini mais constitue un driver additionnel ordinaire.

Les autres bits signalent le support de diverses fonctions. Il faut que vous sachiez à ce propos que DOS connaît en tout 15 différentes fonctions de driver, mais elles ne sont pas nécessairement exploitées par tous les drivers. C'est ainsi que la disponibilité des fonctions optionnelles 03h, 0Ch, 0Dh, 0Eh et 10h est indiquée par ces bits. Notez que certaines de ces fonctions n'existent que depuis la version 3.1 ou 5 de DOS : le bit correspondant de l'attribut de périphérique n'est alors reconnu qu'à partir de cette version. Nous reviendrons là dessus lorsque nous étudierons le détail des fonctions.

Le bit 15 montre qu'on est en présence d'un driver de caractères s'il est à 1. Ce n'est qu'à cette condition que les autres bits acquièrent la signification décrite. Dans le cas d'un driver de blocs, les bits s'interprètent autrement comme le montre la figure suivante :



A côté des bits qui signalent le support de différentes fonctions, c'est surtout le bit 1 qui est significatif. Ce bit n'est exploité que depuis la version 4.0 de DOS. Lorsqu'il est à 1, il indique que le driver est capable de gérer des périphériques et des partitions de plus de 32 Mo, ce qui est possible depuis que la version 4.0 de DOS a agrandi la taille des clusters.

Lorsque la barrière des 32 Mo est dépassée, les secteurs ne peuvent plus être donnés sous la forme de nombres entiers de 16 bits. Le nombre maximum que l'on peut exprimer avec 16 bits est 65536. Multiplié par la taille d'un secteur standard soit 512 Ko ce nombre donne exactement 32 Mo. Les drivers dont le bit 1 est armé exploitent les numéros de secteurs dans un autre format qui leur permet d'aller au-delà de 32 Mo. Nous reviendrons sur ce sujet un peu plus loin.

Autres champs de l'en-tête du driver

Mais revenons à notre en-tête. Après l'attribut de périphérique on trouve dex champs qui contiennent le segment et l'offset des routines de stratégie et d'interruption. Ces routines desservent la communication entre DOS et le driver. En réalité, seuls les offsets sont utilisés car tout comme les programmes COM les drivers de périphériques n'occupent qu'un seul segment. L'indication du segment est de ce fait superflue.

Dans le dernier champ on trouve enfin le nom du driver pour autant qu'il s'agisse d'un driver de caractères. Si ce nom comporte moins de 8 caractères, il doit être complété par des espaces (code ASCII 32). S'il s'agit par contre d'un driver de blocs, le premier octet du champ indique le nombre de périphériques logiques soutenus par ce driver. Les 7 octets suivants doivent alors être mis à 0.

Communication avec les routines de stratégie et d'interruption

La routine de stratégie est appelée par DOS dans deux circonstances : lors de l'installation du driver et avant tout appel de l'une des fonctions du driver. Elle doit recevoir en ES:BX l'adresse d'une structure de données contenant des informations sur l'opération à effectuer et les données correspondantes. La routine d'initialisation n'exécute pas elle-même ces opérations. Elle se contente de stocker l'adresse du bloc de données transmis, après quoi elle rend immédiatement le contrôle à DOS. Le système appelle alors la routine d'interruption du driver qui se charge de l'exécution de l'opération elle-même. Mais nous allons y revenir de suite...

Derrière ce mécanisme se cache une idée très simple : DOS n'a pas besoin de connaître les adresses de toutes les fonctions du driver, il lui suffit de se servir des routines de stratégie et d'interruption comme interface d'accès à ces fonctions. C'est pourquoi le bloc de données dont l'adresse est transmise à la routine de stratégie contient aussi le numéro de la fonction de driver à invoquer. Ce bloc de données se compose en principe de 13 octets auxquels viennent s'en rajouter d'autres en fonction des besoins de la fonction à appeler. La figure suivante montre la structure de ce bloc de données tel qu'on le rencontre dans la plupart des appels, qu'il s'agisse de transmettre des caractères ou des secteurs. Les 13 premiers octets sont repésentés dans chaque appel :

Structure du bloc de paramètres pour l'appel d'une fonction d'un driver de périphérique		
Adresse	**Contenu**	**Type**
+00h	Longueur du bloc de données en octets	1 BYTE
+01h	Numéro du périphérique appelé (drivers de blocs uniquement)	1 BYTE
+02h	Numéro de la fonction à appeler	1 BYTE
+03h	Mot d'état	1 WORD
+05h	Réservé	8 BYTES
+0Dh	Descripteur de support (drivers de blocs uniqt)	1 BYTE
+0Eh	Adresse du buffer de transmission	1 PTR
+12h	Nombre de secteurs à traiter (driver de blocs) Nombre d'octets à traiter (driver de car.)	1 WORD
+14h	Numéro du premier secteur (drivers de blocs uniquement)	1 WORD
Longueur : 22 octets		

Comme nous l'avons indiqué plus haut, la routine d'interruption est appelée immédiatement après que DOS ait déclenché la routine de stratégie.

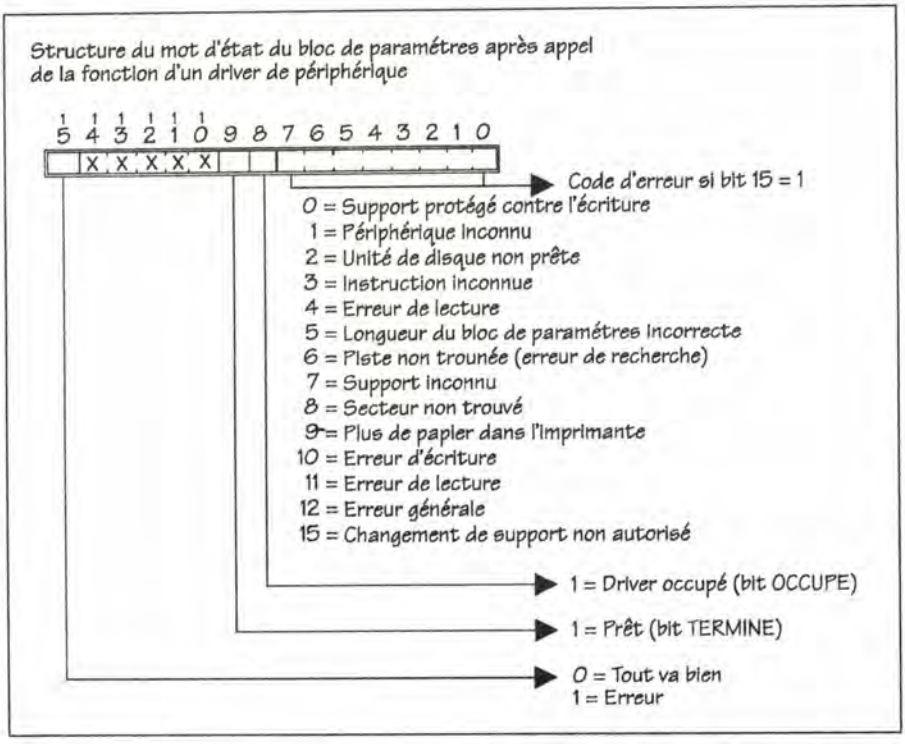
De ce fait la séparation en deux routines n'a pas beaucoup de sens. Elle a probablement été introduite pour préparer l'évolution de DOS vers un système multitâche. Dans ce cas, un driver peut être sollicité en lecture ou en écriture par plusieurs programmes à

la fois et il est intéressant de séparer la "prise de commande" de l'"exécution". On sait aujourd'hui que DOS va rester un système monotâche, mais cette décision ne remet pas en cause la séparation des routines de stratégie et d'interruption.

La première tâche de la routine d'interruption consiste à sauvegarder (sur la pile) les registres du processeur qui risquent d'être modifiés par la suite. Elle lit ensuite dans le champ 3 du bloc de données transmis à la routine de stratégie, le numéro de la fonction à exécuter, qu'elle appelle alors sur-le-champ. Après exécution de ladite fonction, elle inscrit un nombre dans le champ d'état du bloc de données et restaure ensuite les registres du processeur qui avaient été sauvegardés au début. Elle rend enfin le contrôle à la fonction de DOS qui l'a appelée.

Signification du champ d'état

Une fois la routine d'interruption arrivée à son terme, la valeur du champ d'état est très importante pour la fonction DOS appelante car elle indique si l'exécution s'est bien passée. C'est pourquoi toute fonction de driver doit absolument mettre à 1 le bit TERMINE (bit 8) du champ d'état, même si elle est fictive (dummy), c'est-à-dire si elle n'exécute aucune opération. Examinons donc de plus près la structure du champ d'état :



25.3. Les fonctions d'un driver de périphériques

Dans la version 2 de DOS, tout driver à installer doit supporter 13 fonctions, numérotées de 0 à 12, même si leur seul effet est de mettre à 1 le bit TERMINE du champ d'état. Dans les versions ultérieures de DOS plusieurs fonctions supplémentaires sont venues s'y rajouter mais leur support est facultatif. La présence des fonctions est précisée comme nous l'avons vu par certains bits de l'attribut de périphérique

Certaines de ces fonctions ne concernent qu'un des deux types de drivers alors que d'autres (par exemple la fonction d'initialisation) s'appliquent aux deux. L'ancienne règle reste valable : toutes les fonctions même celles qui ne servent à rien doivent au moins mettre à 1 le bit TERMINE.

Le reste de ce chapitre est consacré à la description des différentes fonctions d'un driver. Voici comment seront données ces explications.

Chacune des fonctions tire ses arguments du bloc de données dont l'adresse est transmise à la routine de stratégie. C'est également dans ce bloc de données qu'elle placera ses "résultats".

C'est pourquoi nous indiquerons pour chaque fonction les offsets (par rapport au début du bloc) des champs où doivent être placés les arguments et où viennent se placer les résultats.

Nous indiquerons non seulement les offsets mais aussi s'il s'agit d'informations sous forme d'octets ou de mots. Il existe aussi un troisième type de données appelé PTR. Il s'agit d'un pointeur FAR sur un buffer, qui est donc composé de deux mots consécutifs. Le premier mot contient comme d'habitude l'offset du buffer, le second son segment.

Fonction 00h : Initialisation du driver

Cette fonction est appelée par DOS lors du chargement du système. Elle sert à initialiser le driver de périphérique. L'initialisation peut concerner le matériel, différentes variables internes ou bien encore un détournement d'interruptions. Comme l'ensemble du système n'est pas encore entièrement initialisé à cet instant, la routine d'initialisation ne peut appeler que les fonctions 01h à 0Ch (entrées-sorties de caractères) et 30h (version de DOS) de l'interruption 21h de DOS. Ces fonctions lui permettent déjà de tester le numéro de la version de DOS et d'afficher à l'écran un message signalant que le driver est activé.

Même si le driver qui vient d'être intégré est un driver CON, les sorties sur l'écran sont toujours effectuées par le moyen de l'ancien driver. Le nouveau driver n'est entièrement fonctionnel que lorsque la routine d'initialisation est achevée.

La routine d'initialisation peut prélever deux informations dans le bloc de données : d'abord l'adresse de la mémoire qui contient le texte du fichier CONFIG.SYS ayant requis l'intégration du driver. Ce texte présente le format suivant :

```
DEVICE=ANSI.SYS
```

Nous supposons ici que le driver appelé répond au nom d'ANSI.SYS. L'adresse de mémoire transmise référence le caractère placé à la suite du signe = (soit le A d'ANSI.SYS). Vous avez donc la possibilité d'ajouter d'autres informations à la suite du nom du driver. DOS n'en tiendra aucun compte mais vous pourrez les utiliser comme commentaire

Paramètres d'entrée et de sortie de la fonction 00h d'un driver

- Paramètres d'entrée

Adresse	Contenu	Type
+02h	Numéro de fonction (ici 00h)	1 BYTE
+0Eh	Adresse maximale atteinte par la fin du driver (à partir de DOS 5.0)	1 PTR
+12h	Adresse du caractère qui suit le signe = dans l'instruction DEVICE du fichier CONFIG.SYS	1 PTR
+16h	Désignation du périphérique supporté par le driver (drivers de blocs uniqt et à partir de la version 3.0 du DOS)	1 BYTE

- Paramètres de sortie

Adresse	Contenu	Type
+03h	Mot d'état	1 WORD
+0Dh	Nombre de périphériques supportés (drivers de bloc uniquement)	1 BYTE
+0Eh	Adresse de la première mémoire libre à la suite du driver	1 PTR
+12h	Adresse du vecteur contenant les adresses des blocs de paramètres du DOS (DPB) (drivers de blocs uniquement)	1 PTR
+17h	Indicateur d'erreur, doit prendre une valeur différente de 0 lorsque le driver pour une raison ou une autre n'a pas pu être initialisé (à partir de la version 4.0 de DOS)	1 BYTE

La seconde information n'est fournie qu'à partir de la version 3 de DOS et uniquement pour les drivers de blocs. Il s'agit de la lettre désignant le premier périphérique logique du driver. La valeur 0 signifie ici A, 1 B, etc...

La routine d'initialisation doit également renvoyer 4 paramètres à la fonction DOS appelante. Le premier argument sera, comme pour toutes les fonctions, son état c'est-à-dire l'indication si la fonction a pu être exécutée sans problème. Pour un driver de blocs, il faut également indiquer le nombre de périphériques logiques supportés. A priori DOS peut tirer cette information de l'en-tête du driver mais il se pourrait qu'au

moment de l'exécution le driver prenne une autre décision, par exemple en interrogeant le nombre de partitions effectivement disponibles sur le disque dur.

Le driver doit également transmettre la première adresse libre après la fin de la mémoire vive qu'il occupe, pour que DOS sache où il peut installer le prochain driver. Si le driver pour une raison ou une autre n'a pas pu prendre son service, il devra reporter son adresse de début dans ce champ pour que DOS reprenne la place rendue disponible.

Il faut noter qu'à partir de la version 5 de DOS un driver ne peut plus accaparer toute la mémoire qu'il désire. Il lui est donné la possibilité de se charger dans la mémoire supérieure qui en règle générale est plutôt exiguë par rapport aux grands espaces de la TPA située en-deça de la limite des 640 Ko.

A partir de la version 5 de DOS, le champ qui attend l'adresse où finit le driver reçoit en fait l'adresse d'extension maximale du driver. Si l'espace ainsi proposé ne suffit pas, le driver devrait normalement renoncer à son installation.

S'il le driver est un driver de blocs, il doit transmettre comme dernier argument l'adresse d'un champ qui associe à chaque unité logique une adresse de "bloc de paramètres BIOS" (BPB). Cette structure de données doit être établie par le driver, à l'intérieur de son segment de mémoire, pour chacune des unités logiques supportées. Ainsi le champ transmis ne comporte que des pointeurs NEAR, c'est-à-dire les offsets des différents BPB.

Le premier mot de cet tableau donne l'offset de la première unité, le deuxième mot l'offset de la deuxième unité et ainsi de suite. Le BPB est un bloc de données contenant différentes informations sur une unité périphérique logique. Si plusieurs unités logiques commandées par le driver (ou même toutes) présentent le même format, les éléments du tableau des adresses de BPB peuvent désigner un seul et même BPB.

Structure du bloc de paramètres du BIOS (BPB)		
Adresse	**Contenu**	**Type**
+00h	Octets par secteur	1 WORD
+02h	Secteurs par cluster	1 BYTE
+03h	Secteurs réservés (y compris secteur de boot)	1 WORD
+05h	Nombre de tables d'allocation de fichiers (FAT)	1 BYTE
+06h	Nombre maximum d'entrées dans le répertoire racine	8 BYTES
+08h	Nombre total de secteurs	1 WORD
+0Ah	Descripteur de support	1 BYTE
+0Bh	Nombre de secteurs par table d'allocation de fichiers	1 WORD
Longueur : 13 octets		

Structure du bloc de paramètres du BIOS (BPB)		
Ici commence le BPB étendu qui ne concerne que les drivers de blocs qui opèrent sur des partitions de plus de 32 Mo (à partir de la version 4.0 de DOS)		
+0Dh	Secteurs par piste	1 WORD
+0Fh	Nombre de tête de lecture/écriture	1 WORD
+11h	Secteurs réservés (y compris secteur de boot)	1 DWORD
+15h	Nombre total de secteurs de fichiers	1 DWORD
Longueur : 25 octets		

Codes de descripteur du support (Media Descriptor)	
Code	**Support**
F0h	Disquette 3", 2 faces, 80 pistes, 18 secteurs par piste Disquette 3", 2 faces, 80 pistes, 36 secteurs par piste
F8h	Disque dur
F9h	Disquette 5", 2 faces, 80 pistes, 15 secteurs par piste Disquette 3", 2 faces, 80 pistes, 9 secteurs par piste
FAh	Disquette 5", 1 face, 80 pistes, 8 secteurs par piste Disquette 3", 1 face, 80 pistes, 8 secteurs par piste
FBh	Disquette 5", 2 faces, 80 pistes, 8 secteurs par piste Disquette 3", 2 faces, 80 pistes, 8 secteurs par piste
FCh	Disquette 5", 1 face, 40 pistes, 9 secteurs par piste
FDh	Disquette 5", 2 faces, 40 pistes, 9 secteurs par piste
FEh	Disquette 5", 1 face, 40 pistes, 8 secteurs par piste
FFh	Disquette 5", 2 faces, 40 pistes, 8 secteurs par piste

A partir de la version 4.0 de DOS, la fonction 00H peut renvoyer un indicateur d'erreur à l'offset 17h du bloc de données. Si à l'issue de l'appel DOS trouve l'indicateur différent de 0, il émet un message "CONFIG.SYS : erreur à la ligne xxx".

Fonction 01h : Test de support

Cette fonction ne s'impose que pour un driver de blocs. Sur un driver de caractères, son rôle se limitera à mettre à 1 le bit TERMINE du champ d'état. DOS appelle cette fonction avant certains accès pour déterminer si le support a été changé. L'appel présente un intérêt concret lorsque certaines informations doivent être calculées à partir du répertoire d'un support (d'une disquette par exemple). Si le support n'a pas été changé depuis le dernier accès, DOS dispose encore de ces informations en mémoire

mais dans le cas contraire il devra commencer par les lire sur le support, ce qui ralentira bien sûr considérablement l'exécution de la tâche voulue.

La fonction 01h sert donc dans une situation de ce type à indiquer à DOS si le support a été changé. Dans beaucoup de cas, par exemple pour les lecteurs de disquette, il sera assez difficile de répondre à cette interrogation. C'est pourquoi DOS permet à la fonction 01h de répondre par "oui" ou par "non" mais aussi à la normande par "peut-être". La réponse fournie aura de toute façon une grande influence sur le comportement ultérieur du DOS :

- ✓ Si le support n'a pas été changé, l'accès à ce support peut s'effectuer immédiatement.

- ✓ Si le support a par contre été changé, DOS refermera tous les buffers internes concernant le périphérique logique intéressé. De ce fait, toutes les données qui auraient encore dû être transférées sur ce support seront perdues. DOS appelle ensuite la fonction 02h du driver de périphérique intéressé, charge la FAT puis le répertoire principal. Si la fonction de test de support répond par "peut-être", les prochaines opérations entreprises par DOS dépendront de l'état des buffers internes concernant le périphérique logique correspondant. Si ces buffers sont vides, DOS considérera que le support a bien été changé et il agira comme si la fonction 01h avait répondu par un "oui" franc et massif. Si les buffers ne sont cependant pas encore vides, c'est-à-dire s'ils contiennent encore des données à transférer sur le support, DOS considérera que le support n'a pas été changé et il écrira ces données sur le support. Cette façon de procéder présente toutefois un risque au cas où le support aurait bien été changé. Les données de la structure de données du "faux-nouveau" support seraient en effet complètement désorganisées par les données ainsi écrites, ce qui rendrait le support totalement inutilisable.

Paramètres d' entrée et de sortie de la fonction 01h d'un driver		
- Paramètres d'entrée		
Adresse	**Contenu**	**Type**
+01h	Numéro de l'unité logique appelée (0=1ere unité, 1=2me unité etc...)	1 BYTE
+02h	Numéro de fonction (ici 01h)	1 BYTE
+0Dh	Descripteur de support du périphérique appelé	1 BYTE
- Paramètres de sortie		
Adresse	**Contenu**	**Type**
+03h	Mot d'état	1 WORD
+0Eh	Y a-t-il eu changement de support : FFh = Oui 00h = Peut-être 01h = Non	1 BYTE

Paramètres d' entrée et de sortie de la fonction 01h d'un driver		
+0Fh	Adresse d'un buffer contenant le nom de volume précédent si le support a été changé	1 PTR

La réponse de la fonction de test de support entraîne donc des conséquences importantes et il convient de bien peser cette réponse. Mais examinons d'abord les paramètres transmis à la fonction. Le premier d'entre eux est le numéro de l'unité logique : il n'a évidemment de sens que si le driver supporte plusieurs périphériques logiques.

La fonction attend également le code du descripteur de support. Ce code indique à la fonction le type du dernier support trouvé dans l'unité logique concernée. L'indication ne présente d'intérêt que pour les périphériques pouvant traiter plusieurs formats (comme par exemple les lecteurs de disquette de l'AT, qui peuvent traiter aussi bien des disquettes de 360 Ko que des disquettes de 1,2 Mo).

Lorsque la fonction de test constate que l'unité testée possède un support fixe (cas d'un disque dur par exemple), elle pourra répondre systématiquement que le support "n'a pas été changé". S'il s'agit par contre d'un périphérique qui gère des supports mobiles (par exemple un lecteur de disquette), seule des procédures de tests plus complexes pourront fournir une réponse précise. Si l'on ne tient pas à recourir à des méthodes compliquées, il est préférable de répondre par "peut-être".

Pour être complets, citons ici trois méthodes qui permettent d'obtenir des résultats relativement fiables.

Tout périphérique dont le support est interchangeable doit disposer d'un mécanisme d'ouverture et de fermeture. On pourra donc tester ce mécanisme et déterminer ainsi si un support est retiré de l'appareil. Malgré tout cette méthode ne permet pas de déterminer si le support retiré n'est pas le même que celui qui vient d'être introduit.

Si un support possède un nom de volume, on pourra lire ce nom pour détecter un changement éventuel. Cette méthode n'a toutefois d'intérêt que si l'utilisateur s'impose comme discipline de ne jamais utiliser le même nom pour deux disquettes différentes.

La méthode utilisée par DOS (pour les lecteurs de disquette) repose sur le fait qu'il faut un certain temps pour changer de support. En ce qui concerne les lecteurs de disquette des PC, on a ainsi pu constater que le disk jockey le plus expert a besoin d'au moins 2 secondes pour changer une disquette. Si deux accès à la disquette se suivent à moins de 2 secondes d'intervalle (c'est d'ailleurs généralement le cas car un accès disquette est rarement isolé), on peut en déduire raisonnablement que la disquette n'a pas été changée entre-temps.

Un octet du bloc de données sert à représenter les trois résultats possibles. La valeur -1 (FFh) signifie "changé", 0 "peut-être" et 1 "inchangé".

Si un changement de support a été effectué et détecté par le périphérique (Bit 11 de l'attribut du driver = 1), il faut à partir de la version 3 de DOS transmettre au système l'adresse d'un buffer en mémoire qui contienne le nom de volume du support précédent. Ce nom doit être écrit sous forme de chaîne ASCII et être terminé par un caractère nul (code ASCII 0).

Fonction 02h : Création d'un bloc de paramètres BIOS (BPB)

Cette fonction n'est réellement définie que dans les drivers de blocs. Dans un driver de caractères, elle se contentera de mettre à 1 le bit TERMINE du champ d'état. Elle est appelée chaque fois que le support a été changé sur le périphérique désigné. Elle transmet à DOS un pointeur sur le BPB du support.

En plus du numéro d'unité logique et du descripteur de support, DOS fournit à cette fonction un pointeur sur un buffer. Si le périphérique présente l'un des formats standard (bit 13 de l'attribut du driver = 0), ce buffer contiendra le premier secteur de la FAT. Si ce n'est pas le cas, ce buffer est dénué de signification.

A partir de la version 3 de DOS, cette fonction devrait aussi lire et sauvegarder le nom de volume du support puisque ce dernier doit être transmis à la fonction 01h en cas de détection de changement de support (bit 11 dans l'attribut du driver de périphérique = 1).

Paramètres d'entrée et de sortie de la fonction 02h d'un driver		
- Paramètres d'entrée		
Adresse	**Contenu**	**Type**
+01h	Numéro de l'unité logique appelée 0=1er unité, 1=2me unité	1 BYTE
+02h	Numéro de la fonction (ici 02h)	1 BYTE
+0Dh	Descripteur de support de l'unité appelée	1 BYTE
+0Eh	Adresse d'un buffer contenant une copie de la table d'allocation des fichiers	1 PTR
- Paramètres de sortie		
Adresse	**Contenu**	**Type**
+03h	Mot d'état	1 WORD
+12h	Adresse du bloc de paramètres BIOS (BPB) de l'unité appelée	1 PTR

Fonction 03h : Lecture directe

Cette fonction peut être implémentée sur les drivers de blocs comme sur les drivers de caractères. Elle permet une communication directe entre un programme d'application et le driver qui peut inclure de cette façon une logique de commande supplémentaire. Grâce à cette fonction, un driver de blocs peut recevoir un bloc de données en provenance du programme d'application pour l'interpréter ensuite à sa guise. Toute forme de communication est ainsi réalisable entre le programme d'application et le driver, pour peu que l'un et l'autre parlent le même langage.

Notez cependant que cette fonction ne peut être appelée par le moyen de la fonction 44h de l'interruption 21h que si le bit 14 de l'attribut du driver est mis à 1. Elle attend qu'on lui transmette le nombre de caractères qui seront envoyés au programme d'application ainsi qu'un pointeur FAR sur le buffer où seront stockés les caractères. Le nombre de caractères lus doit correspondre au nombre de caractères transmis, sauf si une erreur survient qui empêche le traitement de l'intégralité des caractères.

Paramètres d'entrée et de sortie de la fonction 03h d'un driver		
- Paramètres d'entrée		
Adresse	**Contenu**	**Type**
+01h	Numéro de l'unité logique appelée (drivers de blocs uniquement)	1 BYTE
+02h	Numéro de la fonction (ici 03h)	1 BYTE
+0Dh	Descripteur de support du périphérique appelé (drivers de blocs uniquement)	1 BYTE
+0Eh	Adresse du buffer de transmission	1 PTR
+12h	Nombre d'octets à lire	1 WORD
- Paramètres de sortie		
Adresse	**Contenu**	**Type**
+03h	Mot d'état	1 WORD
+12h	Nombre d'octets lus (drivers de blocs)	1 WORD

Fonction 04h : Lecture

Contrairement à la fonction 03h, cette fonction ne sert pas à la communication entre un programme d'application et un driver. Elle est simplement destinée à lire des caractères sur le support. Dans le cas d'un driver de disque dur, ce sont des secteurs qui seront lus tandis que le driver de console CON lira des touches du clavier.

Notez la manière dont est transmis le numéro de secteur. Selon qu'il s'agisse d'un driver de 16 ou 32 bits, le numéro de secteur est mémorisé à l'offset 14h du bloc de données, sous forme de mot simple, ou à l'offset 1Ah sous forme de mot double DWORD. On appelle drivers 32 bits des drivers de blocs qui supportent des périphériques et des partitions de plus de 32 Mo et dont le bit 1 de l'attribut de périphérique est à 1. Dans le cas de ces drivers, le numéro de secteur à l'offset 14h est fixé à FFFFh pour signaler l'exploitation de l'offset 1Ah.

Paramètres d'entrée et de sortie de la fonction 04h d'un driver		
- Paramètres d'entrée		
Adresse	**Contenu**	**Type**
+01h	Numéro de l'unité logique appelée (drivers de blocs uniqt) (0=premier périph., 1=deuxième périph. etc.)	1 BYTE
+02h	Numéro de la fonction (ici 04h)	1 BYTE
+0Dh	Descripteur de média de l'unité logique appelée (drivers de blocs uniqt)	1 BYTE
+0Eh	Adresse du buffer de transmission	1 PTR
+12h	Nombre de secteurs à lire (drivers de blocs) Nombre d'octets à lire (drivers de caractères)	1 WORD
+14h	Numéro du premier secteur (drivers de blocs <32Mo)	1 WORD
+1Ah	Numéro du premier secteur (drivers de bloc >32Mo)	1 DWORD
- Paramètres de sortie		
Adresse	**Contenu**	**Type**
+03h	Mot d'état	1 WORD
+12h	Nombre de secteurs lus (drivers de blocs) Nombre d'octets lus (drivers de caractères)	1 WORD

Fonction 05 h : Lecture d'un caractère sans retrait du buffer

Cette fonction ne concerne que les drivers de caractères.

Dans un driver de blocs, elle se contentera de mettre à 1 le bit TERMINE.

Elle permet à DOS de tester si d'autres caractères sont déjà disponibles. Si tel est le cas, le bit OCCUPE doit être mis à 0 et le caractère suivant doit être transmis à DOS. Il importe de ne pas retirer le caractère lu du buffer, de façon qu'un appel ultérieur à une fonction de lecture le renvoie encore. Si aucun caractère n'est disponible, le bit OCCUPE doit être mis à 1. Cette fonction est essentiellement utilisée pour gérer le clavier.

Paramètres d'entrée et de sortie de la fonction 05h d'un driver		
- Paramètres d'entrée		
Adresse	**Contenu**	**Type**
+02h	Numéro de la fonction (ici 05h)	1 BYTE
- Paramètres de sortie		
Adresse	**Contenu**	**Type**
+03h	Mot d'état	1 WORD
+1Dh	Le caractère lu	1 BYTE

Fonction 06h : Test de l'état d'entrée

Comme la précédente, cette fonction ne concerne que les drivers de caractères. Elle devra donc se contenter de mettre à 1 le bit TERMINE dans un driver de blocs.

DOS appelle cette fonction pour savoir si des caractères sont disponibles, c'est-à-dire s'il existe des caractères en attente de lecture dans le buffer interne du driver. Pour un driver de clavier, cela signifie qu'un ou plusieurs caractères ont été tapés sans avoir encore été lus. Si tel est effectivement le cas, le bit OCCUPE du champ d'état est mis à 0, sinon il est mis à 1.

Si un driver de caractères n'a pas de buffer interne, il devra systématiquement mettre à 1 le bit OCCUPE.

Paramètres d'entrée et de sortie de la fonction 06h d'un driver		
- Paramètres d'entrée		
Adresse	**Contenu**	**Type**
+02h	Numéro de la fonction (ici 06h)	1 BYTE
- Paramètres de sortie		
Adresse	**Contenu**	**Type**
+03h	Mot d'état	1 WORD

Fonction 07h : Vidage du buffer d'entrée

Cette fonction est invoquée pour vider les buffers d'entrée internes des drivers de caractères. Elle n'est pas significative pour un driver de blocs : dans ce dernier cas, elle devra simplement mettre l'indicateur TERMINE à 1.

A la suite de l'appel, tous les caractères lus qui n'ont pas encore été transmis à DOS sont irrémédiablement perdus.

Paramètres d'entrée et de sortie de la fonction 07h d'un driver		
- Paramètres d'entrée		
Adresse	**Contenu**	**Type**
+02h	Numéro de la fonction (ici 07h)	1 BYTE
- Paramètres de sortie		
Adresse	**Contenu**	**Type**
+03h	Mot d'état	1 WORD

Fonction 08h : Ecriture

La fonction d'écriture concerne aussi bien les drivers de blocs que les drivers de caractères. Elle sert à transférer des caractères d'un buffer vers le périphérique souhaité. Si une erreur apparaît lors de la transmission, le champ d'état doit la signaler.

Suivant que le driver est un driver de caractères ou de blocs, la fonction s'attendra à recevoir des paramètres différents. Dans les deux cas cependant, la fonction attendra l'adresse d'un buffer à partir duquel devront être transférés un certain nombre de caractères. Pour un driver de caractères, il faudra encore indiquer le nombre d'octets à transférer.

Pour un driver de blocs, par contre, ce n'est pas le nombre de caractères mais le nombre de secteurs à transférer qu'il faudra indiquer. Un driver de blocs devra également recevoir le numéro de l'unité logique concernée, son descripteur de support ainsi que l'adresse sur le support du premier secteur à transférer.

Notez la différence de traitement des drivers de 16 et 32 bits lorsqu'il s'agit d'indiquer des numéros de secteurs.

Paramètres d'entrée et de sortie de la fonction 08h d'un driver		
- Paramètres d'entrée		
Adresse	**Contenu**	**Type**
+01h	Numéro de l'unité logique appelée (drivers de blocs uniquement)	1 BYTE
+02h	Numéro de la fonction (ici 08h)	1 BYTE
+0Dh	Descripteur de support de l'unité logique appelée (drivers de bloc uniquement)	1 BYTE
+0Eh	Adresse du buffer de transmission	1 PTR

Paramètres d'entrée et de sortie de la fonction 08h d'un driver		
+12h	Nombre de secteurs à écrire (drivers de blocs) Nombre d'octets à écrire (drivers de caract.)	1 WORD
+14h	Numéro du premier secteur (drivers de blocs<32Mo)	1 WORD
+1Ah	Numéro du premier secteur (drivers de blocs>32Mo)	1 DWORD
- Paramètres de sortie		
Adresse	**Contenu**	**Type**
+03h	Mot d'état	1 WORD
+12h	Nombre de secteurs écrits (drivers de blocs) Nombre d'octets écrits (drivers de caractères)	1 WORD

Fonction 09h : Ecriture et vérification

Cette fonction est appelée et utilisée comme la fonction 08h, si ce n'est que les caractères écrits doivent être relus pour être vérifiés.

Sur certains périphériques, surtout s'ils sont orientés caractères comme les écrans ou les imprimantes par exemple, la vérification des données est inutile, soit qu'aucune erreur ne puisse se produire (cas de l'écran), soit que les données ne puissent être vérifiées (cas de l'imprimante).

Paramètres d'entrée et de sortie de la fonction 09h d'un driver		
- Paramètres d'entrée		
Adresse	**Contenu**	**Type**
+01h	Numéro de l'unité logique appelée (drivers de blocs uniquement)	1 BYTE
+02h	Numéro de la fonction (ici 09h)	1 BYTE
+0Dh	Descripteur de support de l'unité logique (drivers de bloc uniquement)	1 BYTE
+0Eh	Adresse du buffer de transmission	1 PTR
+12h	Nombre de secteurs à écrire (drivers de blocs) Nombre d'octets à écrire (drivers de caract.)	1 WORD
+14h	Numéro du premier secteur (drivers de blocs<32Mo)	1 WORD
+1Ah	Numéro du premier secteur (drivers de blocs>32Mo)	1 DWORD
- Paramètres de sortie		
Adresse	**Contenu**	**Type**
+03h	Mot d'état	1 WORD

Paramètres d'entrée et de sortie de la fonction 09h d'un driver		
+12h	Nombre de secteurs écrits (drivers de bloc) Nombre d'octets écrits (drivers de caractères)	1 WORD

Fonction 0Ah : Test de l'état de sortie

Cette fonction n'est appelée qu'à l'intérieur d'un driver de caractères. Dans un driver de blocs, elle devra simplement mettre à 1 le bit TERMINE du champ d'état.

Elle indique à la fonction appelante, par le moyen du bit OCCUPE du champ d'état, si le dernier appel en écriture a déjà été complètement traité ou non. DOS a besoin de cette information pour savoir si la fonction d'écriture est prête à recevoir un nouvel appel.

Paramètres d'entrée et de sortie de la fonction 0Ah d'un driver		
- Paramètres d'entrée		
Adresse	**Contenu**	**Type**
+02h	Numéro de la fonction (ici 0Ah)	1 BYTE
- Paramètres de sortie		
Adresse	**Contenu**	**Type**
+03h	Mot d'état (Le bit OCCUPE doit être mis à 1 si la dernière sortie n'est pas encore terminée)	1 WORD

Fonction 0Bh : Vidage du buffer de sortie

Cette fonction n'est également utilisée que sur les drivers de caractères. Dans un driver de blocs, elle devra donc simplement mettre à 1 le bit TERMINE du le champ d'état.

Lorsqu'elle est invoquée, elle vide tous les buffers de sortie, même si ceux-ci contiennent encore des caractères en attente qui seront alors perdus.

Paramètres d'entrée et de sortie de la fonction 0Bh d'un driver		
- Paramètres d'entrée		
Adresse	**Contenu**	**Type**
+02h	Numéro de la fonction (ici 0Bh)	1 BYTE

Paramètres d'entrée et de sortie de la fonction 0Bh d'un driver		
- Paramètres de sortie		
Adresse	**Contenu**	**Type**
+03h	Mot d'état	1 WORD

Fonction 0Ch : Ecriture directe

La fonction d'écriture directe est utilisée aussi bien dans les drivers de de caractères que dans les drivers de blocs.

Elle est la contrepartie exacte de la fonction 03h. Alors que cette dernière permettait aux programmes d'application de lire des caractères sur le driver, la fonction 0Ch effectue un transfert en sens inverse. Un programme d'application peut ainsi écrire directement sur le périphérique après avoir fait part de ses intentions par la fonction 03h.

La structure de la communication est à l'initiative du driver car DOS n'impose aucune norme dans ce domaine.

Le driver doit signaler le support de cette fonction par le bit 14 de l'attribut de périphérique. Ce n'est que si ce bit est égal à 1 qu'un programme d'application peut y faire appel par l'intermédiaire de la fonction 44h de DOS.

Pour ce qui est des paramètres d'entrée et de sortie, cette fonction est identique à la fonction 03h

Paramètres d'entrée et de sortie de la fonction 0Ch d'un driver		
- Paramètres d'entrée		
Adresse	**Contenu**	**Type**
+01h	Numéro de l'unité logique appelée (drivers de blocs uniquement)	1 BYTE
+02h	Numéro de la fonction (ici 0Ch)	1 BYTE
+0Dh	Descripteur de support de l'unité logique appelée (drivers de blocs uniquement)	1 BYTE
+0Eh	Adresse du buffer de transmission	1 PTR
+12h	Nombre de caractères à écrire	1 WORD
- Paramètres de sortie		
Adresse	**Contenu**	**Type**
+03h	Mot d'état	1 WORD
+12h	Nombre de caractères écrits	1 WORD

Fonction 0Dh : Ouverture

La fonction d'ouverture est exploitée aussi bien par les drivers de blocs que par les drivers de caractères. Elle n'est appelée que si le bit 11 de l'attribut de périphérique est à 1. Sa tâche sera différente suivant qu'elle fait partie d'un driver de caractères ou de blocs.

Dans un driver de blocs, la routine en question est appelée lors de chaque ouverture de fichier. Elle permet de déterminer combien de fichiers sont ouverts sur l'unité logique dont le numéro a été précisé. Il faut être prudent à ce sujet car les programmes qui ont recours aux fonctions FCB ont quelque peu tendance à ne jamais refermer les fichiers ouverts. Ce problème peut toutefois être résolu en considérant qu'aucun fichier ne reste ouvert après un changement de support. La méthode n'est cependant d'aucun secours pour les supports inamovibles tels qu'un disque dur par exemple.

Dans un driver de caractères, la fonction d'ouverture peut servir à envoyer une chaîne d'initialisation avant transmission des caractères au périphérique. Elle se révèle souvent très utile, notamment pour la communication avec une imprimante.

Il faut évidemment éviter de définir la chaîne d'initialisation à l'intérieur du driver. Il vaut mieux la transmettre au driver par le programme d'application, par exemple à l'aide de la fonction IOCTL de l'interruption 21h qui appelle la fonction 03h du driver.

La fonction d'ouverture présente aussi un autre intérêt car elle permet d'empêcher que deux opérations (à l'intérieur d'un réseau ou en fonctionnement multitâche) n'accèdent simultanément au même périphérique, ce qui risquerait naturellement d'entraîner un certain désordre.

Notez que la fonction d'ouverture n'est jamais appelée pour les périphériques CON, PRN et AUX qui restent toujours ouverts.

Paramètres d'entrée et de sortie de la fonction 0Dh d'un driver		
- Paramètres d'entrée		
Adresse	**Contenu**	**Type**
+01h	Numéro de l'unité logique appelée (drivers de bloc uniquement)	1 BYTE
+02h	Numéro de la fonction (ici 0Dh)	1 BYTE
- Paramètres de sortie		
Adresse	**Contenu**	**Type**
+03h	Mot d'état	1 WORD

Fonction 0Eh : Fermeture du périphérique

Cette fonction est l'opposée de la fonction 0Dh. Elle aussi n'est supportée qu'à partir de la version 3 de DOS et uniquement si le bit 11 de l'attribut de périphérique est à 1.

Dans un driver de blocs, elle est appelée chaque fois qu'un fichier a été fermé. Elle permet donc de contrôler facilement si on n'a pas tenté de fermer un fichier qui n'a pas été ouvert.

Dans un driver de caractères, la fonction est utile dans la mesure où elle permet d'envoyer une chaîne de clôture quand la sortie sur un périphérique est terminée. Cette chaîne pourra par exemple être un simple Form Feed pour une imprimante. Comme pour la fonction 0Dh, il faut veiller à ne pas affecter de valeur à cette chaîne à l'intérieur du driver. Le mieux est d'initialiser la chaîne par le programme d'application qui appellera à cet effet la fonction IOCTL.

Paramètres d'entrée et de sortie de la fonction 0Eh d'un driver		
- Paramètres d'entrée		
Adresse	**Contenu**	**Type**
+01h	Numéro de l'unité logique appelée (drivers de bloc uniquement)	1 BYTE
+02h	Numéro de la fonction (ici 0Eh)	1 BYTE
- Paramètres de sortie		
Adresse	**Contenu**	**Type**
+03h	Mot d'état	1 WORD

Fonction 0Fh : Détection d'un support amovible

Cette fonction n'est appelée que si le bit 11 de l'attribut de périphérique est à 1. Comme elle ne concerne que les drivers de blocs, elle devra simplement mettre à 1 le bit TERMINE du champ d'état dans le cadre d'un driver de caractères.

La seule tâche de cette fonction consiste à indiquer à l'appelant si le support placé dans le périphérique concerné peut être changé ou non. La fonction 0Fh répond à cette question en mettant à 1 ou à 0 le bit OCCUPE du champ d'état.

Paramètres d'entrée et de sortie de la fonction 0Fh d'un driver		
- Paramètres d'entrée		
Adresse	**Contenu**	**Type**
+01h	Numéro de l'unité logique appelée (drivers de bloc uniquement)	1 BYTE
+02h	Numéro de la fonction (ici 0Fh)	1 BYTE
- Paramètres de sortie		
Adresse	**Contenu**	**Type**
+03h	Mot d'état (Le bit OCCUPE doit être mis à 0 si le support est amovible)	1 WORD

Fonction 10h : Sortie jusqu'à saturation

Cette fonction ne peut être supportée que par les drivers de caractères. Dans un driver de blocs, elle se limitera à mettre à 1 le bit TERMINE du champ d'état. Elle n'est appelée que si le bit 13 de l'attribut est à 1.

Son rôle est de transmettre des caractères vers un périphérique jusqu'à ce que ce dernier indique qu'il est saturé c'est-à-dire qu'il ne peut plus traiter de caractère. Elle est particulièrement bien adaptée à la réalisation de spoolers pour effectuer des impressions en arrière-plan pendant qu'un programme s'exécute au premier plan. Le fait que tous les caractères à transmettre ne puissent être envoyés au périphérique concerné ne constitue pas une erreur mais un mode de fonctionnement normal. Il faut indiquer en plus de l'adresse du buffer le nombre des caractères à transmettre. Lorsque le périphérique indique qu'il est saturé, le nombre de caractère correctement transmis est reporté dans un champ du paramètre de sortie, et la fonction retourne à l'appelant.

Paramètres d'entrée et de sortie de la fonction 10h d'un driver		
Adresse	**Contenu**	**Type**
Paramètres d'entrée		
+02h	Numéro de la fonction (ici 10h)	1 BYTE
+0Eh	Adresse du buffer de transmission	1 PTR
+12h	Nombre d'octets à écrire	1 WORD
Paramètres de sortie		
Adresse	**Contenu**	**Type**
+03h	Mot d'état	1 WORD
+12h	Nombre d'octets écrits	1 WORD

Fonction 17h : Lecture du format d'une unité logique

Cette fonction n'est exploitée que par les drivers de blocs et elle n'est pas obligatoire. Lorsqu'elle est présente, le bit 6 de l'attribut de périphérique doit être à 1 pour que DOS soit informé.

La fonction n'est utilisable qu'en liaison avec le driver de DOS appelé DRIVER.SYS qui permet de faire fonctionner un lecteur de disquette avec deux formats différents. Le rôle de la fonction est d'indiquer à l'appelant lequel des deux formats est présentement actif.

Le code de l'unité logique transmis comme paramètre d'entrée n'est pas indispensable car par définition il ne peut exister qu'un seul driver dans le système dont le lecteur soit commutable.

Paramètres d'entrée et de sortie de la fonction 17h d'un driver		
- Paramètres d'entrée		
Adresse	**Contenu**	**Type**
+01h	Numéro de l'unité logique appelée (0=1re unité, 1=2me unité, etc)	1 BYTE
+02h	Numéro de la fonction (ici 17h)	1 BYTE
- Paramètres de sortie		
Adresse	**Contenu**	**Type**
+01h	Code d'état 0 = le lecteur indiqué n'est pas commutable 1 = premier format actif 2 = deuxième format actif	1 BYTE
+03h	Mot d'état	1 WORD

Fonction 18h: Indication de format alternatif pour une unité logique

Cette fonction symétrique de la précédente permet à DOS d'indiquer au driver que la lecteur géré peut être désigné par un autre caractère sous un autre format.

Paramètres d'entrée et de sortie de la fonction 18h d'un driver		
- Paramètres d'entrée		
Adresse	**Contenu**	**Type**
+01h	Caractère de désignation alternative du lecteur	1 BYTE
+02h	Numéro de la fonction (ici 18h)	1 BYTE

Paramètres d'entrée et de sortie de la fonction 18h d'un driver		
- Paramètres de sortie		
Adresse	**Contenu**	**Type**
+03h	Mot d'état	1 WORD

25.4. Driver d'horloge

Le driver d'horloge est un type tout à fait particulier de driver de périphérique. C'est en effet un driver de caractères dont la seule occupation est d'échanger la date et l'heure avec DOS. Au moment du lancement du système DOS charge un driver d'horloge appelé $CLOCK. Ce driver pourrait porter un autre nom car il n'est pas identifié par DOS d'après son nom mais par le fait que le bit 2 de l'attribut de périphérique est égal à 1. Le bit 15 doit également être égal à 1 car il s'agit d'un driver de caractères. Le driver lui-même est appelé par les fonctions 2Ah à 2Dh de l'interruption 21h de DOS qui lisent et règlent la date ou l'heure par son intermédiaire. En plus de la fonction 00h (pour l'initialisation) le driver doit supporter les fonctions 04h et 08h. En cas d'appel de la fonction 04h (Lecture), la date et l'heure courantes sont transmises à DOS par le driver. Inversement DOS peut fixer une nouvelle date et une nouvelle heure en appelant la fonction 08h (Ecriture). Pour l'une et l'autre fonction la date et l'heure sont transmises dans un buffer de 6 octets. Les données figurant dans le bloc de données correspondent à celles d'un driver de caractères.

Structure du bloc de données pour transmettre la date et l'heure à un driver d'horloge		
Adresse	**Contenu**	**Type**
+00h	Nombre de jours depuis le 1/01/1980	1 WORD
+02h	Minutes	1 BYTE
+03h	Heures	1 BYTE
+04h	Centièmes de seconde	1 BYTE
+05h	Secondes	1 BYTE
Longueur : 6 octets		

Le mode de transmission de la date peut sembler a priori quelque peu curieux. On s'attendrait à une communication séparée de l'année, du mois et du jour. Au lieu de cela c'est le nombre de jours écoulés depuis le 1er janvier 1980 qui est utilisé sous la forme d'un nombre de 16 bits. Pour reconvertir ce nombre en une date de forme classique année, mois et jour, il faut avoir recours à une formule relativement complexe car elle doit notamment prendre en compte les années bissextiles. Pour lire et fixer l'heure, un driver d'horloge utilisera normalement les fonctions 00h et 01h de l'interruption 1Ah du BIOS. Les possesseurs d'AT devraient être intéressés par le dévelop-

pement d'un nouveau driver d'horloge puisque leur ordinateur dispose, comme vous le savez sans doute, d'une horloge temps réel à piles. Dans ce cas les fonctions 00h et 01h de l'interruption 1Ah n'interviennent que sur un compteur de temps qui est une simple variable de programme. La véritable horloge temps réel reste hors d'atteinte. Si la date et l'heure sont réglées à l'aide de la fonction 08h du driver, elles ne seront pas enregistrées définitivement car elles seront oubliées dès que l'ordinateur sera éteint ou dès que le système sera relancé. Il y aurait donc ici matière à développer un nouveau driver d'horloge qui n'accéderait pas à la date et à l'heure par les fonctions 0 et 1 de l'interruption 1Ah mais utiliserait les fonctions 02h à 05h de cette interruption, qui, elles, permettent d'intervenir sur la véritable horloge. En fait cette idée a bien été exploitée par Microsoft : à partir de la version 3.0 de DOS le driver prédéfini $CLOCK accède vraiment à l'horloge.

25.5. Appel d'un driver de périphérique par DOS

Après avoir décrit les différentes fonctions d'un driver de périphérique, nous allons pouvoir nous lancer dans le développement d'un driver maison. Nous consacrerons d'abord notre attention à la séquence des opérations exécutées avant et après appel d'une fonction d'un driver.

Le premier événement de la chaîne est un appel à une fonction de l'interruption 21h de DOS qui touche de près ou de loin aux entrées-sorties de caractères.

Le fait d'invoquer une telle fonction peut avoir pour conséquence toute une série d'autres appels qui se traduisent par un grand nombre d'accès en lecture/écriture.

C'est ce qui se passe par exemple avec la fonction 3Dh de DOS qui permet d'ouvrir un fichier dans n'importe quel sous-répertoire. Pour ouvrir un fichier, DOS doit commencer par le retrouver. A cette fin la fonction devra charger la FAT et même dans certains cas examiner toute une série de répertoires. A chacun de ces accès dont une partie se déroulera par l'intermédiaire de l'interruption 21h, DOS devra déterminer le périphérique qui doit exécuter les lectures ou écritures de caractères. Après cela, DOS mettra en place, à l'intérieur d'une zone de mémoire réservée à cet effet, un bloc de données qui recevra les informations nécessitées par la fonction de driver.

En ce qui concerne les fichiers, DOS doit encore convertir le numéro de l'enregistrement à traiter, inexploitable par une fonction de driver, en un numéro de secteur logique. C'est alors que commence l'appel de la fonction du driver. La routine de stratégie du driver est déclenchée en premier et reçoit l'adresse du bloc de données qui vient d'être mis en place. C'est ensuite la routine d'interruption du driver qui s'exécute. Elle sauvegarde tous les registres et isole dans le bloc de données le numéro de la fonction à appeler, dont elle déclenche aussitôt l'exécution.

Si le driver appelé est un driver de caractères, la fonction appelée doit simplement envoyer au matériel les caractères à transférer ou lui réclamer les caractères à lire.

S'il s'agit d'un driver de blocs (de type mémoire de masse, par exemple lecteur de disquette ou disque dur), le numéro de secteur logique devra être converti en une adresse physique préalablement à tout accès en lecture ou écriture.

Cette adresse physique pourra se composer par exemple d'un numéro de tête, d'un numéro de piste et d'un numéro de secteur. Une fois que la fonction de lecture/écriture a été exécutée, la fonction du driver devra inscrire dans le champ d'état le résultat de son travail pour le communiquer à la fonction appelante.

Le contenu de l'ensemble des registres peut alors être restauré et le contrôle est rendu à la fonction appelante. Nous voilà revenus à l'intérieur de l'interruption 21h qui, suivant le résultat de la fonction de driver, met à 1 ou à 0 l'indicateur de retenue et charge un code d'erreur dans le registre AX.

En tant que premier et dernier intervenant dans le déroulement des opérations, l'interruption rend ensuite le contrôle à la fonction qui l'avait appelée.

25.6. Accès direct aux drivers de périphériques : IOCTL

Au cours de ce chapitre, la fonction IOCTL a été évoquée plusieurs fois sous la forme de la fonction 44h de DOS. C'est à elle que nous allons nous intéresser maintenant car elle représente, au-delà des méthodes présentées au début de ce chapitre, une autre possibilité de communiquer avec un driver de périphérique. Elle n'est toutefois supportée que par les drivers qui dans leur attribut de périphérique présentent un bit IOCTL égal à 1 (bit 14).

La fonction IOCTL n'est elle-même que l'une des nombreuses fonctions qui sont susceptibles d'être appelées par l'interruption 21h du DOS. Son numéro de fonction est 44h. Elle offre toute une série d'options qui peuvent être divisées en trois groupes : configuration du driver, transmission de données et état du driver. Le numéro de chaque option (numéro de fonction IOCTL) doit toujours être transmis à la fonction 44h par l'intermédiaire du registre AL. Après appel de la fonction, l'indicateur de retenue signale la bonne exécution de la fonction. S'il est à 1, une erreur s'est produite dont le code est placé dans le registre AX.

Pour tester l'état d'un driver de caractères (l'état d'un driver de blocs n'est pas directement détectable), on dispose des fonctions IOCTL 06h et 07h. La fonction 06h permet de déterminer si le driver peut recevoir des données alors que la fonction 07h indique l'état de sortie du driver, autrement dit si le driver peut envoyer des données. Les deux fonctions attendent en AL leur numéro et en BX un handle ou numéro d'accès. Par conséquent, lorsqu'on souhaite déterminer l'état d'un driver de caractères, il convient d'ouvrir tout d'abord un handle à l'aide de la fonction "Ouvrir handle" en spécifiant comme "nom de fichier" le nom du driver de caractères concerné.

Après exécution les deux fonctions renvoient en AL la valeur FFh si le driver est prêt. La valeur 0 signale au contraire que le driver est encore occupé.

Les fonctions IOCTL 02h et 03h servent à échanger des données avec un driver de caractères. Comme paramètres elles attendent également un handle en BX mais il faut aussi disposer en CX le nombre d'octets à transférer et en DS:DX une adresse de buffer. La fonction IOCTL 02h transfère le nombre de caractères spécifié depuis le driver dans le buffer indiqué du programme appelant. A l'issue de l'appel, le registre AX contient le nombre d'octets transférés.

La fonction IOCTL 03h transfère à l'inverse les données du buffer spécifié vers le driver. Le registre AX contient également, après exécution, le nombre d'octets transférés.

Ces deux fonctions permettent de construire un driver de telle façon qu'un programme d'application puisse lui transmettre certaines instructions, avec les paramètres appropriés, à l'aide de la fonction 03h. Le driver exploitera alors chaque instruction et exécutera les calculs correspondants. S'il doit fournir en réponse certaines données au programme d'application, celles-ci pourront être réclamées par le programme utilisateur en appelant la fonction 02h.

Les fonctions IOCTL 04h et 05h ont la même tâche que les fonctions 02h et 03h. Il existe cependant une différence de principe car les fonctions 04h et 05h n'appellent pas des drivers de caractères mais des drivers de blocs. C'est pourquoi ces fonctions ne s'attendent plus à recevoir en BX un handle. Il faut mettre en BL la désignation de l'unité logique invoquée (0=A, 1=B, etc...). Pour le reste, ces deux fonctions sont identiques aux fonctions IOCTL 02h et 03h.

Les deux dernières options de la fonction IOCTL, les fonctions IOCTL 00h et 01h servent à fixer et à tester l'état d'un driver de caractère. Pour tester cet état avec la fonction IOCTL 00h, il suffit de mettre dans le registre AH le numéro de la fonction de DOS (44h), dans le registre AL le numéro de l'option (00h) et dans le registre BX le numéro d'un handle correspondant au driver souhaité. A l'issue de l'appel, le registre DX contiendra l'état du driver. Le bit 7 indique si le handle transmis correspond à un fichier ou à un driver de caractères. Dans ce dernier cas, ce bit aura la valeur 1. Le bit 14 indique si les fonctions IOCTL 02h et 03 peuvent émettre ou réceptionner des caractères à destination ou en provenance du driver. Si le driver supporte ces fonctions, le bit 14 sera à 1. Le bit 5 indique si le driver est exploité en mode Raw (bit 5 = 1) ou en mode Cooked (bit 5 = 0). Les bits 0 à 3 indiquent de quel type de driver il s'agit. Si le bit 3 = 1, il s'agit du driver d'horloge. Le bit 2 indique s'il s'agit du driver NUL, le bit 1 si le driver est le driver de sortie CON et le bit 0 s'il s'agit du driver d'entrée CON.

Les bits ont la même signification pour la fonction IOCTL 01h qui permet de fixer l'état d'un driver de caractère. La seule information pertinente à cet égard est le choix du mode Raw ou Cooked. C'est pourquoi on procède généralement de la façon suivante : on appelle tout d'abord d'appeler la fonction IOCTL 00h pour charger en DX l'état du driver, puis on manipule le bit 5 pour le conformer au mode d'exploitation souhaité et on termine en invoquant la fonction 01h pour entériner le réglage.

Les interruptions 25h et 26h

La dernière possibilité de communication avec un driver est fournie par les interruptions 25h et 26h. Ne sont concernés dans ce cas que les drivers de blocs. Les interruptions citées servent à lire (INT 25h) ou à enregistrer (INT 26h) des secteurs déterminés d'une unité de mémoire de masse. DOS ne convertit pas les numéros de secteurs qui lui sont transmis, les accès se font donc exactement sur les secteurs spécifiés. Il sera intéressant d'appeler cette interruption chaque fois qu'il faudra accéder aux structures de données d'une mémoire de masse, c'est-à-dire à la FAT ou à un répertoire. Ces deux interruptions permettent également d'exploiter des disquettes qui tout en étant lisible par le lecteur utilisé ont été formatées sous un autre système d'exploitation et présentent de ce fait une structure logique différente de celle en vigueur sous DOS. Les deux interruptions sont conçues pour recevoir les mêmes paramètres. Il faut mettre le numéro d'unité logique en AL (A = 0, B = 1, etc...), le nombre de secteurs à lire ou à enregistrer en CX, le numéro du premier secteur appelé en DX, une adresse de buffer en DS:BX. A l'issue de l'interruption, l'indicateur de retenue sera égal à 0 si l'accès au support s'est fait sans problème. Sinon l'indicateur de retenue sera mis à 1 et un code d'erreur figurera dans le registre AX.

A partir de la version 4.0 de DOS, le mode de fonctionnement des deux interruptions a dû être quelque peu modifié car si DX contient un numéro de secteur à 16 bits il n'est pas possible d'adresser plus de 65536 secteurs et de franchir la barrière des 32 Mo. Pour accéder à des volumes de grande taille, il faut désormais placer en DS:BX un pointeur qui référence un bloc de données selon le schéma suivant :

Structure du bloc de données à l'appel des interruptions 25h et 26h sous DOS 4.0		
Adresse	**Contenu**	**Type**
+00h	Numéro du premier secteur	1 DWORD
+04h	Nombre de secteurs	1 WORD
+06h	Pointeur sur buffer	1 PTR
Longueur: 0Ah (10 octets)		

En même temps il faut mettre le nombre -1 (FFFFh) dans CX pour que DOS soit informé de l'utilisation du nouveau schéma. Tant que les secteurs adressés sont situés en deçà de 32 Mo, l'ancienne structure peut être conservée même pour les volumes qui dépassent au total 32Mo.

25.7. Conseils pour développer un driver

Le développement d'un driver de périphérique pose le difficile problème de sa mise au point, autrement dit de son débogage. Il faut savoir qu'un driver de périphérique est automatiquement chargé à une adresse de la mémoire fixée par DOS et que le programmeur ignore. Par ailleurs il n'est pas possible de tester un nouveau driver de console CON avec le programme DEBUG car ce débogueur utilise lui-même ce driver pour la sortie et l'entrée de caractères.

C'est pourquoi nous vous conseillons, après avoir programmé le driver proprement dit, de développer un petit programme de test qui appellera les différentes fonctions du driver exactement comme DOS mais sans intégrer le driver dans DOS. L'avantage sera bien sûr que tout se déroulera sous votre contrôle et que vous pourrez suivre toute l'opération à l'aide d'un débogueur. En tout cas il est absolument indispensable de n'intégrer dans le système un nouveau driver (surtout s'il s'agit d'un driver de blocs) qu'après l'avoir testé de façon approfondie. Avec les mémoires de masse gérées par des drivers de bloc, il peut arriver, faute de précautions, qu'un défaut de fonctionnement du driver occasionne une perte de données considérable.

Si vous travaillez avec un disque dur, il faut absolument réaliser une disquette de lancement du système avant procéder au démarrage du système avec le nouveau driver. Si votre driver contient la moindre imperfection qui plante le système au moment de son initialisation (ce qui m'est justement arrivé au cours du développement du driver de Console que nous allons maintenant vous présenter), le démarrage du système se bloque et vous n'avez aucun moyen de revenir à DOS. Dans une telle situation, la seule solution est de relancer le système avec une disquette DOS non modifiée pour accéder à nouveau au disque dur et éliminer le driver en développement jusqu'à ce qu'il soit opérationnel.

25.8. Exemples de drivers

Pour mettre en pratique les informations fournies dans ce chapitre, nous allons maintenant vous présenter un driver de chacun des trois types étudiés.

Il s'agira d'un driver de caractères qui présente exactement la même structure qu'un driver de console ordinaire, d'un driver de blocs permettant de mettre en place un disque virtuel de 160 Ko et enfin d'un driver d'horloge qui opère directement sur l'horloge temps réel sur piles d'un AT.

Un driver doit par définition respecter les règles des programmes COM. Il ne doit occuper qu'un seul segment qui regroupe à la fois le code et les données. Par ailleurs il ne doit pas contenir de références directes à des segments. Mais à la différence des programmes COM il doit commencer à l'offset 0h (et non 100h) car une fois en mémoire il n'est pas disposé à la suite d'un PSP.

Voici d'abord le nouveau driver de console CON pour les entrées-sorties sur clavier-écran :

Listing : CONDRV.ASM

```
;*****************************************************************;
;*                          C O N D R V                          *;
;*---------------------------------------------------------------*;
;*   Fonction      : Ce programme constitue un driver de console  *;
;*                   (clavier et écran) standard. Il pourra vous  *;
;*                   servir de base pour réaliser un driver du style*;
;*                   de ANSI.SYS.                                *;
;*---------------------------------------------------------------*;
;*   Auteur        : MICHAEL TISCHER                             *;
;*   Développé le  : 4.08.87                                     *;
;*   Dernière MAJ  : 2.04.92                                     *;
;*---------------------------------------------------------------*;
;*   Assemblage avec : MASM CONDRV;                              *;
;*                     LINK CONDRV;                              *;
;*                     EXE2BIN CONDRV CONDRV.SYS  ou             *;
;*                                                               *;
;*                     TASM CONDRV                               *;
;*                     LINK CONDRV;                              *;
;*                     EXE2BIN CONDRV CONDRV.SYS                 *;
;*---------------------------------------------------------------*;
;*   Appel  : Copier le programme CONDRV.SYS dans le répertoire  *;
;*            racine, ajouter l'instruction DEVICE=CONDRV.SYS dans le*;
;*            fichier CONFIG.SYS et relancer le système.         *;
;*****************************************************************;

code    segment

        assume cs:code,ds:code,es:code,ss:code

        org 0                                   ;Programme sans PSP donc débute
                                                ;à l'offset 0

;== Constantes ==================================================

inst    equ 2               ;Offset champ d'instr. dans bloc de données
status  equ 3               ;Offset champ d'état dans bloc de données
adr_fin equ 14          ;Offset adr. de fin du driver dans bloc de données
nombre  equ 18              ;Offset nombre dans le bloc de données
adr_b   equ 14          ;Offset adresse de buffer dans bloc de données

TA_CLA  equ 20                          ;Taille du buffer du clavier
nmb_ins equ 16                  ;Sont supportées les fonctions 0 à 16

;== Données =====================================================
;-- En-tête du driver de périphérique ---------------------------

        dw -1,-1                        ;Lien avec le driver suivant
        dw 10101000000000011b                   ;Attribut du driver
        dw offset strat         ;Pointeur sur la routine de stratégie
        dw offset intr          ;Pointeur sur la routine d'interruption
        db "CONDRV  "                   ;Nouveau driver de console

;-- Table de branchement pour les différentes fonctions ---------

tab_fct dw offset init                  ;Fonction  0 : Initialisation
        dw offset dummy                 ;Fonction  1 : Test de support
        dw offset dummy         ;Fonction  2 : Création d'un bloc BPB
        dw offset no_sup                ;Fonction  3 : Lecture directe
        dw offset lire                  ;Fonction  4 : Lecture
        dw offset lire_b  ;Fonction  5 : Lecture sans retrait du buffer
        dw offset dummy         ;Fonction  6 : Test de l'état d'entrée
        dw offset del_b_en    ;Fonction  7 : Vidage du buffer d'entrée
        dw offset ecrire                ;Fonction  8 : Ecriture
        dw offset ecrire        ;Fonction  9 : Ecriture avec vérification
        dw offset dummy         ;Fonction 10 : Test de l'état de sortie
        dw offset dummy       ;Fonction 11 : Vidage du buffer de sortie
        dw offset no_sup        ;Fonction 12 : Ecriture directe
        dw offset dummy     ;Fonction 13 : Ouverture (à partir de 3.0)
        dw offset dummy                 ;Fonction 14 : Fermeture
        dw offset dummy             ;Fonction 15 : Support amovible
        dw offset ecrire      ;Fonction 16 : Sortie jusqu'à saturation

db_ptr  dw (?),(?)              ;Adresse du bloc de données transmis

d_cla   dw 0            ;Pointeur sur prochain car. dans buffer clavier
f_cla   dw 0            ;Pointeur sur dernier car. dans buffer clavier
bu_cla  db TA_CLA dup (?)               ;Buffer clavier interne

;== Routines et fonctions du driver =============================

strat   proc far                        ;Routine de stratégie

        mov  cs:db_ptr,bx           ;Range l'adresse du bloc de données
        mov  cs:db_ptr+2,es                 ;dans la variable DB_PTR
        ret                         ;Retourne au programme appelant

strat   endp

;---------------------------------------------------------------

intr    proc far                        ;Routine d'interruption

        push ax                         ;Sauve les registres sur la pile
        push bx
        push cx
        push dx
        push di
        push si
        push bp
        push ds
        push es
        pushf                   ;Range également le registre des indicateurs

        push cs                     ;Fixe le registre de segment de données
        pop  ds                     ;Le code coïncide ici avec les données

        les  di,dword ptr db_ptr    ;Adresse du bloc de données dans ES:DI
        mov  bl,es:[di+inst]          ;Va chercher  le code d'instruction
        cmp  bl,nmb_ins           ;Le code d'instruction est-il autorisé ?
        jle  bc_ok                                      ;OUI --> bc_ok

        mov  ax,8003h                       ;Code "Instruction inconnue"
        jmp  short intr_end             ;Retourne  au programme appelant

        ;-- Le code d'instruction était correct --> exécute l'instruction -

bc_ok:
        shl  bl,1           ;Calcule le pointeur sur la table de branchement
        xor  bh,bh                                          ;Annule BH
        call [tab_fct+bx]                               ;Appelle la fonction
        les  di,dword ptr db_ptr    ;Adresse du bloc de données dans ES:DI

        ;-- Exécution de la fonction terminée ----------------------

intr_end label near

        or   ax,0100h                               ;Met à 1 le bit Terminé
        mov  es:[di+status],ax    ;Sauvegarde le tout dans le champ d'état

        popf                            ;Restaure le registre des indicateurs
        pop  es                             ;Restaure les autres registres
        pop  ds
        pop  bp
        pop  si
        pop  di
        pop  dx
        pop  cx
        pop  bx
        pop  ax

        ret                                 ;Retourne  au programme appelant

intr    endp

;---------------------------------------------------------------

dummy   proc near                       ;Cette routine ne fait rien

        xor  ax,ax                              ;Annule le bit Occupé
        ret                             ;Retourne au programme appelant

dummy   endp

;---------------------------------------------------------------

no_sup  proc near               ;Cette routine est appelée pour toutes les
                                ;fonctions qui sont en fait interdites

        mov  ax,8003h                       ;Erreur : instruction inconnue
        ret                             ;Retour au programme appelant
```

```
!
!no_sup   endp
!
!;-------------------------------------------------------------------
!
!store_c  proc near              ;Sauvegarde un caractère dans le buffer
!                                                  ;clavier interne
!                                          ;Entrée : AL = caractère
!                                  ;        BX = position du caractère
!
!     mov  [bx+bu_cla],al     ;Place le caractère dans le buffer interne
!     inc  bl                          ;Augmente le pointeur sur la fin
!     cmp  bl,TA_CLA                        ;Fin du buffer atteinte ?
!     jne  store_e                                    ;NON --> STORE_E
!
!     xor  bl,bl                      ;Nouvelle fin = début du buffer
!
!store_e:
!     ret                                     ;Retour au programme appelant
!
!store_c  endp
!
!;-------------------------------------------------------------------
!
!lire     proc near              ;Charge un nombre déterminé de caractères
!                                                 ;du clavier dans un buffer
!
!     mov  cx,es:[di+nombre]                      ;Lit nombre de caractères
!     jcxz lire_e                                 ;Teste si égal à 0
!     les  di,es:[di+adr_b]     ;Adresse du buffer de caractère en ES:DI
!     cld                                    ;Si STOSB, comptage incrémental
!     mov  si,d_cla          ;Pointe sur prochain car. dans buffer clavier
!     mov  bx,f_cla          ;Pointe sur dernier car. dans buffer clavier
!
!lire_1:
!     cmp  si,bx             ;Encore des caractères dans buffer clavier ?
!     jne  lire_3                                         ;OUI --> LIRE_3
!
!lire_2:
!     xor  ah,ah                    ;Numéro de la fonction de Lecture = 0
!     int  16h                   ;Appelle l'interruption clavier du BIOS
!     call store_c            ;Sauve les caractères dans le buffer interne
!     cmp  al,0                                ;Teste si code étendu
!     jne  lire_3                                      ;non --> LIRE_3
!
!     mov  al,ah                                    ;Code étendu en AH
!     call store_c                             ;Le sauvegarde également
!
!lire_3:
!     mov  al,[si+bu_cla]                ;Lit caractère du buffer clavier
!     stosb                    ;Transfert dans buffer de la fonction d'appel
!     inc  si                  ;Augmente le pointeur sur prochain caractère
!     cmp  si,TA_CLA                        ;Fin du buffer atteinte ?
!     jne  lire_4                                     ;NON --> LIRE_4
!
!     xor  si,si                     ;Le prochain caractère est le premier
!                                          ;caractère dans le buffer clavier
!
!lire_4:
!     loop lire_1               ;Répète jusqu'à ce que tous caractères lus
!     mov  d_cla,si     ;Sauve la position du prochain caractère dans le
!                                                        ;buffer clavier
!     mov  byte ptr f_cla,bl    ;Sauve pos. du dernier caractère dans le
!                                                        ;buffer clavier
!
!lire_e:
!     xor  ax,ax                                    ;Tout est en ordre
!     ret                                  ;Retour au programme appelant
!
!lire     endp
!
!;-------------------------------------------------------------------
!
!lire_b   proc near                ;Lit le prochain caractère au clavier
!                                          ;mais le laisse dans le buffer
!
!     mov  ah,1                 ;Numéro de fonction pour interruption BIOS
!     int  16h                    ;Appelle l'interruption clavier du BIOS
!     je   lire_b1                              ;Pas de caractère --> LIRE_B1
!
!     mov  es:[di+13],al        ;Inscrit caractère dans le bloc de données
!     xor  ax,ax                                          ;Tout est en ordre
!     ret                                        ;Retour au programme appelant
!
!lire_b1  label near
!
!     mov  ax,0100h             ;Met à 1 le bit occupé (pas de caractère)
!     ret                                        ;Retour au programme appelant
!
!lire_b   endp
!
!;-------------------------------------------------------------------
!del_b_en proc near                                ;Vide le buffer d'entrée
!
!     mov  ah,1                 ;Fonction : encore caractères dans buffer ?
!     int  16h                    ;Appelle l'interruption clavier du BIOS
!     je   del_e                ;Plus de caractère dans le buffer --> FIN
!
!     xor  ah,ah                  ;Fonction : retirer caractère du buffer
!     int  16h                    ;Appelle l'interruption clavier du BIOS
!     jmp  short del_b_en                ;Teste si restent des caractères
!
!del_e:
!     xor ax,ax                                           ;Tout est en ordre
!     ret                                        ;Retour au programme appelant
!
!del_b_en endp
!
!;-------------------------------------------------------------------
!
!ecrire proc near                  ;Affiche un nombre de caractères déterminé
!                                                          ;sur l'écran
!
!     mov  cx,es:[di+nombre]                      ;Lit nombre de caractères
!     jcxz f_ecri                                 ;Teste si égal à 0
!     lds  si,es:[di+adr_b]     ;Adresse du buffer de caractère dans DS:SI
!     cld                                    ;Comptage incrémental si LODSB
!
!     mov  ah,3                                   ;Lit page d'écran courante
!     int  10h                          ;Appelle interruption vidéo du BIOS
!
!     mov  ah,14                ;Numéro de fonction pour interruption du BIOS
!
!ecri_1:
!     lodsb                          ;Place le caractère à afficher dans AL
!     int  10h                         ;Appelle l'interruption vidéo du BIOS
!     loop ecri_1               ;Répète jusqu'à ce que tous caractères sortis
!
!f_ecri:
!     xor ax,ax                                           ;Tout est en ordre
!     ret                                        ;Retour au programme appelant
!
!ecrire endp
!
!;-------------------------------------------------------------------
!init     proc near                                ;Routine d'initialisation
!
!     mov  word ptr es:[di+adr_fin],offset init   ;Fixe l'adresse de fin
!     mov  es:[di+adr_fin+2],cs                                  ;du driver
!
!     xor  ax,ax                                          ;Tout est en ordre
!     ret                                      ;Retourne  au programme appelant
!
!init     endp
!
!;=================================================================
!
!code     ends
!        end
!
```

L'en-tête de ce driver signale clairement qu'il s'agit d'un driver de caractères qui gère aussi bien le périphérique d'entrée standard (le clavier) que le périphérique de sortie standard (l'écran). Si vous mettez à 1 les deux bits appropriés de l'attribut de périphérique, ce driver, une fois intégré dans le système, sera mis en service à chaque appel précédemment aiguillé sur le driver CON. Comme tout autre driver, il dispose naturellement d'une routine de stratégie et d'une routine d'interruption. La première sauvegarde simplement l'adresse du bloc de données transmis dans la variable DB_PTR.

La routine d'interruption, lorsqu'elle est appelée, commence par placer sur la pile les registres qu'elle va modifier, avant de lire dans le bloc de données transmis le numéro de la routine à appeler. Elle examine ensuite si cette fonction est supportée par CONDRV. Si ce n'est pas le cas, on saute directement à la fin de la routine d'interruption où le code d'erreur approprié est enregistré dans le champ d'état du bloc de données transmis. Les registres qui avaient été sauvegardés sur la pile sont alors restaurés et le contrôle est rendu à la fonction de DOS qui avait appelé la routine.

Si par contre la fonction appelée est bien supportée, l'offset de la routine demandée à l'intérieur du driver est lu dans la table TAB_FCT pour permettre son appel. Si vous examinez ladite table, vous constaterez que les noms des routines DUMMY et NO_SUP y apparaissent plusieurs fois. DUMMY est appelé pour toutes les fonctions qui ne peuvent être utilisées que dans un driver de blocs ou qui, plus généralement, ne sont pas définies dans ce driver. C'est pourquoi la seule fonction de la routine DUMMY est de mettre à 0 le registre AX et du même coup le bit OCCUPE dans le champ d'état. Cette disposition est imposée par les fonctions de niveau supérieur de DOS.

La routine NO_SUP est quant à elle appelée à la place de toutes les routines qui ne doivent pas être déclenchées par les fonctions supérieures de DOS parce que CONDRV a indiqué dans son attribut de driver qu'il ne supporte pas ces fonctions.

Examinons maintenant les autres fonctions du driver, dans l'ordre dans lequel elles apparaissent dans le listing.

La première routine que nous trouvons après NO_SUP est STORE_C. Elle n'est pas appelée par les fonctions de rang supérieur du DOS mais par les autres routines à l'intérieur même du driver. Elle sert à sauvegarder un caractère dans le buffer interne du driver. Le driver devrait en principe pouvoir se dispenser d'un tel buffer puisque c'est à travers les fonctions du BIOS, qui dispose lui-même d'un buffer, que le driver lit les caractères du clavier. Le problème est que le BIOS renvoie deux caractères chaque fois que l'utilisateur actionne une touche à code étendu (touches de direction, touches de fonction, etc...). Si les fonctions de rang supérieur de DOS ne réclament de CONDRV que la lecture d'un seul caractère, il ne faut pas pour autant que le second caractère soit perdu. C'est pourquoi il faut le sauvegarder dans un buffer pour qu'il puisse être renvoyé à DOS lors du prochain appel de la fonction de lecture. C'est en cela que consiste la tâche de STORE_C.

La routine suivante est la fonction de lecture à laquelle nous venons de faire allusion. Elle détermine tout d'abord le nombre de caractères à lire en fonction du bloc de données fourni par DOS. Si ce nombre est égal à 0, la routine s'achève sur-le-champ.

C'est alors que démarre une boucle qui sera parcourue autant de fois qu'il y a de caractères à lire. Cette boucle sert à tester s'il reste encore des caractères dans le buffer interne du driver. Si tel est le cas, le prochain caractère y est lu pour être transféré dans le buffer de la fonction appelante. S'il n'y a plus de caractères dans le buffer associé au clavier, on fait appel à la fonction 00h de l'interruption 16h du BIOS 16h pour aller

lire un caractère au clavier . Ce caractère est lui aussi sauvegardé dans le buffer de clavier interne. S'il s'agit d'un code de clavier étendu, il est séparé en deux caractères.

La prochaine étape consiste alors à retirer à nouveau le caractère du buffer de clavier pour le transférer dans le buffer de la fonction appelante. L'ensemble de l'opération est répété jusqu'à ce que tous les caractères réclamés aient été transmis à DOS. La routine est alors terminée.

La fonction suivante, appelée LIRE_B, est également invoquée par les fonctions de rang supérieur de DOS. Elle sert à tester si un caractère a été ou non entré au clavier. Si tel n'est pas le cas, elle met à 1 le bit OCCUPE du champ d'état du bloc de données fourni par DOS avant de retourner à la fonction appelante. Si par contre un caractère déjà saisi n'a pas encore été lu, ce caractère est lu, transmis à la fonction appelante de DOS par l'intermédiaire du bloc de données après quoi le bit OCCUPE est mis à 0. Le caractère en question ne sera pas retranché du buffer de clavier : ainsi il pourra être transmis à DOS dans le cadre d'un appel ultérieur de la fonction de lecture. Pour tester si un caractère est déjà prêt, la fonction LIRE_P utilise la fonction 01h de l'interruption 16h du BIOS.

La fonction appelée DEL_B_EN est également à la disposition des fonctions de rang supérieur du DOS. Elle sert à vider le buffer de clavier. A cet effet, elle lit les caractères de ce buffer, par l'intermédiaire de la fonction 0 de l'interruption clavier du BIOS, jusqu'à épuisement, c'est-à-dire jusqu'à ce que la fonction 01h signale qu'il n'y a plus de caractères à lire. Le rôle de cette fonction est alors terminé et elle revient à la fonction appelante, non sans avoir au préalable mis à 0 le bit OCCUPE.

ECRIRE sert à retirer un nombre donné de caractères d'un buffer transmis par DOS pour les afficher à l'écran. C'est la fonction 0Eh de l'interruption vidéo du BIOS qui est utilisée à cet effet. Une fois que tous les caractères ont été amenés sur l'écran, le bit OCCUPE du champ d'état est mis à 0 et la fonction est terminée. Notez qu'elle intervient également lorsque vous appelez les fonctions DOS d'"Ecriture et vérification". Aucune erreur ne peut en effet se produire lors d'un affichage à l'écran, de sorte que toute vérification est complètement inutile.

La dernière fonction, la routine d'initialisation, est la première appelée par DOS. Il n'est pas nécessaire d'initialiser des variables ou le matériel dans CONDRV. La seule tâche de la routine consiste donc à enregistrer l'adresse où se termine le driver dans le bloc de données transmis. Pour cela elle transmet sa propre adresse de début puisqu'elle ne servira plus après ce premier appel et qu'elle est située à la fin du driver.

Sous sa forme actuelle, ce driver ne présente pas un intérêt pratique considérable puisqu'il vous offre uniquement des fonctions déjàa définies par le driver CON de DOS. Il pourra cependant vous être extrêmement utile par la suite si vous le prenez pour base du développement d'un driver étendu du type ANSI.SYS, afin de pouvoir mieux contrôler la gestion de l'écran.

Vous pourriez par exemple intégrer dans ce driver une gestion de fenêtres complète qui pourrait être appelée directement par tous vos programmes, quel que soit leur langage...

Voici maintenant le listing du driver de blocs qui définit un disque virtuel de 160 Ko :

Listing : RAMDISK.ASM

```
;*****************************************************************;
;*                          R A M D I S K                        *;
;*-------------------------------------------------------------- *;
;*  Fonction       : Ce programme est un driver pour disque virtuel *;
;*                   de 160 Ko                                   *;
;*-------------------------------------------------------------- *;
;*  Auteur         : MICHAEL TISCHER                             *;
;*  Développé le   :  4.08.1987                                  *;
;*  Dernière MAJ   :  2.04.1992                                  *;
;*-------------------------------------------------------------- *;
;*  Assemblage     : MASM RAMDISK;                               *;
;*                   LINK RAMDISK;                               *;
;*                   EXE2BIN RAMDISK RAMDISK.SYS    ou           *;
;*                                                               *;
;*                   TASM RAMDISK                                *;
;*                   LINK RAMDISK;                               *;
;*                   EXE2BIN RAMDISK RAMDISK.SYS                 *;
;*-------------------------------------------------------------- *;
;*  Appel          : Copier le programme SYS dans le répertoire racine,*;
;*                   ajouter l'instruction DEVICE=RAMDISK.SYS dans le  *;
;*                   fichier CONFIG.SYS puis relancer le système.      *;
;*****************************************************************;

code    segment

        assume cs:code,ds:code,es:code,ss:code

        org 0                                   ;Programme sans PSP donc débute
                                                ;à l'adresse d'offset 0

;-- Constantes ---------------------------------------------------

inst    equ 2               ;Offset champ d'instr. dans blocs de données
status  equ 3                 ;Offset champ d'état dans bloc de données
nmb_per equ 13                ;Offset nombre de périphériques supportés
changmt equ 14                              ;Offset support changé ?
adr_fin equ 14          ;Offset adr. de fin du driver dans bloc de données
adr_b   equ 14            ;Offset adr. buffer dans bloc de données
nmb_ins equ 16              ;Sont supportées les fonctions 0 à 16
nombre  equ 18                ;Offset nombre dans bloc de données
bpb_adr equ 18                ;Offset adresse du BPB du support
secteur equ 20                ;Offset premier numéro de secteur
des_per equ 22        ;Offset désignation périphérique du disque virtuel

;-- Données ------------------------------------------------------

prem_o  equ this byte                   ;Premier octet du driver

;-- En-tête du driver de périphérique ----------------------------

        dw -1,-1                        ;Lien avec le driver suivant
        dw 0100100000000000b                    ;Attribut de driver
        dw offset strat             ;Pointeur sur la routine de stratégie
        dw offset intr          ;Pointeur sur la routine d'interruption
        db 1                            ;Un seul périphérique supporté
        db 7 dup (0)                            ;Octets pour le nom

;-- Table de branchement pour les différentes fonctions ----------

tab_fct dw offset init                  ;Fonction  0 : Initialisation
        dw offset med_test              ;Fonction  1 : Test de support
        dw offset get_bpb       ;Fonction  2 : Création d'un bloc BPB
        dw offset lire                  ;Fonction  3 : Lecture directe
        dw offset lire                          ;Fonction  4 : Lecture
        dw offset dummy   ;Fonction  5 : Lecture sans retrait du buffer
        dw offset dummy         ;Fonction  6 : Test de l'état d'entrée
        dw offset dummy       ;Fonction  7 : Vidage du buffer d'entrée
        dw offset ecrire                        ;Fonction  8 : Ecriture
        dw offset ecrire        ;Fonction  9 : Ecrire et vérification
        dw offset dummy       ;Fonction 10 : Test de l'état de sortie
        dw offset dummy      ;Fonction 11 : Vidage du buffer de sortie
        dw offset ecrire                ;Fonction 12 : Ecriture directe
        dw offset dummy     ;Fonction 13 : Ouverture (à partir de 3.0)
        dw offset dummy                 ;Fonction 14 : Fermeture
        dw offset no_rem            ;Fonction 15 : Support amovible ?
        dw offset ecrire      ;Fonction 16 : Sortie jusqu'à saturation

db_ptr  dw (?),(?)              ;Adresse du bloc de données transmis
rd_seg  dw (?)                ;RD_SEG:0000 est le début du disque virtuel

bpb_ptr dw offset bpb,(?)                   ;Reçoit l'adresse du BPB

boot_sec db 3 dup (0)           ;Ici figure normalement une instruction
                              ;de saut à la routine de lancement du système
        db "MITI 1.0"               ;Constructeur et numéro de version
bpb     dw 512                                  ;512 octets par secteur
        db 1                                    ;1 secteur par cluster
        dw 1                          ;1 secteur réservé (Secteur boot)
        db 1                       ;1 Table d'allocation des fichiers (FAT)
        dw 64                       ;64 entrées maxi dans répertoire racine
        dw 320                              ;320 secteurs au total = 160 Ko
        db 0FEh                   ;Descripteur de support (Media Descriptor)
                                    ;(1 face avec 40 pistes de 8 secteurs)
        dw 1                                ;Chaque FAT occupe un secteur

    ;-- Pas de routine de boot car le système ne peut ----------------
    ;-- évidemment pas être chargé à partir d'un disque virtuel -------

vol_name db "RAMDISK    "                           ;Le nom du volume
        db 8                                ;Attribut, nom du volume défini

;-- Routines et fonctions du driver ------------------------------

strat   proc far                                ;Routine de stratégie

    mov  cs:db_ptr,bx               ;Range l'adresse du bloc de données
    mov  cs:db_ptr+2,es                         ;dans la variable DB_PTR

    ret                                 ;Retour au programme appelant

strat   endp

;-----------------------------------------------------------------

intr    proc far                                ;Routine d'interruption

    push ax                             ;Sauve les registres sur la pile
    push bx
    push cx
    push dx
    push di
    push si
    push bp
    push ds
    push es
    pushf                       ;Range également le registre des indicateurs

    push cs                           ;Fixe le registre de segment de données
    pop  ds                            ;Le code coïncide ici avec les données

    les  di,dword ptr db_ptr      ;Adresse du bloc de données dans ES:DI
    mov  bl,es:[di+inst]            ;Va chercher le code d'instruction
    cmp  bl,nmb_ins                   ;Code d'instruction autorisé ?
    jle  bc_ok                                          ;OUI --> bc_ok

    mov  ax,8003h                     ;Code pour "instruction inconnue"
    jmp  short intr_end

    ;-- Code d'instruction correct --> exécute l'instruction ---------

bc_ok:
    shl  bl,1                   ;Calcule pointeur sur table de branchement
    xor  bh,bh                                              ;Annule BH
    call [tab_fct+bx]                           ;Appelle la fonction

    ;-- Exécution de la fonction achevée -----------------------------

intr_end label near
    push cs                           ;Fixe le registre de segment de données
    pop  ds                            ;Le code coïncide ici avec les données

    les  di,dword ptr db_ptr      ;Adresse du bloc de données dans ES:DI
```

```
    or   ax,0100h                          ;Met à 1 le but Terminé
    mov  es:[di+status],ax          ;Sauve le tout dans le champ d'état

    popf                           ;Restaure le registre des indicateurs
    pop  es                              ;Restaure les autres registres
    pop  ds
    pop  bp
    pop  si
    pop  di
    pop  dx
    pop  cx
    pop  bx
    pop  ax
    ret                                          ;Retour au programme appelant

intr    endp

;-------------------------------------------------------------------------

init    proc near                                ;Routine d'initialisation

    ;-- Le code qui suit sera effacé par le disque ---------------------
    ;-- virtuel après l'installation -----------------------------------

    ;-- Détermine la désignation de périphérique du disque virtuel ----

    mov  ah,30h           ;Lit la version du DOS à l'aide de fct. 30(h)
    int  21h                          ;de l'interruption du DOS 21(h)
    cmp  al,3                                 ;Au moins version 3 ?
    jb   prinm                                        ;OUI --> PRINM

    mov  al,es:[di+des_per]          ;Lit la désignation de périphérique
    add  al,"A"                                 ;Conversion en lettre
    mov  mf_per,al              ;Sauvegarde dans message d'installation

prinm:
    mov  dx,offset initm             ;Adresse du message d'installation
    mov  ah,9                  ;Affiche numéro de fonction pour chaîne
    int  21h                          ;Appelle l'interruption du DOS

    ;-- Calcule l'adresse du premier octet à la suite du disque -------
    ;-- virtuel et la fixe comme adresse de fin du driver --------------

    mov  word ptr es:[di+adr_fin],offset ramdisk+8000h
    mov  ax,cs                    ;Le disque virtuel a une taille de 32Ko
    add  ax,2000h                                       ;plus 2 * 64Ko
    mov  es:[di+adr_fin+2],ax
    mov  byte ptr es:[di+nmb_per],1          ;1 périphérique supporté
    mov  word ptr es:[di+bpb_adr],offset bpb_ptr   ;Adresse du pointeur
    mov  es:[di+bpb_adr+2],ds                                    ;BPB

    mov  ax,cs                       ;Segment du début du disque virtuel
    mov  bpb_ptr+2,ds                ;Segment du BPB dans le pointeur BPB
    mov  dx,offset ramdisk                        ;Calcul avec offset 0
    mov  cl,4                               ;Divise l'offset par 16 pour
    shr  dx,cl                        ;la convertir en adresse de segment
    add  ax,dx                          ;Additionne les deux adresses et
    mov  rd_seg,ax                                          ;les range

    ;-- Met en place le secteur de boot --------------------------------

    mov  es,ax                    ;Transférer l'adresse de segment dans ES
    xor  di,di ;Le secteur de boot commence au 1er octet disque virtuel
    mov  si,offset boot_sec     ;Adresse du secteur boot dans la mémoire
    mov  cx,15                  ;Seuls les 15 premiers mots sont utilisés
    rep  movsw           ;Copie le secteur de boot dans le disque virtuel

    ;-- Met en place la table d'allocation des fichiers ----------------

    mov  di,512          ;La FAT commence à l'octet 512 du disque virtuel
    mov  al,0FEh                 ;Ecrit le descripteur de support dans le
    stosb                                          ;premier octet de la FAT
    mov  ax,0FFFFH               ;Sauvegarde dans la FAT les octets 2 et 3
    stosw                                                  ;de la FAT
    mov  cx,236                  ;Les 236 mots restants occupés par la FAT
    inc  ax                                              ;Met AX à 0
    rep  stosw                   ;Met toutes les entrées de la FAT à l'état
                                                           ;non occupé

    ;-- Met en place le répertoire racine avec le nom de volume --------

    mov  di,1024          ;Le répertoire racine commence au 3ème secteur
    mov  si,offset vol_name    ;Adresse du nom de volume dans la mémoire
    mov  cx,6                 ;Le nom de volume a une longueur de 6 mots
    rep  movsw           ;Copie le nom de volume dans le disque virtuel

    mov  cx,1017                 ;Remplit de zéros le reste du répertoire
    xor  ax,ax                                 ;dans les secteurs 2, 3, 4 et 5
    rep  stosw

    xor  ax,ax                                              ;Tout est en ordre
    ret                                          ;Retour au programme appelant

init    endp
;-------------------------------------------------------------------------

dummy   proc near                                ;Cette routine ne fait rien

    xor  ax,ax                                            ;Annule le bit Occupé
    ret                                          ;Retour au programme appelant

dummy   endp

;-------------------------------------------------------------------------

med_test proc near                   ;Le support du disque virtuel ne peut
                                                                ;être changé
    mov  byte ptr es:[di+changmt],1
    xor  ax,ax                                            ;Annule le bit occupé
    ret                                          ;Retour au programme appelant

med_test endp

;-------------------------------------------------------------------------

get_bpb  proc near                       ;Transmet à DOS l'adresse du BPB

    mov  word ptr es:[di+bpb_adr],offset bpb
    mov  word ptr es:[di+bpb_adr+2],ds

    xor  ax,ax                                            ;Annule le bit occupé
    ret                                          ;Retour au programme appelant

get_bpb  endp

;-------------------------------------------------------------------------

no_rem   proc near                   ;Un disque virtuel n'est pas amovible

    mov  ax,20                                            ;Met à 1 le bit occupé
    ret                                          ;Retour au programme appelant

no_rem   endp
;-------------------------------------------------------------------------

ecrire   proc near

    xor  bp,bp                       ;Transmission DOS --> disque virtuel
    jmp  short move                                      ;Copie les données

ecrire   endp

;-------------------------------------------------------------------------

lire     proc near

    mov  bp,1                        ;Transmission disque virtuel --> DOS

lire     endp

;-- MOVE : transfer un nbr de secteurs entre disque virtuel et DOS -----
;-- Entrée    : BP = 0 : transfert DOS/disque virtuel (écriture)
;--                  1 : transfert du disque virtuel à DOS (lecture)
;-- Sortie    : aucune
;-- Registres : AX, BX, CX, DX, SI, DI, ES, DS et INDICATEURS
;-- Infos     : Les informations nécessaires (Nombre, premier secteur)
;--             sont fournies par le bloc de données transmis par DOS

move     proc near

    mov  bx,es:[di+nombre]                  ;Lite le nombre de secteurs
    mov  dx,es:[di+secteur]                  ;Numéro du premier secteur
    les  di,es:[di+adr_b]                  ;Adresse du buffer dans ES:DI

move_1:
    or   bx,bx                                    ;Encore des secteurs à lire ?
    je   move_e                                        ;Plus de secteur --> FIN
    mov  ax,dx                                      ;Numéro de secteur en AX
    mov  cl,5                      ;Calcule le nombre de paragraphes (unités
    shl  ax,cl                                    ;de segment) en multipliant par 32
    add  ax,cs:rd_seg  ;et additionne à début de segment disque virtuel
    mov  ds,ax                                             ;Transfère en DS
    xor  si,si                                      ;L'adresse d'offset est 0
    mov  ax,bx                                ;Nombre de secteurs à lire en AX
    cmp  ax,128                     ;Reste-t-il plus de 128 secteurs à lire ?
    jbe  move_2                                  ;NON --> lire tous les secteurs
    mov  ax,128                                ;OUI --> lire 128 secteurs (64 Ko)
move_2:
    sub  bx,ax                                ;Retranche le nombre de secteur lus
    add  dx,ax                              ;Additionne au prochain secteur à lire
    mov  ch,al                                ;Nombre de secteurs à lire * 256 mots
    xor  cl,cl                        ;Met à 0 l'octet faible du compteur de mots
    or   bp,bp                                                 ;Faut-il lire ?
    jne  move_3                                                ;NON --> MOVE_3
    mov  ax,es                                                 ;Range ES en AX
```

```
;     push ds                                   ;Empile DS
;     pop  es                             ; et le récupère en ES
;     mov  ds,ax                    ;ce qui revient à échanger ES et DS
;     xchg si,di                          ;et échanger SI et DI
;move_3:
;     rep  movsw                ;Copie les données dans le buffer de DOS
;     or   bp,bp                                ;Faut-il lire ?
;     jne  move_1          ;NON --> copier éventuellement d'autres sect.
;     mov  ax,es                                ;Range ES en AX
;     push ds                                   ;Empile DS
;     pop  es                             ; et le récupère en ES
;     mov  ds,ax                    ;ce qui revient à échanger ES et DS
;     xchg si,di                      ;et échanger à nouveau SI et DI
;     jmp  short move_1       ;Copie éventuellement d'autres secteurs
;
;move_e:
;     xor  ax,ax                                ;Tout est en ordre
;     ret                             ;Retour au programme appelant
;
;move      endp
;
;;-- C'est ici que commence le disque virtuel à proprement parler --------
;
;    if ($-prem_o) mod 16                              ;Doit commencer à une
;      org ($-prem_o) + 16 - (($-prem_o) mod 16)        ;adresse de mémoire
;    endif                                               ;divisible par 16
;
;ramdisk  equ this byte
;
;initm    db "**** DISQUE VIRTUEL de 160 Ko comme lecteur "
;mi_per   db "?"
;         db ": installé.",13,10,"(c) 1987, 92 MICHAEL TISCHER$",13,10,10
;
;;-------------------------------------------------------------------------
;
;code      ends
;         end
```

L'armature de base de ce driver ressemble beaucoup à celle du driver CONDRV que nous venons de voir précédemment. Les fonctions supportées ne sont toutefois pas les mêmes puisque ces drivers relèvent de deux types différents.

Intéressons-nous tout d'abord à la première fonction du driver, la routine d'initialisation. Elle est beaucoup plus développée que la routine équivalente du driver CONDRV et elle n'a pas non plus été placée à la fin du driver. Elle demeurera donc en mémoire une fois son exécution terminée, bien qu'elle ne soit plus utilisée. Nous ne tarderons pas à comprendre pourquoi il en est ainsi.

Cette routine lit tout d'abord le numéro de la version de DOS à l'aide de la fonction 30h. Si ce numéro de version est supérieur ou égal à 3, le bloc de données transmis par DOS contiendra la désignation de périphérique du disque virtuel. Dans ce cas ladite désignation sera lue, convertie en une lettre et reportée dans le message d'installation qui est affiché par la fonction 09h de DOS.

La routine calcule ensuite l'adresse où se termine le disque virtuel. Comme la zone de données proprement dite du disque virtuel commence immédiatement après la dernière routine du driver, il faut ajouter 160 Ko à l'adresse où se termine le driver. On envoie ensuite à DOS l'adresse d'une variable (BPB_PTR) qui contient elle-même l'adresse du bloc de paramètres du BIOS décrivant le format du disque virtuel. Ce bloc de paramètres indique ici que le disque virtuel fonctionne avec des secteurs de 512 octets, que chaque cluster ne comporte qu'un seul secteur, qu'il n'y a qu'un secteur réservé (le secteur de boot), qu'il n'existe qu'une seule FAT. Par ailleurs le répertoire racine peut comporter 64 entrées au maximum, le disque virtuel compte 320 secteurs, autrement dit 160 Ko, la FAT occupe un seul secteur et enfin que le descripteur de support vaut FEh, ce qui définit une disquette simple face de 40 pistes de 8 secteurs.

Une fois que ces paramètres ont été reportés dans le bloc de données de DOS, on calcule le segment de la zone de données du disque virtuel. Cette adresse n'est en fait requise qu'à des fins internes et elle n'est pas communiquée à DOS.

Il s'agit à présent de formater le disque virtuel, c'est-à-dire qu'il faut le doter d'un secteur de lancement du système (secteur de boot), d'une FAT (table d'allocation des fichiers) et d'un répertoire racine. C'est maintenant qu'apparaît clairement la raison pour laquelle

la routine d'initialisation n'a pas été placée après la dernière routine du driver, c'est-à-dire au début de la zone de données du disque virtuel. Comme les structures de données indiquées vont se placer dans les premiers secteurs du disque virtuel, la routine INIT s'effacerait donc elle-même si elle se situait à cet endroit, ce qui ne manquerait pas de planter le système.

Le secteur de lancement du système, ou secteur de boot, occupe comme d'habitude l'intégralité du premier secteur du disque mais seuls ses 15 premiers mots sont enregistrés car ce sont les seules informations indispensables à DOS.

Il est évident que le secteur de lancement du système a été installé ici uniquement par souci de compatibilité avec le format habituel. Il n'est pas possible en effet de lancer le système à partir d'un disque virtuel.

La FAT est mise en place dans le second secteur du disque virtuel. Les deux premiers éléments de la table sont constitués par le descripteur de support, les autres éléments sont initialisés à 0 pour signaler qu'aucun cluster n'est encore occupé et que le disque virtue est encore vierge.

La dernière structure de données installée est le répertoire racine qui ne comporte au départ aucune entrée en dehors du nom de volume. Le rôle de la routine d'initialisation est alors achevé et elle rend la main à la fonction appelante.

Examinons maintenant dans l'ordre les autres routines du driver. INIT est suivi de la routine DUMMY qui a la même fonction que dans CONDRV.

Vient ensuite la routine MED_TEST qui n'existe que dans les drivers de blocs et qui indique à DOS si le support géré a été changé. Nous renvoyons bien sûr systématiquement la valeur 1 pour "inchangé".

La routine suivante, GET_BPB a simplement pour objet de fournir à nouveau à DOS l'adresse de la variable qui contient l'adresse du BPB du disque virtuel, comme l'a déjà fait la routine d'initialisation.

La routine suivante s'appelle NO_REM. Elle permet à DOS de déterminer si le support géré (notre disque virtuel en l'occurrence) est amovible ou non. Comme notre support est inamovible, nous mettons à 1 le bit OCCUPE du champ d'état.

Le listing fait alors apparaître les deux fonctions principales du driver qui s'occupent de la lecture et de l'écriture de données. Comme dans CONDRV, c'est la fonction d'écriture ordinaire qui est appelée lorsqu'on invoque la fonction "Ecriture et vérification" car aucune erreur de données n'est possible lors d'un accès à la mémoire vive. Notre routine ne fait d'ailleurs pas grand chose ; elle charge simplement la valeur 0 dans le registre BP, après quoi elle se branche sur la routine MOVE. La routine LIRE fonctionne de la même façon, si ce n'est bien sûr que c'est la valeur 1 qu'elle charge dans le registre BP.

MOVE est une routine élémentaire qui sert à transférer des données. Lorsqu'elle est appelée, le registre BP est simplement utilisé pour indiquer si les données doivent être transférées du disque virtuel vers DOS ou bien en sens inverse. Toutes les autres informations, l'adresse du buffer de DOS, le nombre de secteurs à transférer et le premier secteur à transférer sont communiqués par le bloc de données que lui fournit DOS. Les commentaires du listing décrivent en détail le fonctionnement de cette routine et il n'est donc pas nécessaire que nous y revenions.

Ce disque virtuel peut évidemment être encore agrandi. Si la taille de votre mémoire vive le permet, il pourrait par exemple être intéressant de porter sa capacité à 360 Ko. Les possesseurs d'AT peuvent aussi envisager de loger le disque virtuel au-delà de la limite de 1 Mo. Dans ce cas, la transmission des données entre DOS et le disque virtuel devra s'effectuer par l'intermédiaire de la fonction 87h de l'interruption 15h.

Voici maintenant un dernier exemple de driver de périphérique. Il s'agit cette fois d'un driver d'horloge qui permet d'accéder directement à l'horloge sur piles de l'AT. Grâce à lui, la date et l'heure sont sauvegardées directement dans l'horloge en temps réel lorsque vous appelerez les instructions DATE et TIME du DOS. Inversement, lorsque vous voudrez connaître la date ou l'heure, ces informations seront tirées directement des mémoires de l'horloge en temps réel.

Regardons d'abord le listing de ce driver, que nous décrirons ensuite de façon détaillée.

Listing : HEUREAT.ASM

```
;*****************************************************************;
;*                          H E U R E A T                        *;
;*---------------------------------------------------------------*;
;*   Fonction : Ce programme est un driver d'horloge             *;
;*              permettant à DOS d'accéder à l'horloge sur piles *;
;*              pour toutes les fonctions de date et d'heure     *;
;*---------------------------------------------------------------*;
;*   Auteur       : MICHAEL TISCHER                              *;
;*   Développé le : 04.08.1987                                   *;
;*   Dernière MAJ : 03.03.1992                                   *;
;*---------------------------------------------------------------*;
;*   Assemblage   : MASM HEUREAT;                                *;
;*                  LINK HEUREAT;                                *;
;*                  EXE2BIN HEUREAT HEEREAT.SYS    ou            *;
;*                                                              *;
;*                  TASM HEUREAT                                 *;
;*                  LINK HEUREAT;                                *;
;*                  EXE2BIN HEUREAT HEUREAT.SYS                  *;
;*---------------------------------------------------------------*;
;*   Appel    : Copier le programme .SYS dans le répertoire racine,*;
;*              ajouter l'instruction DEVICE=HEUREAT.SYS dans le  *;
;*              fichier CONFIG.SYS puis relancer le système.      *;
;*****************************************************************;

code    segment

        assume cs:code,ds:code,es:code,ss:code

        org 0                                   ;Programme sans PSP, donc
                                                ;débute à l'offset 0

;-- Constantes ---------------------------------------------------

inst    equ 2               ;Offset champ d'instr. dans bloc de données
status  equ 3               ;Offset champ d'état dans bloc de données
adr_fin equ 14      ;Offset adr. de fin du driver dans bloc de données
nombre  equ 18              ;Offset nombre dans bloc de données
adr_b   equ 14          ;Offset adresse de buffer dans bloc de données

;-- Données ------------------------------------------------------

;-- En-tête du driver de périphérique ----------------------------

        dw -1,-1                        ;Lien avec le driver suivant
        dw 1000000000001000b                    ;Attribut du driver
        dw offset strat           ;Pointeur sur la routine de stratégie
        dw offset intr          ;Pointeur sur la routine d'interruption
        db "$CLOCK    "                 ;Nouveau driver d'horloge

db_ptr  dw (?),(?)              ;Adresse du bloc de données transmis

tb_mois db 31                    ;Table indiquant le nombre de jours
fevrier db 28                                   ;de chaque mois
        db 31,30,31,30,31,31,30,31,30,31

;-- Routines et fonctions du driver ------------------------------

strat   proc far                                ;Routine de stratégie

        mov  cs:db_ptr,bx         ;Range l'adresse du bloc de données dans
        mov  cs:db_ptr+2,es                     ;la variable DB_PTR

        ret                                 ;Retour au programme appelant

strat   endp

;-----------------------------------------------------------------

intr    proc far                                ;Routine d'interruption

        push ax                         ;Sauve les registres sur la pile
        push bx
        push cx
        push dx
        push di
        push si
        push bp
        push ds
```

```
;     push es
;     pushf                          ; y compris le registre des indicateurs
;
;     cld                          ;Fixe le sens d'incrémentation des chaines
;
;     push cs                      ;Fixe le registre de segment de données
;     pop  ds                       ;Le code coïncide ici avec les données
;
;     les  di,dword ptr db_ptr       ;Adresse du bloc de données en ES:DI
;     mov  bl,es:[di+first]            ;Va chercher code d'instruction
;     cmp  bl,4                      ;Faut-il lire l'heure et la date ?
;     je   lire_dh                                  ;OUI --> LIRE_DH
;     cmp  bl,8                      ;Faut-il régler la date et l'heure ?
;     je   ecrt_dh                                  ;OUI --> ECRI_DH
;     or   bl,bl                       ;Faut-il initialiser le driver ?
;     jne  fct_inc                           ;NON --> Fonction inconnue
;
;     jmp  init                                 ;Initialise le driver
;
;fct_inc: mov  ax,8003h                  ;Code pour "instruction inconnue"
;
;     ;-- Exécution de la fonction terminée ----------------------------
;
;intr_end label near
;     or   ax,0100h                                 ;Fixe à 1 le bit Terminé
;     mov  es:[di+status],ax      ;Sauvegarde le tout dans le champ d'état
;
;     popf                              ;Restaure le registre des indicateurs
;     pop  es                                ;Restaure les autres registres
;     pop  ds
;     pop  bp
;     pop  si
;     pop  di
;     pop  dx
;     pop  cx
;     pop  bx
;     pop  ax
;
;     ret                                          ;Retour au programme appelant
;
;intr     endp
;
;;-------------------------------------------------------------------------
;
;lire_dh label near              ;Lite la date et l'heure dans l'horloge
;
;     mov  byte ptr es:[di+nombre],6                ;6 octets transmis
;     les  di,es:[di+adr_b]              ;ES:DI désigne le buffer de DOS
;
;     mov  ah,4                        ;Numéro de la fonction "lire date"
;     int  1Ah                      ;Appelle l'interruption horloge du BIOS
;     call date_ofs          ;Convertit la date en offset après le 1.1.1980
;     stosw                                    ;Sauvegarde dans buffer
;
;     mov  ah,2                       ;Numéro de la fonction "lire l'heure"
;     int  1Ah                      ;Appelle l'interruption horloge du BIOS
;     mov  bl,ch                                ;Range les heures dans BL
;     call bcd_bin                                 ;Convertit en minutes
;     stosb                                    ;Sauvegarde dans buffer
;     mov  cl,bl                                          ;Heure en CL
;     call bcd_bin                              ;Convertit l'heure
;     stosb                                    ;Sauvegarde dans buffer
;     xor  al,al                                ;0 centième de seconde
;     stosb                                    ;Sauvegarde  dans buffer
;     mov  cl,dh                                       ;Secondes en CL
;     call bcd_bin                              ;Convertit les heures
;     stosb                                    ;Sauvegarde dans buffer
;
;     xor  ax,ax                                    ;Tout est en ordre
;     jmp  short intr_end                  ;Retour au programme appelant
;
;;-------------------------------------------------------------------------
;
;ecrt_dh  label near             ;Ecrit la date et l'heure dans l'horloge
;
;     mov  byte ptr es:[di+nombre],6                ;6 octets transmis
;     les  di,es:[di+adr_b]              ;ES:DI désigne le buffer de DOS
;
;     mov  ax,es:[di]              ;Cherche le nombre de jours depuis le
;     push ax                             ;1.1.1980, mémorise ce nombre
;     call ofs_date                             ;et le convertit en date
;     mov  ch,19h                            ;L'année commence par 19..
;     mov  ah,5                       ;Numéro de la fonction "fixer la date"
;     int  1AH                      ;Appelle l'interruption horloge du BIOS
;     mov  al,es:[di+2]              ;Cherche les minutes dans le buffer
;     call bin_bcd                              ;Les convertit en BCD
;     mov  cl,al                                ;et les place dans CL
;     mov  al,es:[di+5]              ;Cherche les secondes dans le buffer
;     call bin_bcd                              ;Les convertit en BCD
;     mov  dh,al                                ;et les placer en DH
;     mov  al,es:[di+3]              ;Cherche les heures dans le buffer
;     call bin_bcd                              ;Les convertit en BCD
;     mov  ch,al                                ;et les place dans CH
;     xor  dl,dl                                          ;Pas d'heure d'été
;     mov  ah,3                       ;Numéro de la fonction "Fixer l'heure"
;     int  1AH                      ;Appelle l'interruption horloge du BIOS
;
;     ;-- Calcule le jour de la semaine ------------------------------
;     xor  dx,dx                                  ;Mot fort pour la division
;     pop  ax                          ;Retire de la pile le nombre de jours
;     or   ax,ax                                   ;Ce nombre est-il 0 ?
;     je   nodiv                                        ;OUI --> pas de division
;     xor  dx,dx                                  ;Mot fort pour la division
;     mov  cx,7                                ;Sept jours dans une semaine
;     div  cx                                        ;Divise AX par 7
;nodiv:
;     add  dl,3                          ;Le 1.1.80 était un Mardi (Jour 3)
;     cmp  dl,8                           ;Sommes-nous Dimanche ou Lundi ?
;     jb   nosomo                      ;NON --> aucune correction nécessaire
;     sub  dl,cl                                    ;Corrige la valeur
;nosomo:
;     mov  al,6                              ;Octet 6 du RTC = jour de la semaine
;     out  70h,al  ;Envoie l'adresse au registre d'adresse du circuit RTC
;     mov  al,dl                                 ;Charge le jour de la semaine en AL
;     out  71h,al             ;Envoie le jour de la semaine au registre de
;                                                    ;données du circuit RTC
;     xor  ax,ax                                        ;Tout est en ordre
;     jmp  intr_end                        ;Retour au programme appelant
;
;;-- OFS_DATE: Convertit en date le nombre de jours depuis le 1.1.1980 --
;;-- Entrée  : AX = Nombre de jours depuis le 1.1.1980
;;-- Sorties : CL = Année, DH = Mois et DL = Jour
;;-- Registres : AX, BX, CX, DX, SI et les indicateurs sont modifiés
;;-- Infos     : La conversion de l'offset se fait en accédant au tableau
;;--             TB_MOIS
;
;ofs_date proc near
;
;     mov  cl,80                                  ;Partir de l'année 1980,
;     mov  dh,01                                        ;du mois de janvier
;an:  mov  bx,365                    ;Nombre de jours dans une année normale
;     test cl,3                             ;Est-ce une année bissextile ?
;     jne  an1                                          ;NON --> an1
;     inc  bl                        ;L'année bissextile compte un jour de plus
;an1: cmp  ax,bx                              ;Encore une année écoulée ?
;     jb   mo                                   ;NON --> calcule le mois
;     inc  cl                                   ;OUI --> augmente l'année
;     sub  ax,bx                ;Retranche le nombre de jours de cette année
;     jmp  short an                             ;Calcule l'année suivante
;
;mo:       mov  bl,28                ;Jours de février pour une année normale
;     test cl,11b                             ;Est-ce une année bissextile ?
;     jne  pasbis2                                      ;NON --> PASBIS2
;     inc  bl                    ;Dans une année bissextile, février a 29 jours
;pasbis2:
;     mov  fevrier,bl                    ;Sauve le nombre de jours en février
;
;     mov  si,offset tb_mois                    ;Adresse de la table des mois
;     xor  bh,bh                          ;Tous les mois ont moins de 256 jours
;mol: mov  bl,[si]          ;Va chercher le nombre de jours dans le mois
;     cmp  ax,bx                                   ;Encore un mois écoulé ?
;     jb   jou                                     ;NON --> calcule le jour
;     sub  ax,bx                          ;OUI --> retranche les jours du mois
;     inc  dh                                         ;Augmente le mois
;     inc  si                    ;SI pointe sur le prochain mois de la table
;     jmp  short mol                             ;Calcule le mois suivant
;
;jou: inc  al                                  ;Le reste + 1 donne le jour
;     call bin_bcd                              ;Convertit le jour en BCD
;     mov  dl,al                               ;Transfère le jour dans DL
;     mov  al,dh                               ;Transfère le mois dans AL
;     call bin_bcd                                    ;Convertit en BCD
;     mov  dh,al                                      ;Ramène en DH
;     mov  al,cl                              ;Transfère l'année en AL
;     call bin_bcd                                    ;Convertit en BCD
;     mov  cl,al                                      ;Ramène en CL
;
;     ret                                      ;Retour au programme appelant
;
;ofs_date endp
;
;;-- BIN_BCD: Convertit un nombre binaire en BCD ------------------------
;;-- Entrée  : AL = valeur binaire
;;-- Sortir  : AL = valeur BCD correspondante
;;-- Registres : AX, CX et les indicateurs sont modifiés
;
;bin_bcd  proc near
;
;     xor  ah,ah                          ;Prépare une division sur 16 bits
;     mov  ch,10                        ;Nous travaillons en système décimal
;     div  ch                                         ;Divise AX par 10
;     shl  al,1                  ;Décale le quotient de 4 chiffres sur la gauche
;     shl  al,1
;     shl  al,1
;     shl  al,1
```

```
;     or   al,ah                                  ;Masque le reste
;     ret                                 ;Retour au programme appelant
;
bin_bcd  endp
;
;:-- DATE_OFS: Convertit une date en nombre de jours depuis le 1.1.1980 -
;:-- Entrée  : CL = Année, DH = Mois et DL = Jour
;:-- Sorties  : AX = Nombre de jours depuis le 1.1.1980
;:-- Registres : AX, BX, CX, DX, SI et FLAGS sont modifiés
;:-- Infos : La conversion de la date se fait avec TB_MOIS
;
date_ofs proc near
;
;     call bcd_bin                        ;Convertit l'année en binaire
;     mov  bl,al                         ;et transfère le résultat en BL
;     mov  cl,dh                             ;Transfère le mois dans CL
;     call bcd_bin                       ;Convertit le mois en binaire
;     mov  dh,al                                  ;et le ramène en DH
;     mov  cl,dl                           ;Transfère le jour dans CL
;     call bcd_bin                       ;Convertit le jour en binaire
;     mov  dl,al                                  ;et la ramène dans DL
;
;     xor  ax,ax                                                ;0 jour
;     mov  ch,bl                                    ;Mémorise l'année
;     dec  bl                               ;Revient une année en arrière
annee:
;     cmp  bl,80                          ;Décompté jusqu'à l'année 1980 ?
;     jb   mois                                 ;OUI --> convertit le mois
;     test bl,11b                        ;Est-ce une année bissextile ?
;     jne  pasbis                                          ;NON --> PASBIS
;     inc  ax                    ;Une année bissextile compte un jour de plus
pasbis:
;     add  ax,365                              ;Ajoute les jours de l'année
;     dec  bl                                  ;Revient une année en arrière
;     jmp  short annee                           ;Traite l'année précédente
;
mois:
;     mov  bl,28                    ;Jours de février dans une année normale
;     test ch,11b                          ;Est-ce une année bissextile ?
;     jne  pasbis1                                        ;NON --> PASBIS1
;     inc  bl                ;Dans une année bissextile, février a 29 jours
pasbis1:
;     mov  fevrier,bl                  ;Sauvegarde dans la table des mois
;     xor  ch,ch                    ;Tous les mois ont moins de 256 jours
;     mov  bx,offset tb_mois                 ;Adresse de la table des mois
mois1:
;     dec  dh                                        ;Diminue le nombre de mois
;     je   jour                  ;Tous les mois ont été calculés --> JOUR
;     mov  cl,[bx]             ;Va chercher le nombre de jours dans le mois
;     add  ax,cx                     ;Additionne au nombre total de jours
;     inc  bx                  ;BX pointe sur le prochain mois de la table
;     jmp  short mois1                          ;Calcule le mois suivant
;
jour:
;     add  ax,dx                                    ;Ajoute le jour actuel
;     dec  ax                               ;Retranche un jour (1.1.80 = 0)
;     ret                                       ;Retour au programme appelant
;
date_ofs  endp
;
;:-- BCD_BIN: Convertit un nombre BCD en nombre binaire ----------------
;:-- Entrée  : CL = valeur BCD
;:-- Sortie  : AL = valeur binaire correspondante
;:-- Registres : AX, CX et les indicateurs sont modifiés
;
bcd_bin  proc near    ;Convertt une valeur BCD dans CL en son équivalent
                                        ;binaire et renvoie le résultat en AL
;
;     mov  al,cl                                    ;Transfère la valeur dans AL
;     shr  al,1          ;Effectue un décalage de 4 chiffres sur la droite
;     shr  al,1
;     shr  al,1
;     shr  al,1                                                   ;Mat AH à 0
;     xor  ah,ah                          ;Nous travaillons en système décimal
;     mov  ch,10                                  ;Multiplie AX par 10
;     mul  ch                                     ;Transfère CL dans CH
;     mov  ch,cl                        ;Met à 0 le quartet fort de CH
;     and  ch,1111b                               ;Additionne AL et CH
;     add  al,ch                            ;Retour au programme appelant
;     ret
;
bcd_bin  endp
;
;:-------------------------------------------------------------------------
;
init     proc near                                  ;Routine d'initialisation
;
;     :-- Le code suivant peut être effacé par DOS après ----------------
;     :-- installation de l'horloge -------------------------------------
;
;     mov  word ptr es:[di+adr_fin],offset init    ;Fixe l'adresse de fin
;     mov  es:[di+adr_fin+2],cs                                  ;du driver
;
;     mov  ah,9                          ;Affiche le message d'installation
;     mov  dx,offset initm                                ;Adresse du texte
;     int  21h                                  ;Appelle l'interruption du DOS
;
;     xor  ax,ax                                          ;Tout est en ordre
;     jmp  intr_end                             ;Retour au programme appelant
;
initm    db 13,10,"**** HEUREAT installé. (c) 1987, 92 by"
         db " Michael TISCHER",13,10,"$"
;
init     endp
;
;:-------------------------------------------------------------------------
;
code     ends
         end
;
```

La structure de base de ce driver ne se distingue de celle des précédents que dans la mesure où les différentes fonctions sont appelées directement, sans passer par une table d'adresses. Comme ce driver ne doit en effet supporter que les fonctions 00h, 04h et 08h, il peut tester directement le numéro de fonction transmis par DOS pour voir s'il s'agit bien d'une de ces fonctions, et signaler une erreur si ce n'est pas le cas. Outre la routine INIT qui fixe simplement l'adresse où se termine le driver, comme dans CONDRV, ce driver ne comporte, du point de vue de DOS, que les fonctions "Lecture de la date et de l'heure" ou "Réglage de la date et de l'heure".

La communication de l'heure s'effectue d'une façon très simple. En lecture, il suffit de prélever l'heure dans les mémoires appropriées de l'horloge, de la convertir du format BCD au format binaire puis de la reporter dans le buffer de DOS. Le réglage s'effectue en sens inverse : les données sont tout d'abord lues dans le buffer de DOS puis converties du format binaire au format BCD et enfin reportées dans les mémoires de l'horloge.

Si cette transmission de données ne pose aucun problème avec ces deux fonctions, c'est que DOS utilise pour indiquer l'heure le même format que l'horloge : heures, minutes et secondes sont codées chacune dans un octet.

Avec la date les choses se compliquent. L'horloge stocke en effet le jour, le mois et l'année dans autant d'octets différents alors que DOS fournit la date sous la forme du nombre de jours écoulés depuis le 1.1.1980. Lors du réglage de la date et de l'heure, il est donc nécessaire de convertir tout d'abord ce nombre en une date au format jour, mois, année. C'est naturellement l'inverse qui se produit lorsqu'est appelée la fonction de lecture. Il faut alors convertir la date fournie par l'horloge en nombre de jours. Essayons donc de suivre le déroulement des opérations.

La routine de conversion commence par l'année 1980 et met à 0 le nombre de jours écoulés depuis le 1er janvier 1980. Elle teste alors si cette année est inférieure à l'année actuelle. Si oui, le nombre de jours que compte cette année est additionné au nombre de jours écoulés, sans oublier naturellement d'ajouter un jour supplémentaire pour toute année bissextile. Cette boucle est répétée jusqu'à ce que l'année actuelle soit atteinte. On calcule ensuite le nombre de jours en février de cette année et on le reporte dans une table qui contient le nombre de jours de chaque mois.

L'étape suivante consiste à ajouter au nombre de jours écoulés le nombre de jours de tout mois inférieur au mois actuel. Une fois le mois actuel atteint, il ne reste plus qu'à additionner le jour actuel au nombre de jours écoulés. C'est ainsi qu'on obtient le nombre de jours écoulés depuis le 1.1.1980.

Cette valeur est alors reportée dans le buffer de DOS et le tout est transmis à DOS.

La conversion du nombre de jours écoulés en une date au format ordinaire se déroule exactement à l'identique mais en sens inverse. On commence tout d'abord par l'année 1980 et on teste si le nombre de jours que compte cette année est inférieur au nombre de jours écoulés. Si tel est le cas, l'année est incrémentée et le nombre de jours de cette année est retranché du nombre de jours écoulés. Cette boucle est répétée jusqu'à ce que le nombre de jours d'une année soit supérieur au nombre de jours écoulés. On calcule alors le nombre de jours en février de cette année et on le reporte dans la table des mois.

Une nouvelle boucle commence avec le mois de janvier, pour tester si le nombre de jours de chaque mois est inférieur ou égal au nombre de jours écoulés. Tant que c'est le cas, le nombre de jours du mois considéré est retranché du nombre de jours écoulés. Lorsque le nombre de jours d'un mois est enfin supérieur au nombre de jours écoulés, la boucle est interrompue. Il ne reste plus alors qu'à augmenter le nombre pour obtenir le jour du mois et compléter ainsi la date.

La dernière opération consiste à convertir au format BCD la date ainsi calculée et à l'enregistrer dans les mémoires de l'horloge.

25.9. Drivers sous forme de programmes EXE

Ces dernières années ont vu apparaître de nombreux drivers présentés sous forme de fichiers EXE. Ces drivers peuvent être intégrés dans le système soit par une instruction DEVICE= du fichier de configuration CONFIG.SYS soit par un appel au niveau de la ligne de commande. A partir de la version 5.0 DOS contient également une chimère de ce type : le driver EMM386.EXE.

En fait comme la plupart des autres drivers EXE, si on y regarde de plus près, ce programme n'est pas un véritable driver mais plutôt un programme résident. La dénomination de driver signifie ici que par le moyen du fichier CONFIG.SYS le programme peut être intégré au système à un moment où il n'a pas à craindre la concurrence des autres programmes résidents.

Ces "drivers résidents" se font passer pour des drivers de caractères pour que DOS ne leur affecte pas de désignation d'unité logique. Leur nom est défini de façon à ne pas entrer en collision avec celui d'un fichier (par ex EMMXXX0 pour les drivers d'EMS). Comme ils ne prennent pas en charge les devoirs normaux d'un driver ordinaire, ils n'en supportent pas les fonctions, exception faite de la routine d'initialisation (00h). Cette dernière routine doit s'exécuter de façon que DOS installe le programme résident comme un driver.

Comme on n'est jamais sûr que les autres fonctions du driver ne vont pas être déclenchées d'une manière ou d'une autre, les appels injustifiés ne devront pas être ignorés mais donner lieu à une erreur par activation de l'indicateur d'état dans le bloc de données.

C'est exactement ce que fait le programme présenté ci-dessous appelé EXESYS.ASM. Il a été conçu de manière à pouvoir être appelé indifféremment comme driver classique ou comme programme résident. Pour les drivers classiques, DOS n'exige pas l'extension SYS même si elle est presque toujours utilisée. DOS est donc capable de charger comme drivers des programmes EXE à condition qu'ils tiennent sur un seul segment et que leur en-tête se présente à l'offset 0h.

En assembleur cette condition est facile à réaliser il suffit de mettre en début de segment la directive

```
org 0h
```

et de rédiger ensuite l'en-tête. Au moment de l'installation, DOS y prélèvera l'offset des routiens de stratégie et d'interruption pour les invoquer ensuite comme d'habitude.

Mais pour que le même programme puisse aussi être appelé au niveau de la ligne de commande du système, il doit présenter un point d'entrée à la manière d'un programme EXE ordinaire. Pas de problème : il suffit de faire suivre la directive END, à la fin du

programme, par le nom d'une routine d'initialisation ad hoc, comme on le fait toujours quand on réalise un programme EXE.

Notre monstre à deux faces peut détecter assez simplement s'il est exploité comme driver ou comme programme ordinaire de DOS. En effet la routine d'initialisation n'est appelée que lorsqu'il est lancé comme programme EXE, alors que la routine de stratégie et d'interruption n'entre en action qu'en cas d'installation comme driver.

Mais quel est l'avantage d'appeler le programme résident lorsqu'il est déjà intégré comme driver ? On pourrait par exemple simuler le pilotage d'un driver en prévoyant que le programme demande certaines informations à l'utilisateur lorsqu'il est appelé à partir de la ligne de commande et qu'il transmette ces informations au driver. Avec les fonctions IOCTL (fonction 44h de DOS) et les fonctions du driver, la programmation ne pose aucun problème.

Le programme suivant constitue un modèle de développement qui peut être aussi bien lancé comme programme EXE que comme driver. Sa seule tâche consiste à afficher un message qui indique son mode d'appel. Mais il contient tout ce qu'il faut pour être étendu selon vos idées personnelles.

Listing : EXESYS.ASM

```
;*****************************************************************
;*                        E X E S Y S . A S M                    *
;*---------------------------------------------------------------*
;*   Fonction       : Propose un squelette de driver qui peut être *
;*                    lancé à partir de la ligne de commande     *
;*---------------------------------------------------------------*
;*   Auteur         : MICHAEL TISCHER                            *
;*   Développé le   : 1.11.1991                                  *
;*   Dernière MAJ   : 2.03.1992                                  *
;*---------------------------------------------------------------*
;*   Assemblage     : MASM EXESYS;                               *
;*                    LINK EXESYS;                               *
;*                                                               *
;*                    TASM EXESYS       ou                       *
;*                    TLINK EXESYS                               *
;*****************************************************************

code    segment

        assume cs:code,ds:code

        org 0                                   ;Programme sans PSP, débute
                                                ; à l'offset 0

;== Constantes ================================================

befehl  equ 2   ;Offset du champ d'instruction dans le bloc de données
status  equ 3           ;Offset du champ d'état dans le bloc de données
end_adr equ 14  ;Offset de l'adr. de fin du driver dans bloc de données

;== Données ===================================================

;-- En-tête du driver ---------------------------------------

        dw -1,-1                                ;Lien avec le driver suivant
        dw 1010000000000000b                            ;Attribut du driver
        dw offset strat                 ;Pointeur sur la routine de strategie
        dw offset intr                  ;Pointeur sir la routine d'interruption
        db "$$EXESYS"                                   ;Nom du driver

db_ptr  dw (?),(?)                      ;Adresse du bloc de données transmis

;===============================================================
;==                  Routines du programme EXE                ==
;===============================================================

exestart proc far

        push cs                                                 ;DS = CS
        pop  ds

        mov  ah,09h                                     ;Affiche un message
        mov  dx,offset exemes
        int  21h

        mov  ax,4C00h                   ;Terminaison normale d'un programme
        int  21h

exestart endp

exemes  db "EXESYS - (c) 1991, 92 by Michael TISCHER", 13,10,10
        db "Apel comme un programme EXE !", 13, 10, "$"


;===============================================================
;==                  Routines et fonctions du driver          ==
;===============================================================

strat   proc far                                        ;Routine de stratégie

        mov  cs:db_ptr,bx               ;Mémorise en DB_PTR l'adresse du
        mov  cs:db_ptr+2,es                             ; bloc de données

        ret                                             ;Retourne à l'appelant

strat   endp

;---------------------------------------------------------------

intr    proc far                                        ;Routine d'interruption

        push ax                         ;Sauvegarde les registres sur la pile
        push bx
        push cx
        push dx
        push di
        push si
        push bp
        push ds
        push es
        pushf                           ;y compris le registres des indicateurs

        push cs                         ;Fixe le registre du segment de données
```

```
;
;
;
;
;
;
;
;
intr_end label near
;
;
;
;
;
;
;
;
;
;
;
;
intr    endp
```

```
;
;:-----------------------------------------------------------------------
;:-- Lorsque l'appel est adressé au driver, la mémoire est libérée
;:-- à partir d'ici
;
init    proc near                               ;Routine d'initialisation
;
;
;
;
;
;
;
;:-- Données qui ne sont plus nécessaires après l'initialisation -----
;
ddmes   db "EXESYS - (c) 1991, 92 by Michael TISCHER", 13,10,10
        db "Appel au driver ", 13, 10, "$"
;
;:-----------------------------------------------------------------------
;
code    ends
        end exestart
;
```

25.10. Le CD-ROM : un driver tout à fait particulier

Quelques années seulement après son entrée dans l'univers de la hi-fi et l'explosion des ventes qui s'en est suivie, l'industrie du disque laser s'attaque maintenant au marché du PC. Force est de reconnaître qu'un lecteur de CD-ROM et un PC constituent un attelage intéressant, car un lecteur de disque laser met à la disposition de l'ordinateur une mémoire de masse qui est certes encore limitée à la lecture mais qui est amovible et permet de loger, sur son sillon unique, 660 Mo de données, qu'il s'agisse de textes, d'images, etc. Le CD-ROM est-il appelé à ouvrir une nouvelle ère de l'informatique ?

Ce n'est pas impossible car dans le domaine de la Hi-Fi on a pu constater que les prix des disques laser ont fondu avec la diffusion toujours plus large des lecteurs de disques laser. On trouve déjà aux USA, sous forme de CD-ROM, de nombreux ouvrages imprimés à des prix relativement faibles, par exemple des annuaires téléphoniques, des catalogues de livres, la Bible ou encore la traduction anglaise de la Pravda. On trouve également, numérisés sur CD-ROM, des cartes géographiques, des bibliothèques de photos, des recueils de programmes du domaine public, des banques de données médicales, avec des titres nouveaux tous les jours sur un marché en expansion.

L'avantage par rapport aux ouvrages imprimés classiques est évident : les informations qui ont été saisies et numérisées peuvent ensuite être retraitées par ordinateur sous n'importe quelle forme. Les possibilités ainsi offertes sont pratiquement illimitées si l'on songe combien il est facile de combiner des informations entre elles, de les évaluer, de les examiner sous différents angles et aussi, aspect non négligeable, de les mettre à la disposition d'autres publics.

Pour bénéficier dès aujourd'hui de ces possibilités, on peut acquérir, pour une somme allant de 6000 à 7000 F, un lecteur de CD-ROM qui peut être exploité par un PC comme périphérique externe ou interne. L'architecture matérielle d'un PC lui permet

très bien d'intégrer un tel lecteur, mais quelques problèmes se posent néammoins sur le plan logiciel, car DOS n'est pas prévu a priori pour supporter ce type de machines.

Cette section aura donc pour objet de montrer comment, à l'aide de drivers et de programmes auxiliaires appropriés, on peut parvenir à faire accepter le lecteur de CD-ROM comme un simple lecteur de disquette (en lecture seulement). Il ne nous est toutefois pas possible de vous garantir que les informations fournies ici auront pour vous un intérêt pratique, mais cette section nous permettra en tout cas une incursion captivante dans le domaine des drivers de périphériques. Elle s'adresse donc à tous les lecteurs intéressés par l'organisation interne de leur système d'exploitation.

Comme l'ont montré les chapitres précédents, les drivers de périphériques font le lien entre DOS et les périphériques externes tels que l'écran, l'imprimante, les lecteurs de disquette et le disque dur. DOS, nous l'avons vu, distingue deux sortes de drivers, les drivers de blocs et les drivers de caractères. Un lecteur de CD-ROM étant une mémoire de masse où les informations sont lues par blocs, on pourrait supposer que c'est tout naturellement par un driver de blocs qu'un lecteur de CD-ROM sera relié au reste du système. Mais voilà, ce n'est pas possible et les complications commencent car DOS exige des périphériques de blocs un certain nombre de conditions qu'un lecteur de CD-ROM ne peut remplir.

Notez déjà le problème posé par la capacité d'un CD-ROM. Normalement la capacité d'un périphérique de blocs est limitée, avant la version 4.0 de DOS en tout cas, à 32 Mo. Par ailleurs un CD-ROM ne dispose pas d'une table d'allocation des fichiers (File Allocation Table, FAT) car il n'est pas possible d'écrire sur les secteurs et aucun secteur ne peut donc être attribué à la FAT. A la place de la FAT, le CD-ROM comporte une sorte de table des fichiers dans laquelle sont inscrites les adresses où commencent les différents sous-répertoires et fichiers.

Pour réaliser un driver de CD-ROM, il est donc préférable d'avoir recours à la classe des drivers de caractères pour lesquels DOS ne pose aucune condition de structure des périphériques. Mais à bien y réfléchir, ces drivers conviennent assez mal à la communication avec un lecteur de CD-ROM car ils ne transfèrent pas les caractères par blocs mais un par un. Ils ne fonctionnent pas avec une désignation d'unité logique mais avec un nom de fichier tel que CON ou NUL. En dépit de ces inconvénients, le driver de CD-ROM devra se présenter à DOS comme un driver de caractères pour éviter des accès en lecture à une FAT inexistante. Le driver n'imposera cependant pas de nom de périphérique mais tirera un nom de son appel à l'intérieur du fichier de configuration CONFIG.SYS.

Comme le driver de CD-ROM est en général fourni par le fabricant du lecteur, il porte un nom du style SONY.SYS ou HITACHI.SYS et son appel de l'intérieur du fichier de configuration CONFIG.SYS se présentera par exemple de la façon suivante

```
DEVICE=HITACHI.SYS /D:CDR1
```

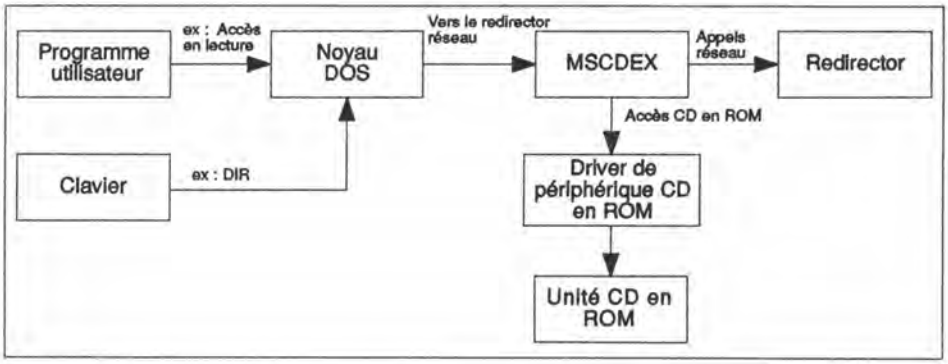
Le driver choisit ici "CDR1" comme nom du lecteur de CD-ROM.

Après appel de la routine d'initialisation par DOS, le driver révèle son vrai visage en s'initialisant comme un driver de blocs complété par quelques fonctions spéciales pour suppporter le CD-ROM. Pour DOS, il reste néammoins un driver de caractères de sorte qu'il est par exemple impossible d'examiner le répertoire du CD-ROM ou bien d'accéder à l'un des fichiers.

Pour franchir cet obstacle, un programme résident appelé MSCDEX (Microsoft CD-ROM Extension) est fourni avec le lecteur de CD-ROM en plus du driver. Ce programme est destiné à être appelé de l'intérieur du fichier AUTOEXEC. Il faut lui fournir dans la ligne de commande un paramètre représentant le nom du driver de CD :

```
MSCDEX /D:CDR1
```

MSCDEX ouvre tout d'abord ce driver par la fonction Open de DOS et lui attribue une désignation d'unité logique. MSCDEX fait en même temps croire à DOS que le périphérique en question est un lecteur lointain du réseau, tel que DOS les supporte à partir de la version 3.1.



Accès à un lecteur de CD-ROM par MSCDEX et le driver associé

MSCDEX fait ainsi un grand pas vers le but recherché car les lecteurs du réseau sont traités par DOS comme de grands fichiers pouvant comporter plus de 32 Mo. Ces périphériques ne sont pas appelés directement par DOS, mais par l'intermédiaire de ce qu'on appelle un redirecteur. Mais la partie résidante de MSCDEX s'installe dans ce redirecteur, interceptant ainsi tous les appels audit redirecteur. Si un appel destiné au lecteur de CD-ROM est rencontré, cet appel n'est pas retransmis au redirecteur de réseau classique mais chaque instruction est convertie en appels des différentes fonctions du driver de CD-ROM. La liaison entre DOS et le lecteur de CD-ROM est ainsi parfaite et un accès aux sous-répertoires et aux fichiers est possible à tout moment.

26. La gestion du système de fichiers sous MS-DOS

Du point de vue de l'utilisateur, le phénomène paraît très simple : il suffit de copier les fichiers d'un répertoire à l'autre, formater une disquette ou supprimer un fichier, le tout au moyen d'une seule commande. Mais derrière les coulisses, il en est tout autrement. DOS doit parcourir un long chemin avant de localiser un fichier sur le disque dur ou découvrir une place disque disponible afin de pouvoir y stocker un nouveau fichier.

Les unités de disquettes et disques durs ne veulent rien savoir à propos des fichiers et sous-répertoires. Elles ne connaissent que les secteurs, pistes et têtes. Par conséquent, DOS doit équiper chaque mémoire de masse d'un système permettant de basculer du niveau physique vers le niveau logique. Un tel système de fichiers se compose de toute une série de structures de données décrivant la taille d'une disquette ou du disque dur et leur contenu. Ce chapitre explicite le mode de configuration de ce système de fichiers.

26.1. La structure fondamentale du système de fichiers

La notion du volume constitue le noyau de la gestion de fichiers sous MS-DOS. Chaque disquette ou chaque unité de disque dur, désignée par une lettre individuelle, représente un volume. Contrairement aux disques durs, les disquettes ne peuvent pas être réparties entre plusieurs volumes. Jusqu'à la version 4.0 (non comprise) de DOS, les volumes étaient limités à une taille maximale de 32 Mo si bien qu'il fallait subdiviser les disques durs de plus de 32 Mo en plusieurs volumes pour ne pas perdre inutilement de la place mémoire.

Vous saurez comment la barrière des 32 Mo a été franchie depuis la version 4.0 au cours de ce chapitre. Mais notez d'ores et déjà que sous DOS chaque volume dispose d'une structure homogène qu'il soit défini sur une disquette ou un disque dur. Ici, la taille ne joue aucun rôle puisqu'elle dépend tout simplement de la taille des différentes structures de données nécessaires pour gérer le volume.

Noms de labels de volumes

Même si cela n'est pas toujours demandé, chaque volume peut être affecté d'un nom de volume lors de sa création. La commande DIR affiche les noms des volumes (labels) lorsqu'ils existent. Chaque volume dispose de son propre répertoire principal qui peut contenir à son tour des sous-répertoires et des fichiers. Ces sous-répertoires et fichiers peuvent être gérés par les fonctions de l'interruption 21H.

Secteurs

Du point de vue du DOS, un volume représente avant tout une succession de secteurs occupant chacun 512 octets. Les différents secteurs portent un numéro logique et leur numérotation commence à 0. Par exemple, un volume de 10 Mo se compose de 20480 secteurs numérotés de 0 à 20479. DOS n'agit pas du tout sur la disposition des différents secteurs. Cela incombe uniquement au driver de périphérique à travers lequel la communication entre DOS et le support physique est établie.

Peu importe qu'un driver de périphérique éparpille à tout hasard les différents secteurs logiques sur le support désigné ou les dispose en respectant un ordre strict. Le plus important est qu'il existe une affectation claire et nette entre les secteurs logiques et physiques.

Sur le plan interne, DOS opère avec des numéros de secteurs logiques alors que les diverses fonctions de fichier du DOS API font référence aux fichiers et sous-répertoires. Ainsi, DOS doit convertir les appels de fonctions en appels de secteurs. Il se sert à cet effet des répertoires du volume ainsi que d'une structure appelée FAT dont nous reparlerons plus loin.

Le tableau suivant affiche la structure fondamentale d'un volume

Structure d'un volume

Secteur de boot
Première table d'allocation des fichiers (FAT)
Une ou plusieurs copies de la FAT
Répertoire racine avec nom de volume
Zone de données pour les fichiers et les sous-répertoires

Comme le montre le tableau ci-dessus, chaque volume est divisé en différentes zones recevant d'une part les différentes structures de données du DOS et d'autre part les différents fichiers. Ces structures sont définies lors du formatage d'un volume à l'aide de l'instruction FORMAT.

Vous avez sans doute remarqué que la figure précédente ne donne aucune information sur la taille des différentes structures et zones. Il s'agit en effet d'une action quelque peu impossible car la taille coïncide toujours à celle du volume concerné. Une fois le volume formaté, on obtient la taille des différentes zones à partir des informations fournies par le secteur de boot.

26.2. Le secteur de boot

Lors du formatage d'un volume, le secteur de boot est toujours placé dans le premier secteur du volume pour que DOS puisse le localiser facilement. En dehors des informations concernant la taille et la structure du volume, il contient surtout le bootstrap loader qui permet d'amorcer le DOS.

Ce secteur contient toutes les informations nécessaires au lancement du DOS. Contrairement à ce que beaucoup supposent à tort, le DOS ne figure pas dans la mémoire du PC immédiatement après la mise en marche de ce dernier. Il doit, au contraire, être tout d'abord chargé puis lancé. A cet effet, c'est en premier lieu le BIOS qui se charge de l'initialisation du système après mise en marche de l'ordinateur et qui, une fois son travail terminé, charge en mémoire le secteur logique 0 de la disquette figurant dans le lecteur de disquette ou bien du disque dur, après quoi ce secteur est exécuté à partir de l'adresse 0.

Vérifiez que le premier secteur physique a été effectivement chargé et non le premier secteur logique. Il est important de faire attention à cette circonstance qui permet de déterminer que le premier secteur logique doit coïncider avec le premier secteur physique. Au moment où le secteur de boot est chargé, aucun driver de périphérique ne se trouve encore en mémoire pour autoriser l'accès au support. Même si un driver de périphérique dispose arbitrairement les divers secteurs logiques sur le support mémoire désigné, le secteur logique 0 doit néanmoins concorder avec le secteur physique.

Le tableau suivant décrit la structure du secteur de boot.

Structure du secteur de boot d'un volume

Adresse	Contenu	Type
+00 h	Instruction de saut à la routine boot	3 BYTE
+03h	Nom du fabricant et numéro de version	8 BYTE
+0Bh	Octets par secteur	1 WORD *
+0Dh	Secteurs par cluster	1 BYTE *
+0Eh	Nombre de secteurs réservés	1 WORD *
+10h	Nombre de File Allocation Tables (FATs)	1 BYTE *
+11h	Nombre d'entrées dans le répertoire racine	1 WORD *
+13h	Nombre de secteurs dans le volume	1 WORD *
+15h	Descripteur de support	1 BYTE *
+16h	Nombre de secteurs par FAT	1 WORD *
+18h	Secteurs par piste	1 WORD

Structure du secteur de boot d'un volume		
Adresse	**Contenu**	**Type**
+1Ah	Nombre de têtes de lecture/écriture	1 WORD
+1Ch	Distance du premier secteur du volume au premier secteur du support de mémoire de masse	1 WORD
+1Eh-1FFh	Routine boot	482 BYTE
Taille : 512 octets		
* = Bloc de paramètres BIOS (BPB)		

L'exécution du code de programme dans le secteur de boot commence toujours par le premier octet, c'est-à-dire à l'adresse d'offset 00h. C'est pourquoi le secteur de boot contient toujours à cet endroit une instruction JUMP (en langage machine) après exécution de laquelle le programme se poursuit. Cette instruction peut être une instruction de saut normale ou bien ce qu'on appelle un "short jump". Or le champ réservé à cette instruction de saut a une longueur de 3 octets alors qu'un "short jump" ne nécessite que 2 octets.

Cette instruction sera donc toujours suivie de l'instruction NOP qui ne fait rien et qui mesure 1 octet, de façon à ce que les 3 octets du champ soient comblés. Ce champ est suivi de toute une série de champs fournissant des informations bien précises sur la structure du support. Le premier de ces champs mesure 8 octets et il contient le nom du fabricant de l'ordinateur avec lequel ce support a été formaté ainsi que le numéro de la version du DOS sous laquelle cela a été effectué. Les champs suivants stockent le format physique du support, c'est-à-dire le nombre d'octets par secteur, le nombre de secteurs par piste, etc...

Ici est également inscrite la taille des différentes structures de données du DOS qui sont placées sur le support. Ces informations sont très importantes pour les différentes fonctions BIOS de l'interruption 13h dont les drivers de périphérique servent généralement à écrire dans les secteurs et les lire. C'est pourquoi cette zone est appelée bloc de paramètres du BIOS (BPB). Les informations du secteur de boot sont donc désignées par une partie du BPB. Mais DOS utilise ces informations dans des buts variés.

Le BPB est encore suivi de trois champs qui ne sont pas utilisés par le DOS mais qui peuvent fournir au driver de périphérique des informations supplémentaires sur le support.

Cette zone est suivie de ce qu'on appelle la routine boot strap, à laquelle l'instruction de saut saute au tout début du secteur de boot. Il incombe à cette routine de charger et de lancer le DOS à travers les différents éléments du système qui ont été présentés au chapitre 16.

Le secteur de boot peut encore être suivi de plusieurs secteurs réservés, qui peuvent, par exemple, contenir la suite du code du programme boot strap. Le nombre de ces

secteurs est inscrit dans le BPB à partir de l'adresse 0Eh. Il comprend toujours le secteur de boot, de sorte qu'un 1 dans ce champ indique qu'il n'y a pas d'autre secteur réservé, ce qui est le cas sur la plupart des PC.

L'option permettant d'étendre le secteur de boot à plusieurs secteurs aide considérablement à varier la disposition des différents éléments de la structure des fichiers. En effet, la FAT, sa duplication et le répertoire principal font naturellement un bond en arrière lorsque de multiples secteurs réservés font irruption dans le secteur de boot.

26.3. La FAT (Table d'allocation des fichiers)

Lorsque le DOS veut créer de nouveaux fichiers ou bien agrandir des fichiers déjà existants, encore faut-il bien sûr qu'il sache quels secteurs sont encore libres sur le support utilisé. Le DOS tire ces informations d'une structure appelée FAT (File Allocation Table), qui est placée immédiatement à la suite de la zone réservée sur le support. Chaque entrée de cette table correspond à un nombre de secteurs bien déterminé, qui se suivent immédiatement sur le support d'un point de vue logique pour former un groupe appelé cluster. Le nombre de secteurs constituant un cluster est inscrit dans la cellule de mémoire 0Dh du secteur de boot, qui fait partie de la table de paramètres du BIOS. Seules sont cependant autorisées les valeurs représentant une puissance de 2. Sur le disque dur du XT, cette cellule de mémoire contient par exemple la valeur 8 et un cluster comporte donc 8 secteurs contigus. Sur les disques durs connectés aux AT, 386 et 486, un cluster ne comporte généralement que 4 secteurs contigus.

La table suivante vous indique que le nombre de secteurs formant un cluster varie en fonction du support de mémoire.

Nombre de secteurs par cluster	
Machine	**secteur/Cluster**
Unité de disquette simple face	1
Unité de disquette haute capacité	1
Unité de disquette double face	2
Disque dur de l'AT	4
Disque dur du XT	8

Le tableau précédent indique qu'un programme de formatage n'est pas défini par rapport aux valeurs spécifiées, mais la taille d'un cluster peut être sélectionnée automatiquement. Cela est d'autant plus valable que depuis la version DOS 4.0 les volumes de plus de 32 Mo peuvent être configurés avec un nombre plus important de secteurs par cluster.

Avec plus de 32 Mo par volume et 4 secteurs par cluster, on avoisine en fin de compte la limite des 65536 clusters si bien qu'il faut plus de 16 bits pour lire le numéro de cluster. Le nombre toujours accru de secteurs regroupés dans un cluster permet d'augmenter quelque peu l'organisation de la structure des fichiers sans être obligé d'avoir recours à la structure actuelle.

Le tableau suivant montre la répartition des clusters effectuée par DOS 4.0 dans les volumes de plus de 32 Mo. Les volumes peuvent être augmentés de cette manière jusqu'à une capacité de 2 Go. Dans ce cas, le dernier cluster d'un fichier reçoit naturellement une place mémoire plus importante car le dernier cluster n'est complété que si la taille du fichier est un multiple de la taille du cluster.

Tailles de volume et cluster sous DOS 4.0					
Taille de volume jusqu'à	128 Mo	256 Ko	512 Mo	1028 Mo	2048 Mo
Taille de cluster	2 Ko	4 Ko	8 Ko	16 Ko	32 Ko
Secteurs par cluster	4	8	16	32	64

Fragmentation des fichiers

Le regroupement de plusieurs secteurs dans un cluster est dû à la logique employée par le DOS pour écrire des fichiers sur un support. Le DOS ne choisit en effet pas systématiquement des secteurs contigus pour sauvegarder les fichiers car il est plus économique de décomposer un fichier, autant que nécessaire, de façon à occuper tous les secteurs encore libres même si ces secteurs ne se suivent pas. Cette méthode de décomposition ne va cependant pas sans ralentir quelque peu l'accès aux fichiers puisque la tête de lecture/écriture doit être repositionnée pratiquement lors de chaque accès en lecture. Il convient donc d'éviter tout de même une trop grande fragmentation des fichiers et c'est pourquoi on regroupe en clusters plusieurs secteurs contigus sur un support. On est ainsi assuré que les secteurs d'un cluster pourront uniquement comporter une partie d'un fichier déterminé.

Si on travaillait au lieu de cela en utilisant tous les secteurs de façon totalement indépendante, un fichier de 24 secteurs pourrait fort bien, dans un cas limite, être réparti dans 24 secteurs complètement isolés les uns des autres, de sorte que la tête de lecture/écriture devrait être déplacée 24 fois pour lire la totalité du fichier. Le principe des clusters, tel qu'il est appliqué sur l'AT, c'est-à-dire avec 4 secteurs par cluster, permet donc de gagner beaucoup de temps puisqu'un fichier de cette taille n'occupe plus que 6 clusters différents et que la tête de lecture/écriture devra au maximum être déplacée 6 fois.

Il y a cependant un revers à la médaille. Cette méthode permet de gagner du temps en moyenne mais elle a aussi un inconvénient de taille. Chaque fichier occupant en effet au moins un cluster, une place mémoire parfois considérable risque d'être gaspillée.

Songez par exemple à votre fichier AUTOEXEC.BAT qui ne comporte certainement pas plus de 150 octets. Un secteur suffirait largement à stocker un fichier de cette taille et on gaspillerait même déjà près de 400 octets. Or sur le disque dur d'un AT, le moindre fichier occupe un cluster entier, c'est-à-dire 2048 octets, dont moins de 10 % seront utilisés. Plus de 1,5 Ko seront donc gaspillés pour tout fichier de moins de 512 octets mais aussi pour tout fichier dont le dernier cluster n'est rempli qu'à concurrence de moins de 512 octets !

Si la fragmentation des fichiers ralentit considérablement la vitesse d'accès lors de la lecture et l'écriture, ce n'est pas une raison valable pour rejeter l'organisation en clusters. Cela fait d'ailleurs l'affaire des fameux programmes de défragmentation qui, pendant longtemps, étaient fournis comme parties intégrantes de produits tels que PC Tools ou Norton Utilities. Leur principale fonction est d'annuler la fragmentation des fichiers et réorganiser ensuite le support de mémoire pour que tous les fichiers soient disposés en clusters contigus.

Structure de la FAT

La taille des différentes entrées de cette table est de 12 bits sous les versions 1 et 2 du DOS. Sous la version 3 du DOS, cette même taille dépend du nombre de clusters : s'il y a plus de 4096 clusters, 16 bits sont utilisés, sinon les 12 bits usuels par entrée. La largeur d'une entrée de la FAT tend à augmenter le nombre de clusters adressables et ainsi la taille du volume à gérer de cette manière.

Avec une FAT 12 bits, il est en fait possible de gérer 4096 clusters ce qui correspond à une capacité de 8 Mo dans un regroupement de quatre secteurs par cluster. On peut certes augmenter cette capacité en rajoutant davantage de secteurs dans un cluster, mais cela risque de provoquer une dépense inutile de place mémoire. Ainsi, dans les PC équipés de disques durs de 20, 40 Mo ou davantage, on rencontre tout simplement une FAT de 16 bits permettant d'adresser un maximum de 65536 clusters.

Il n'est pas possible de lire correctement la FAT si on ignore le nombre de bits par entrée de la FAT. Cette valeur doit donc être calculée avant tout accès à la FAT. On utilise à cet effet les informations fournies dans le bloc de paramètres du BIOS. Le nombre total de secteurs sur le volume est indiqué à partir de la cellule de mémoire 13h. Il suffit de diviser cette valeur par le nombre de secteurs par cluster (voir plus haut) pour obtenir le nombre de clusters dans le volume.

Les deux premières entrées de la FAT sont réservées et elles n'ont rien à voir avec l'occupation des différents clusters. Suivant la taille des différentes entrées, ces deux entrées réservées offrent 24 bits (3 octets) ou 32 bits (4 octets). Le premier octet contient ici ce qu'on appelle le descripteur de support (Media Descriptor) alors que les octets suivants contiennent systématiquement la valeur 255. Le descripteur de support figure également à l'adresse 15h du BPB. Il indique le type de machine permettant de traiter le support (une disquette par exemple). Les codes suivants sont possibles :

Codes du descripteur de support (Media Descriptor)	
Code	**Support**
F0h	Disquette 3"1/2, 2 faces, 80 pistes, 18 secteurs par piste Disquette 3"1/2, 2 faces, 80 pistes, 36 secteurs par piste
F8h	Disque dur
F9h	Disquette 5"1/4, 2 faces, 80 pistes, 15 secteurs par piste Disquette 3"1/2, 2 faces, 80 pistes, 9 secteurs par piste
FAh	Disquette 5"1/4, 1 face, 80 pistes, 8 secteurs par piste Disquette 3"1/2, 1 face, 80 pistes, 8 secteurs par piste
FBh	Disquette 5"1/4, 2 faces, 80 pistes, 8 secteurs par piste Disquette 3"1/2, 2 faces, 80 pistes, 8 secteurs par piste
FCh	Disquette 5"1/4, 1 face, 40 pistes, 9 secteurs par piste
FDh	Disquette 5"1/4, 2 faces, 40 pistes, 9 secteurs par piste
FEh	Disquette 5"1/4, 1 face, 40 pistes, 8 secteurs par piste
FFh	Disquette 5"1/4, 2 faces, 40 pistes, 8 secteurs par piste

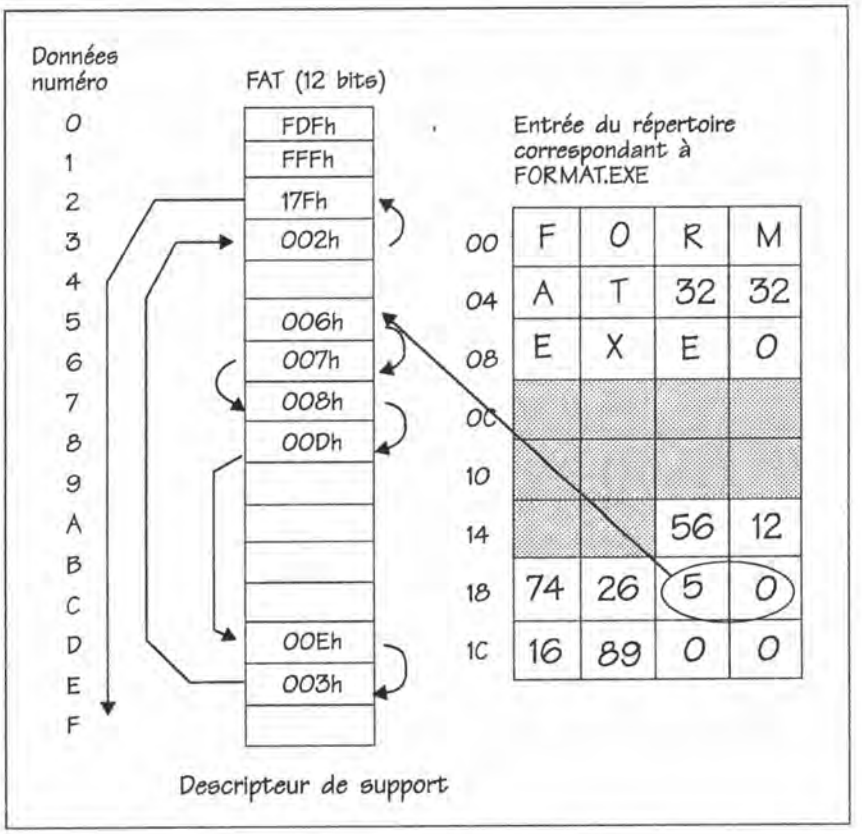
Codification des entrées de la FAT

Vous vous êtes certainement déjà demandé pourquoi, au fond, les différentes entrées de la FAT devraient comporter 12 ou 16 bits s'il s'agit simplement de spécifier si un cluster est occupé ou non. Un bit suffirait en effet pour indiquer que le cluster est occupé (avec la valeur 1) ou qu'il est libre (avec la valeur 0).

S'il n'en est pas ainsi, vous vous doutez bien que cela est dû au fait que les entrées de la FAT n'ont pas simplement pour tâche de désigner les clusters libres mais aussi d'enregistrer dans quels clusters sont stockées les données de chaque fichier. L'entrée du répertoire consacrée à chaque fichier indique au DOS dans quel cluster sont stockées les premières données d'un fichier. Le numéro de ce cluster coïncide avec le numéro de l'entrée de la FAT correspondante. Cette entrée contient à son tour le numéro du cluster dans lequel sont stockées les données suivantes du fichier et ainsi de suite jusqu'à la fin du fichier. Cependant, les numéros de cluster ne se rapportent pas au début du volume mais au début de la zone de données comme nous le verrons au cours de ce chapitre. Prenons un exemple pour illustrer ce point : Supposons que dans une entrée de répertoire il existe un fichier dont le début se trouve dans le quatrième cluster du volume. Pour obtenir maintenant le numéro du cluster suivant, il faut lire la quatrième entrée de la FAT (attention ! la longueur d'une entrée de la FAT peut être de 12 ou 16 bits). Le numéro du cluster suivant du fichier se trouve dans cette dernière.

Comme le montre la figure suivante, il se forme ainsi une chaîne permettant de localiser, dans l'ordre qui convient, les différents clusters constitutifs d'un fichier. Et à partir des numéros de clusters, on obtient les numéros logiques des différents secteurs que DOS

doit communiquer au driver de périphérique lorsqu'il souhaite lire ces secteurs ou y écrire. Les drivers de périphérique de DOS ne travaillent pas en fait au niveau des clusters mais au niveau des secteurs. La conversion entre cluster et secteur logique devient ainsi très souple puisqu'il suffit de multiplier le numéro de cluster par le nombre de secteurs par cluster.



Affectation des clusters d'un fichier définie par l'entrée de la FAT

L'entrée de la FAT coïncidant avec le dernier cluster doit contenir un code particulier indiquant à DOS la fin du fichier. Cela est signalé par un numéro de cluster qui est supérieur à FF8h dans une FAT 12 bits et à FFF8h dans une FAT 16 bits. Ces numéros de clusters ne sont pas les seuls et uniques réservés à des fins spéciales, comme le montre d'ailleurs le tableau suivant. Il existe ainsi les numéros FF0h à FF6h ou FFF0h à FFF6h pour les clusters réservés. Ils désignent par exemple le répertoire principal d'un volume dont la taille et l'état sont fixes et dont les clusters ne doivent pas former une chaîne à travers la FAT.

De même, le code cluster FF7h ou FFF7h a également une signification particulière. Il marque les clusters comportant des secteurs défectueux pour qu'ils ne soient pas utilisés pour stocker des fichiers au risque de provoquer une perte de données.

Les clusters inoccupés sont signalés par le code 0h. Il sert également pour le premier cluster du volume mais, tout comme le premier cluster, il ne représente aucun secteur parce que le type du support du volume se trouve stocké à l'intérieur. Cela explique également pourquoi le numéro de cluster doit être réduit à deux lors de la conversion des numéros de clusters en numéros de secteurs. Mais nous reviendrons sur ce point ultérieurement.

Code cluster FAT 12 bits	Signification
000h	Cluster est libre
FF0h - FF6h	Cluster réservé
FF7h	Cluster défectueux, inutilisé
FF8h - FFFh	Dernier cluster d'un fichier
xxxh	Prochain cluster d'un fichier
Code cluster FAT 16 bits	**Signification**
0000h	Cluster est libre
FFF0h - FFF6h	Cluster réservé
FFF7h	Cluster défectueux, inutilisé
FFF8h - FFFFh	Dernier cluster d'un fichier
xxxxh	Prochain cluster d'un fichier

Copies multiples de la FAT

Comme la liaison des divers clusters dans lesquels est rangé un fichier s'effectue exclusivement à travers la FAT, celle-ci occupe une place prépondérante à l'intérieur de la structure des fichiers de DOS. Cette coordination se réalise toujours négativement chaque fois qu'une erreur électronique ou un virus fait irruption dans la FAT. Dans ce cas, tous les fichiers et répertoires deviennent inaccessibles à l'utilisateur bien qu'ils soient encore présents parce que le DOS ne parvient plus à regrouper les différents clusters en une entité.

Par conséquent, le DOS invite un programme de formatage à prévoir non seulement une mais plusieurs copies identiques de la FAT. Leur nombre doit être conservé à l'intérieur du secteur de boot à l'adresse d'offset 10h. Si le DOS rencontre un tel support, il examine simultanément toutes les copies de la FAT et coche à chaque fois les clusters nouvellement occupés ou libérés lors de la création ou suppression de fichiers.

Le DOS permet aux fabricants d'ordinateurs de stocker plusieurs copies conformes de la FAT sur le support. L'avantage est que cela permet de remplacer la FAT principale par une autre au cas où elle aurait été endommagée. On évite ainsi une perte totale de données, ce qui est très appréciable.

L'instruction CHKDSK permet par exemple de tester si les différentes FAT sont identiques.

26.4. Le répertoire principal

Le répertoire racine d'un volume est placé immédiatement à la suite de la dernière copie de la FAT. Il comporte comme tous les autres sous-répertoires des entrées de 32 octets fournissant des informations sur les différents fichiers et sous-répertoires. Outre le nom du volume, il indique l'attribut, la date de création et de dernière modification et l'heure, surtout l'état du fichier, c'est-à-dire le numéro de son premier cluster.

Le nombre maximum d'entrées dans le répertoire racine est limité par sa taille stockée dans le secteur de boot à l'adresse d'offset 11h. En raison de la structure statique du répertoire racine, sa taille ne peut pas croître lorsque le nombre de fichiers et sous-répertoires augmente dans le répertoire principal d'un volume. Il arrive ainsi que le répertoire racine soit saturé et qu'un message d'erreur apparaisse sur l'écran lors de la création de sous-répertoires ou de la duplication de fichiers.

Lors du formatage d'un volume, il faut sélectionner la capacité du répertoire racine en fonction de la taille du volume pour éviter un tel désagrément, à moins que vous ne souhaitiez regrouper tous vos programmes dans le répertoire racine d'une disquette ou d'un disque dur.

Le nombre des entrées du répertoire racine provient de la multiplication par 16 des secteurs réservés à cet effet puisqu'avec une entrée de 32 octets, on peut stocker 16 entrées de répertoire différentes dans un secteur.

La structure des entrées de répertoire

Le tableau suivant montre la structure d'une entrée de répertoire pouvant décrire des fichiers et sous-répertoires :

Structure d'une entrée du répertoire		
Adresse	**Contenu**	**Type**
+00h	Nom du fichier (comblé avec des espaces)	8 BYTE
+08h	Extension de fichier (comblée avec espaces)	3 BYTE

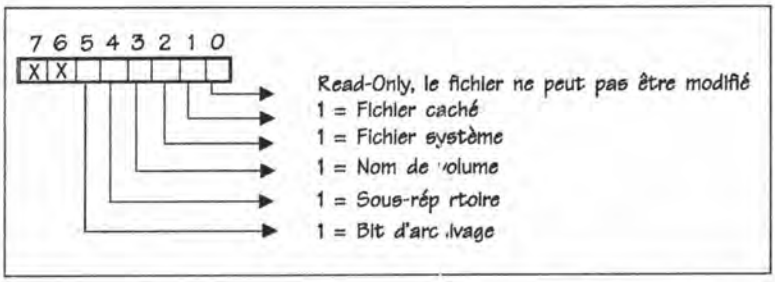
Structure d'une entrée du répertoire		
Adresse	**Contenu**	**Type**
+0Bh	Attribut de fichier	1 BYTE
+0Ch	Réservé	10 BYTE
+16h	Heure de la dernière modification	1 WORD
+18h	Date de la dernière modification	1 WORD
+1Ah	Premier cluster du fichier	1 WORD
+1Ch	Taille du fichier	1 DOWRD
Taille : 32 octets		

Les 8 premiers octets contiennent normalement le nom du fichier. Si ce nom comporte moins de 8 caractères, il est comblé avec des espaces (code ASCII 32). Si l'entrée du répertoire ne contient cependant pas les informations concernant un fichier, parce qu'elle est employée à un autre usage, le premier octet du nom de fichier, c'est-à-dire le premier octet de l'entrée du répertoire, reçoit un code particulier :

Code	Signification
00h	Dernière entrée du répertoire
05h	Le premier caractère du nom de fichier est le caractère ASCII E5h
2Eh	Fichier concernant le répertoire actuel. Si un autre 2Eh suit, il se rapporte au répertoire père.
E5h	Le fichier a été supprimé

Le second champ reçoit l'extension de trois lettres du fichier. Ici également, l'extension est comblée le cas échéant avec des espaces pour obtenir cette longueur de 3 octets. Le point entre le nom de fichier et l'extension, qui doit être employé lorsque vous spécifiez les deux éléments d'un nom de fichier, n'est pas stocké.

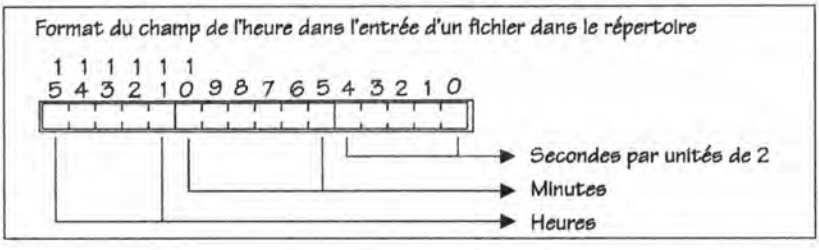
Vient ensuite un champ d'attribut d'une taille d'un octet. La figure suivante montre comment les différents bits de ce champ spécifient chacun un attribut bien précis. Les différents attributs peuvent être combinés. Un fichier pourra donc présenter à la fois (comme c'est le cas par exemple du fichier IBMBIO.COM) les attributs LECTURE_SEULE, SYSTEME et CACHE.

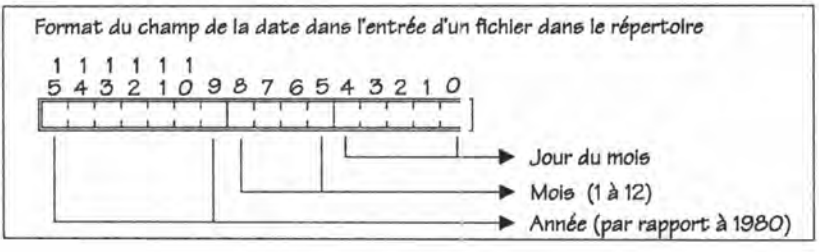


La signification des bits 0 à 4 est assez évidente. Celle du bit 5 appelle, par contre, quelques explications supplémentaires. Ce bit est dit bit d'archivage car il sert à la mise en place de copies de sécurité. Chaque fois qu'un fichier est créé ou modifié, ce bit est fixé sur 1. Si un programme de sauvegarde (par exemple le programme BACKUP du DOS est utilisé après cela), ce programme sauvegardera le fichier mais fixera ensuite le bit d'archivage sur 0. Si ce programme ou tout autre programme de sauvegarde est à nouveau appelé, il pourra alors déterminer d'après l'état de ce bit si le fichier a ou non été modifié depuis la dernière sauvegarde. Tant que ce bit vaut 0, cela signifie qu'il n'est pas nécessaire de sauvegarder le fichier à nouveau. Par contre, si le bit d'archivage contient la valeur 1, c'est que le fichier a été modifié entre-temps et qu'il convient donc de le sauvegarder à nouveau.

Le champ d'attribut est suivi d'un champ réservé que le DOS utilise pour ses opérations internes. Jusqu'à présent, Microsoft n'a pas officialisé sa signification.

Les champs d'heure et de date indiquent à quel moment le fichier a été créé ou bien modifié pour la dernière fois. Ces deux informations sont sauvegardées sous forme d'un mot (2 octets) mais leurs formats sont différents.





Aux champs heure et date vient se brancher un champ particulièrement décisif qui indique le numéro du cluster contenant les premiers secteurs du fichier. Il conduit également à l'entrée de la FAT révélant ainsi le cluster suivant du fichier. Notez toutefois que ce champ ne joue aucun rôle dans un nom de volume parce qu'il n'est pas relié à un fichier mais indique tout simplement le nom du volume. Il en est tout autrement dans le cas des sous-répertoires, comme nous allons le voir ci-après.

La taille du fichier en octets est stockée sous forme de 2 mots, le mot de plus faible poids étant stocké en premier. La petite formule suivante permet de calculer la taille du fichier à partir de ces deux mots :

```
Taille du fichier = Mot1 + Mot2 * 65536
```

Sous-répertoires

Si le bit 4 du champ d'attribut est mis, c'est que l'entrée considérée représente un sous-répertoire. Si, en outre, le bit 1 est également mis dans ce champ, cela signifie que ce sous-répertoire peut être appelé mais qu'il ne sera pas affiché par l'instruction DIR. Dans une entrée de ce type, les champs de nom de fichier et d'extension reçoivent le nom du sous-répertoire, les champs de date et heure reçoivent les date et heure de sa création et le champ de longueur du fichier contient toujours 0.

Le champ qui indique normalement le premier cluster du fichier indique dans ce cas quel cluster contient les entrées de ce sous-répertoire. Ces entrées présentent la même structure sur 32 octets que les entrées du répertoire racine. Comme pour un fichier normal, l'entrée de la FAT qui correspond au cluster du sous-répertoire désigne le prochain cluster de ce sous-répertoire si ce répertoire compte plus d'un cluster.

Ce n'est pas le cas pour le répertoire racine bien que celui-ci s'étende généralement sur plusieurs secteurs ou clusters car ces clusters se suivent toujours sur le plan logique. Les différents clusters de la racine ne pourraient de toute façon pas être reliés à travers la FAT puisque celle-ci se réfère uniquement à la zone de données du volume, c'est-à-dire à la zone qui reçoit les fichiers et les sous-répertoires mais pas le répertoire racine.

La méthode que nous venons de décrire révèle deux aspects importants de la gestion des fichiers par le DOS : les différents fichiers d'une unité de mémoire de masse sont distingués d'après les répertoires dont ils relèvent mais les fichiers d'un répertoire déterminé ne sont pas nécessairement logés dans une zone particulière de cette unité de mémoire ; ils peuvent en effet être répartis dans des emplacements très divers de la mémoire de masse.

Lorsqu'un sous-répertoire est créé, deux entrées y sont immédiatement mises en place d'office, sous les noms '.' et '..'. Ces entrées ne peuvent être éliminées et elles ne peuvent être supprimées que lors de l'effacement du sous-répertoire tout entier. Le premier de ces deux fichiers désigne le sous-répertoire actuel et son champ de cluster contient le

numéro du premier cluster du sous-répertoire actuel. La seconde entrée désigne ce qu'on appelle le répertoire-père, qui correspond exactement, dans l'arbre des répertoires, au noeud précédant le répertoire actuel. Si le répertoire-père est la racine, le champ de cluster contient la valeur 0. C'est cette entrée qui permet au DOS de retrouver le chemin qui mène à la racine puisqu'en recherchant chaque fois le répertoire-père du sous-répertoire on se rapproche toujours plus du répertoire racine.

Bien que le DOS traite les sous-répertoires et le nom de volume comme des fichiers, on ne peut y accéder à travers les fonctions de manipulation d'un fichier. Nous ne pouvons d'ailleurs que vous déconseiller de faire la moindre tentative dans ce sens.

26.5. La zone de données

Mais revenons à notre examen de la structure d'une mémoire de masse sous le DOS. Après le répertoire racine, que nous venons de décrire, vient la zone des fichiers, qui occupe le reste de la place mémoire disponible sur une mémoire de masse. Cette zone reçoit les différents fichiers ainsi que les différents sous-répertoires. A chaque cluster de cette zone correspond naturellement une entrée de la FAT qui peut être modifiée lorsque sont manipulés les fichiers placés dans cette zone.

Si un fichier est agrandi, par exemple, le DOS réserve un cluster encore libre pour y stocker les données suivantes de ce fichier. L'entrée de la FAT qui correspondait jusqu'ici au dernier cluster du fichier voit la marque de fin qu'elle contenait remplacée par l'indication du nouveau cluster qui reçoit à son tour la marque de fin. Pour rechercher un cluster pouvant recevoir les nouvelles données du fichier, les versions 1 et 2 du DOS examinaient la FAT du début à la fin, pour réserver le premier cluster non marqué occupé.

La version 3 applique une méthode un peu plus complexe de façon à choisir dans la mesure du possible un cluster qui se situe à proximité de l'ancien cluster, de façon à minimiser les temps d'accès au fichier. Lorsqu'un fichier est raccourci ou supprimé, les clusters ainsi libérés sont marqués comme libres dans la FAT. Ils peuvent donc être réutilisés dès la prochaine création ou dès le prochain agrandissement d'un fichier.

Nous n'en avons pas fini avec les explications d'ordre théorique. En conclusion de ce chapitre, nous allons en effet essayer de comprendre comment nous pouvons accéder à l'aide de la FAT aux différents secteurs d'un fichier.

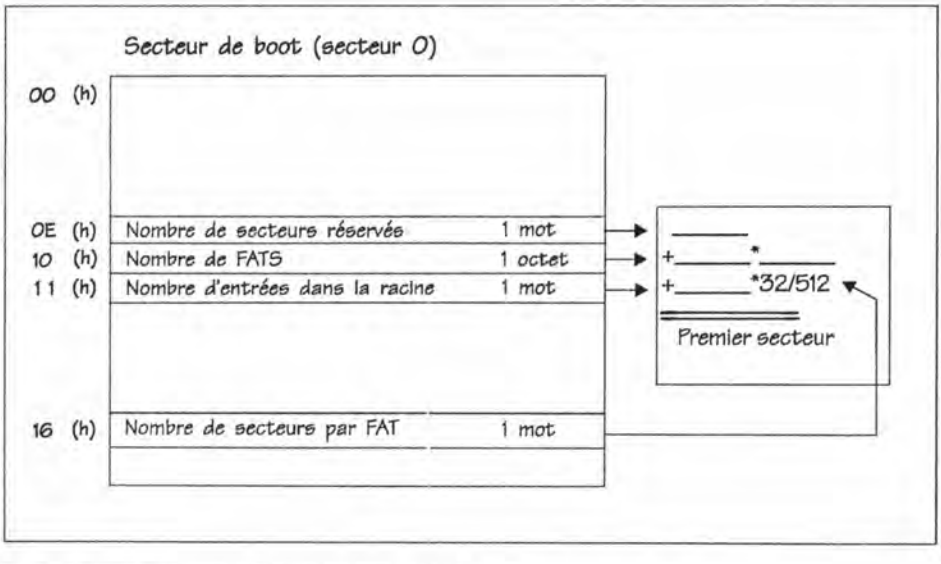
Pour commencer, il nous faudra déterminer à partir de l'entrée de répertoire du fichier voulu son premier cluster et du même coup l'entrée de la FAT désignant le cluster suivant. Il s'agira ensuite de calculer le secteur correspondant à partir de cette valeur de cluster. Il faut tout d'abord retrancher 2 du numéro de cluster car les deux premières entrées de la FAT sont occupées (descripteur de support).

C'est donc la troisième entrée de la FAT qui correspond au cluster numéro zéro sur le support. Le numéro de cluster doit ensuite être multiplié par le nombre de secteurs par cluster.

Nous n'avons cependant toujours pas le numéro de secteur qui convient car les numéros de cluster dans l'entrée du répertoire ainsi que dans la FAT ne se réfèrent pas au premier secteur du volume mais au premier secteur d'une zone de données du volume. Or comme nous l'avons vu au début de ce chapitre, cette zone ne commence qu'après le répertoire racine du volume.

Il convient donc d'additionner encore au numéro de secteur calculé le numéro du premier secteur de la zone de données. Ce numéro peut être calculé à partir des informations fournies dans le bloc de paramètres du BIOS, à l'intérieur du secteur de boot. Comme vous le savez, la zone de données d'un volume est précédée du secteur de boot, de la FAT et des copies de celle-ci et, en dernier lieu, du répertoire racine. Il nous faut donc additionner la longueur de chacune de ces zones de données. Le nombre de secteurs réservés (y compris le secteur de boot) nous est fourni par la cellule de mémoire 0Eh du secteur de boot alors que la cellule de mémoire 16h indique le nombre de secteurs par FAT.

Cette valeur doit être multipliée par le nombre de FAT, dans la cellule de mémoire 10h, à quoi il faut additionner le nombre de secteurs réservés. Il ne reste plus qu'à lire, dans le mot à l'adresse 11h du secteur de boot, le nombre d'entrées dans le répertoire racine. En multipliant cette valeur par 32 (octets par entrée), puis en divisant le résultat par 512 (octets par secteur), nous obtenons le nombre de secteurs de la racine. Nous additionnons le résultat au total précédent et nous obtenons ainsi le numéro du premier secteur de la zone des données.



Calcul du premier secteur de la zone des données

En additionnant ce total au numéro du premier secteur de notre fichier que nous venons de calculer, nous obtenons le numéro de secteur absolu du premier secteur de notre fichier. Les secteurs placés d'un point de vue logique à la suite de ce premier secteur forment le premier cluster de notre fichier, en fonction du nombre de secteurs que contient un cluster. Nous pouvons maintenant lire ou écrire des données dans ce cluster ou secteur en employant ce numéro de secteur dans les fonctions du DOS qui se cachent derrière les interruptions 25h et 26h.

Si nous voulons déterminer ensuite quel est le cluster suivant et donc quelle est la prochaine entrée de la FAT, c'est dans la FAT que nous devrons lire ces informations. La façon de procéder pour ce faire dépend ici essentiellement de la largeur des différentes entrées de la FAT.

Il sera en effet très facile de lire le prochain cluster si la largeur des entrées de la FAT est de 16 bits. Dans ce cas, il suffit de multiplier le numéro de cluster par 2. Le résultat de cette multiplication représente l'adresse d'offset, par rapport au début de la FAT dans la mémoire, à laquelle est indiqué le prochain cluster.

Si la largeur de la FAT est par contre de 12 bits, le numéro de cluster devra d'abord être multiplié par 1,5. La partie entière de ce produit sera alors utilisée comme offset par rapport à la FAT pour lire le mot figurant à cette adresse. Si le produit était entier, il suffira ensuite de faire subir au mot lu un ET logique avec la valeur 0FFFh pour obtenir le numéro du prochain cluster. Si le produit n'était pas par contre un nombre entier, il n'y a pas lieu d'opérer de combinaison ET logique et le mot doit simplement être décalé de 4 bits vers la droite, c'est-à-dire divisé par 16.

Nous connaissons ainsi le premier cluster du fichier. Si sa valeur est comprise entre FF8h et FFFh pour une FAT de 12 bits ou entre FFF8h et FFFFh, c'est que la fin du fichier est atteinte. Sinon l'opération doit être renouvelée depuis le début.

26.6. Formats de disquettes

En ce qui concerne les disquettes, DOS prévoit des formats bien précis spécifiant exactement les diverses caractéristiques d'un volume. DOS ne peut pas traiter d'autres formats de disquettes sans demander l'aide de drivers spécifiques.

Les tableaux suivants énumèrent les formats de disquettes 5 pouces 1/4 et 3 pouces 1/2 :

Les différents formats DOS de disquettes 5"1/4					
Existe depuis la version DOS	1.0	1.10	2.0	2.0	3.0
Descripteur de support	FEh	FFH	FCh	FDh	F9h
Nombre de têtes de lect./écr.	1	2	1	2	2
Nombre de pistes par tête	40	40	40	40	80
Nombre de secteurs par piste	8	8	9	9	15
Nombre d'octets par secteur	512	512	512	512	512
Nombre de secteurs par clust.	1	2	1	2	1
Nombre de secteurs réservés	1	1	1	1	1
Nombre de secteurs par FAT	1	1	2	2	7
Nombre de FATs	2	2	2	2	2
Secteurs dans répert. racine	4	7	4	7	14
Entrée dans répert. racine	64	112	64	112	224
Nombre total de secteurs	320	640	360	720	2400
Secteurs libres pour fichiers	313	630	351	708	2371
Nombre de clusters	313	315	351	354	2371
Capacité totale	160 Ko	320 Ko	180 Ko	360 Ko	1,2 Mo
Capacité totale pour fichiers	156,5 Ko	315 Ko	175,5 Ko	354 Ko	1,185Mo

Les différents formats DOS de disquettes 3,5"		
Existe depuis la version DOS	3.0	3.30
Descripteur de support	F9h	F0H
Nombre de têtes de lect./écr.	2	2
Nombre de pistes par tête	80	80
Nombre de secteurs par piste	9	18
Nombre d'octets par secteur	512	512
Nombre de secteurs par clust.	2	1
Nombre de secteurs réservés	1	1
Nombre de secteurs par FAT	3	9
Nombre de FATs	2	2

Les différents formats DOS de disquettes 3,5"		
Existe depuis la version DOS	3.0	3.30
Secteurs dans répert. racine	7	14
Entrée dans répert. racine	112	224
Nombre total de secteurs	1440	2880
Secteurs libres pour fichiers	1426	2847
Nombre de clusters	720	2880
Capacité totale	720 Ko	1.5 Mo
Capacité totale pour fichiers	713 Ko	1.4 Mo

27. Le multiplexeur

Parmi les différentes commandes de DOS il en existe certaines qui fonctionnent comme des programmes résidents. Elles s'installent en mémoire pour pourvoir à leurs tâches en arrière-plan. Parmi ces commandes figurent PRINT, ASSIGN, SHARE, APPEND, DOSKEY et quelques autres. Une fois ancrées en mémoire ces commandes s'incrustent dans l'interruption 2Fh appelée multiplexeur. Ce chapitre est destiné à démontrer le fonctionnement du multiplexeur et à étudier les commandes qui l'exploitent.

27.1. Fonctionnement d'un multiplexeur

Le nom porté par cette interruption est dû à ce qu'elle ne limite pas ses services à un seul programme DOS : elle est ouverte à tous les programmes résidents qui ont besoin d'une interface de communication vers l'extérieur. Très souvent le premier appel à un programme résident provoque son installation tandis que les appels ultérieurs à partir de la ligne de commande de DOS servent à modifier des paramètres dans la copie installée ou à retirer le programme de la mémoire. Le programme appelé entre en contact avec la copie installée par l'interruption multiplexeur.

Les programmes d'application peuvent eux aussi exploiter ces appels pour intervenir dans le fonctionnement de différentes commandes résidentes de DOS ou pour vérifier si le programme est actif. Un programme de réseau qui est obligé de s'appuyer sur les fonctions de la commande SHARE de DOS peut ainsi tester si SHARE a déjà été chargé et interrompre son exécution par un message d'erreur si tel n'est pas le cas.

Ce sont les programmes résidents eux-mêmes qui font en sorte que l'interruption 2Fh puisse être exploitée simultanément par plusieurs programmes. Le processus est le suivant.

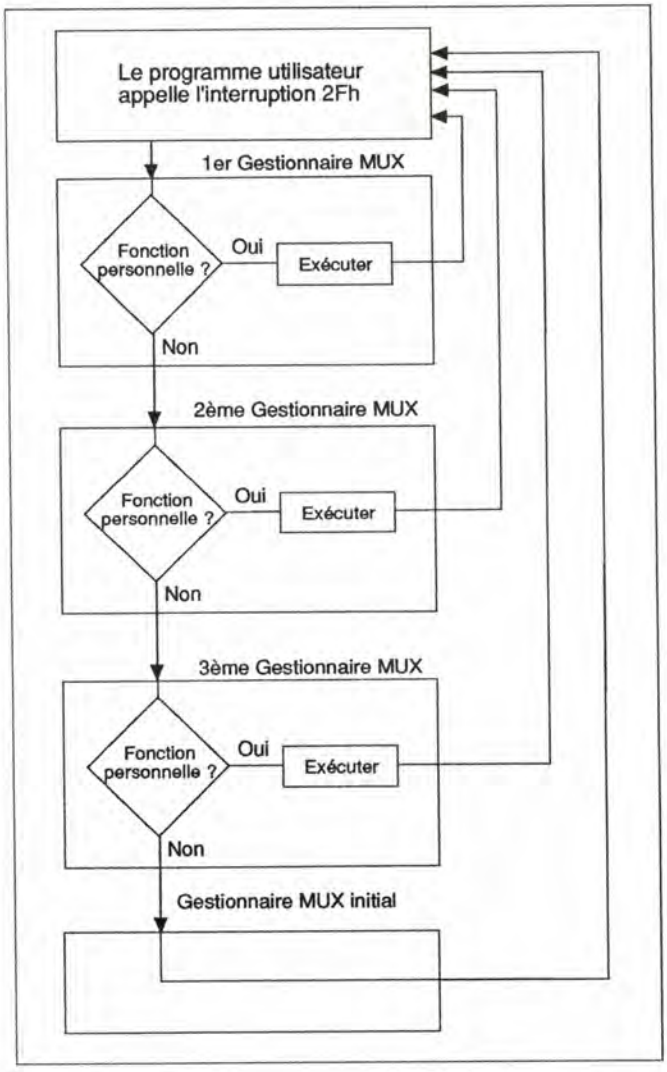
Un programme désireux d'utiliser le multiplexeur (MUX) doit commencer par s'attribuer un numéro de 8 bits. Ce numéro est appelé code MUX. Les codes de 00h à BFh sont réservés aux programmes de DOS mais la plage de 0Ch à FFh est libre et peut être utilisée par les programmes d'application.

Lors de son installation un programme résident devra prévoir un gestionnaire multiplexeur en plus de tous les autres gestionnaires d'interruption qui lui sont indispensables. Il est donc obligé de détourner l'interruption 2Fh vers une routine qui remplace l'ancien gestionnaire d'interruption. Plus tard si un programme déclenche le multiplexeur, le gestionnaire multiplexeur du programme résident sera exécuté.

Ce gestionnaire d'interruption doit d'abord tester si c'est son programme qui est concerné ou un autre qui fait partie de la chaîne des gestionnaires multiplexeurs. A cet effet il examine le registre AH, car à l'appel de l'interruption il doit s'y trouver le code MUX du programme concerné.

Si AH contient son propre code MUX, la suite est simple à imaginer. Il va lire les autres registres du processeur qui ont une signification pour lui, exécute sa fonction et retourne à l'appelant par une instruction IRET.

Si en examinant le code MUX, il constate qu'un autre programme est visé, il n'a pas le droit de mettre fin à l'appel en rendant le contrôle à l'appelant. Il doit appeler l'ancien gestionnaire multiplexeur qu'il a repoussé lors de son installation. Par conséquent le programme résident prendra soin de sauvegarder l'adresse du gestionnaire précédent mais cette précaution ne pose aucun problème.



Chaîne des gestionnaires MUX

Chaque gestionnaire multiplexeur se comportant de cette façon, il se forme une chaîne. Tout appel au multiplexeur est transmis de gestionnaire en gestionnaire jusqu'à ce que l'un d'eux découvre son code MUX et exécute la fonction désirée sans repasser le contrôle plus loin. A la fin de la chaîne se trouve toujours le gestionnaire d'origine installé par le BIOS au lancement du système et qui contient simplement la commande IRET.

Concurrence entre les gestionnaires multiplexeurs

Un problème se pose lorsque deux gestionnaires multiplexeurs utilise le même code d'identification. Dans ce cas c'est la situation du gestionnaire dans la chaîne des interruptions qui décide de la fonction qui sera déclenchée. C'est toujours le dernier gestionnaire installé qui aura la main car il plus proche du début de la chaîne. Le gestionnaire plus ancien ne peut faire valoir aucun droit.

Pour les programmes DOS il ne faut pas s'attendre à des difficultés de ce type. En effet Microsoft leur a attribué des codes MUX distincts. Mais si vous développez vous-même des programmes résidents et si vous faites appel au multiplexeur, vous devriez normalement vivre dans la crainte qu'un autre programme n'exploite déjà le même code. Mais heureusement il existe une possibilité de recherche au démarrage d'un programme.

Par convention chaque programme est tenu de préparer une fonction pour le test d'installation. Elle est exécutée lorsqu'à l'appel du multiplexeur on met en AH le code du programme et en AL la valeur 00h. Si l'un des gestionnaires déjà installés reconnaît son code, il doit retourner en AL une valeur différente de 0 (en général FFh).

Si aucun des gestionnaires ne se sent responsable pour le code indiqué, il passe finalement aux mains du dernier de la chaîne, c'est-à-dire à l'original qui est constitué de la seule instruction IRET et qui ne modifie donc pas AL. Si à l'issue de l'appel, le registre AL n'est pas différent de 0, on peut être sûr que le code testé n'a pas encore été utilisé par un gestionnaire.

Le chapitre 32 montre l'utilité du multiplexeur dans le développement des programmes résidents

27.2. Exploitation du multiplexeur par les programmes de DOS

Cette section est consacrée à l'étude du multiplexeur au service des différents programmes de DOS. Un programme d'application ne pourra à mon avis exploiter que les seules fonctions de la commande PRINT qui permet d'effectuer une impression en arrière-plan mais vous jugerez vous-même.

Liste de commandes de DOS qui s'intègrent dans le multiplexeur	
Code MUX	**Commande de DOS**
01h	PRINT
06h	ASSIGN
10h	SHARE
1Ah	ANSI.SYS
43h	HIMEM
48h	DOSKEY
ADh	KEYB
B0h	GRAFTABL
B7h	APPEND

Nous allons examiner une à une les fonctions des différentes commandes de DOS en les décrivant brièvement. Vous trouverez en annexe XX la liste des paramètres d'entrée et de sortie

PRINT Code MUX 01h

PRINT était la première commande à utiliser le multiplexeur. Il lui permet de rester en contact avec sa partie résidente, d'ajouter des fichiers à la file en attente d'impression, de retirer des fichiers de la file d'attente ou de la supprimer intégralement.

Les programmes d'application peuvent exploiter ces fonctions lorsqu'ils ont l'intention d'exécuter des travaux d'impression volumineux sans immobiliser l'utilisateur. En général les imprimantes sont bien plus lentes que les logiciels, c'est là la seule raison qui justifie l'existence de PRINT.

Pour exploiteer PRINT un programme d'application doit émettre les impressions souhaitées dans un fichier puis insérer ce fichier dans la file d'attente de PRINT. Avec les fonctions de PRINT c'est très facile.

Fonctions multiplexées de la commande PRINT	
Fonction	**Rôle**
00h	Tester l'installation
01h	Ajouter un fichier à la file d'attente
02h	Retirer un ichier de la file d'attente

Fonctions multiplexées de la commande PRINT	
Fonction	**Rôle**
03h	Retirer tous les fichiers de la file d'attente
04h	Arrêter l'impression et lire l'état de l'impression
05h	Reprendre l'impression
06h	Tester l'imprimante

ASSIGN	Code MUX 06h
SHARE	Code MUX 10h
ANSI.SYS	Code MUX 1Ah
GRAFTABL	Code MUX B0h

Ces commandes ne comportent qu'une seule fonction numérotée 00h qui sert au test d'installation. Si à l'issue de l'appel, le registre AL contient la valeur FFh, c'est que le programme correspondant est installé. Le code MUX est à lire dans le tableau ci-dessus.

HIMEM.SYS	Code MUX 43h

Le driver HIMEM.SYS qui est responsable de la gestion de la mémoire selon le standard XMS possède deux fonctions multiplexées. La première qui porte le numéro 00h sert au test d'installation et ne vaut la peine d'être mentionnée que dans la mesure où contrairement à ses consoeurs elle ne retourne pas la valeur FFh mais 80h en cas d'installation de HIMEM.SYS. La deuxième fonction renvoie le point d'entrée dans le driver par lequel les différentes fonctions XMS seront appelées. Le multiplexeur joue un peu le rôle d'une clé qui ouvre la porte de HIMEM.SYS.

Fonctions multiplexées du driver HIMEM.SYS	
Fonction	**Rôle**
00h	Test d'installation
10h	Indique le point d'entrée pour l'appel des fonctions XMS

Le chapitre 12 consacré à l'accès à la mémoire étendue donne davantage de détail sur le fonctionnement du driver HIMEM.SYS

DOSKEY (à partir de la version 5.0 DOS) Code MUX 48h

A partir de la version 5.0 de DOS le programme résident DOSKEY permet de mémoriser et de rappeler les commandes saisies. Il autorise aussi l'usage d'abréviations par exploitation de macros.

Ces propriétés n'existent pas seulement au niveau de la ligne de commande de DOS elles sont aussi à la disposition des programmes d'application grâce à deux fonctions multiplexées. Le code de DOS est 48h, les numéros des deux fonctions sont 00H et 01h. La fonction 00h permet de tester si DOSKEY est installé. C'est la fonction de test d'installation usuelle.

La deuxième fonction permet à un programme d'application de demander à DOSKEY de prendre en compte la saisie effectuée comme si elle avait lieu au niveau de DOS. Le maniement des touches de direction verticales permet donc de repasser en revue les saisies faites jusqu'ici au niveau de DOS et les nouvelles saisies vont entrer dans le buffer où puise DOSKEY.

KEYB Code MUX ADh

Le driver de clavier KEYB.COM offre quatre fonctions. Elles permettent de lire le numéro de version de KEYB, la page de code courante et l'indicateur de nationalité

Fonctions multiplexées du driver de clavier KEYB

Fonction	Rôle
80h	Lire le numéro de version
81h	Fixe la page de code courante
82h	Fixe l'indicateur de nationalité
83h	Lit l'indicateur de nationalité

APPEND Code MUX B7h

Parmi les différentes commandes de DOS qui s'intègrent dans le multiplexeur, la commande APPEND se caractérise par le code le plus élevé. La numérotation des fonctions est un peu bizarre mais les services offerts correpondent à la palette des options disponibles au niveau de DOS

Fonctions multiplexées de la commande APPEND	
Fonction	**Rôle**
00h	Tester l'installation
02h	Tester la compatibilité avec la version 5.0 de DOS
04h	Lire le répertoire
06h	Lire le mode opératoire
07h	Fixer le mode opératoire
11h	Etablir la conversion en noms de fichiers complets

28. La programmation en réseau sous DOS

Les réseaux se sont considérablement répandus ces dernières années dans le monde des PC. L'expansion s'est amorcée dans les grandes sociétés et s'est poursuivie dans les entreprises moyennes pour finalement toucher également les petites sociétés. Les cartes de réseaux ne valent plus que quelques centaines de francs

La diffusion des réseaux s'accompagne naturellement d'un accroissement de la demande de logiciels capables de fonctionner sous réseau. Ce chapitre est justement destiné à vous initier à la réalisation de programmes fonctionnant sous réseau. Il s'agit en fait d'une simple introduction car traiter le sujet dans toute sa complexité dépasserait l'ambition de cet ouvrage. Mais les indications données devraient vous suffire pour réaliser votre premier programme capable de fonctionner sous réseau.

28.1. Les fondements

Les réseaux ont plusieurs tâches à remplir. Dans leur forme classique ils relient plusieurs PC (postes ou stations de travail) à un serveur qui est généralement constitué par un PC particulièrement puissant avec une grande capacité en mémoire vive et en disque dur. Le serveur offre ses ressources et différents services aux stations de travail qui sont branchées sur le réseau. Le plus important de ces services est le partage des fichiers grâce auquel certains fichiers ne sont pas gérés localement dans les différents postes de travail mais de façon centralisée dans le serveur de fichiers.

Les fichiers en question sont généralement issus de bases de données qui ont une très grande importance dans l'exploitation quotidienne et qui peuvent être traitées par plusieurs postes de travail en même temps. A cet égard, l'exemple de la gestion de stocks est particulièrement illustratif. Plusieurs agents spécialisés disposent d'autant de postes de travail pour consulter l'état des stocks qui est maintenu à jour dans une même base de données.

Les programmes susceptibles d'être appelés par l'ensemble des postes de travail du réseau sont également stockés sur le serveur pour économiser de la place sur les postes et minimiser le temps d'installation. Il est plus simple d'installer un seule fois un programme sur le serveur plutôt que de faire la même opération sur tous les postes de travail. Il suffit de penser à l'exemple de Windows.

Mais c'est justement cet exemple qui nous montre aussi qui malgré l'installation d'un programme unique sur le serveur, il reste indispensable de maintenir des fichiers sur les postes de travail. Car s'il est vrai que les fichiers programmes de Windows (WIN.COM, PROGMAN.EXE, PAINTBRUSH.EXE, etc...) sont les mêmes pour tous les utilisateurs, chacun d'eux a besoin de son propre WIN.INI ou SYSTEM.INI qui reflètera son environnement individuel.

Pourtant la plupart du temps ces mêmes fichiers sont aussi hébergés sur le serveur. Mais pour chaque utilisateur une zone individuelle est prévue dans laquelle il peut déposer ses fichiers privés sans avoir à craindre qu'un collègue n'y effectue des modifications. Cette garantie est apportée par le système d'exploitation du réseau, le composant le plus important du réseau.

Un serveur de réseau ne se contente pas de mettre son disque dur à la disposition des différents postes de travail. Il facilitera aussi les impressions et les communications en permettant aux postes d'accéder à une imprimante, à un modem ou à un autre réseau. Les investissements en matériel en seront réduits d'autant: il ne sera pas nécessaire par exemple de doter tous les postes d'une imprimante individuelle.

Depuis l'avènement d'OS/2, les serveurs SQL connaissent un grand succès. Il s'agit de serveurs fonctionnant sous OS/2 et qui offrent aux postes de travail un accès aux bases de données SQL. SQL veut dire "Structured Query Language", c'est un langage de requête pour interroger des bases de données relationnelles.

Le serveur SQL ne gère pas seulement les bases de données SQL mais il traite aussi les requêtes des postes de travail qui s'adressent à lui. Si un poste de travail émet une requête SQL pour demander les adresses complètes des clients dont le code postal est 75000, le serveur SQL renverra automatiquement les enregistrements désirés. On parle alors d'appel de procédure à distance (Remote Procedure Call). A partir des postes de travail une routine (procédure) du serveur est appelée pour prendre en charge un certain travail.

Le système d'exploitation du réseau

Qui dit réseau dit pratiquement Novell car, avec ses différentes versions de Netware cette société est incontestablement le plus grand fournisseur de systèmes de réseaux au monde. Mais il existe quand même d'autres sociétés qui fabriquent soit des systèmes compatibles avec Netware soit des systèmes particuliers, par exemple IBM, 3Com ou BanyanVines.

Un système d'exploitation de réseau comporte toujours deux éléments. D'abord un programme serveur qui gère le serveur de fichiers. Ensuite un logiciel installé sur chaque poste de travail pour lui permettre d'accéder au serveur. Ce logiciel se présente généralement sous forme de driver ou de programme résident qui intervient dans le noyau de DOS pour modifier le mode d'accès aux fichiers. Car les postes de travail continuent de fonctionner sous DOS et peuvent s'adresser au serveur comme à une unité de disquette ou de disque dur ordinaire.

Le logiciel de réseau intercepte tous les accès aux fichiers et ne transmet au noyau standard de DOS que ceux qui concernent les périphériques locaux. Supposons que pour un poste donné le serveur soit désigné par la lettre S. Le logiciel de réseau interceptera un accès au fichier "S:LETTRE.TXT" pour le détourner vers le serveur. Les fichiers du serveur pourront ainsi être traités comme des fichiers DOS ordinaires.

Un utilisateur n'aura cependant pas accès à tous les fichiers et répertoires stockés sur le disque dur du serveur. Sur un disque qui peut atteindre 1 Go leur nombre dépasse plusieurs milliers. L'administrateur du réseau fixe les répertoires et fichiers qui sont accessibles aux différents postes de travail. Les droits d'accès sont souvent liés à un mot de passe que l'utilisateur doit taper au moment où il accède pour la première fois au serveur ("Login").

Comme plusieurs utilisateurs auront ainsi le droit d'utiliser le serveur, ce dernier ne sera généralement pas exploité sous DOS. Le système traditionnel ne serait pas capable de satisfaire assez vite les postes de travail. Netware de Novell installe ainsi sur le serveur un système d'exploitation indépendant qui s'exécute en mode protégé et n'a plus besoin de DOS avec lequel il n'a rien à voir. Même le disque dur du serveur n'est plus géré selon le schéma classique de DOS mais avec un système d'enregistrement propre à Novell. Même si vous arrivez à lancer le serveur avec une disquette DOS, l'accès au disque dur sera impossible car son format est inconnu de DOS.

Pourtant si vous rédigez des programmes pour réseau sous DOS vous n'avez pas à vous faire de souci à ce sujet . A moins que vous ne développiez des outils pour Netware ce qui est évidemment une autre paire de manches.

De la même façon vous n'entrerez jamais en contact avec le matériel du réseau. Qu'il s'agisse d'un anneau à jeton, d'EtherNet ou d'ArcNet est sans importance. Ce qui compte, ce sont les interfaces des fonctions qui permettent d'accéder au serveur et à la communication avec les autres postes de travail.

Les interfaces

Trois interfaces se sont établies comme standards sur les PC. Il s'agit d'abord d'une série de fonctions rudimentaires offertes par DOS à partir de la version 3.0. Elles permettent à plusieurs postes de travail d'accéder simultanément à des fichiers du serveur et se trouvent analysées dans la section suivante.

Mais depuis la domination de Novell une autre interface s'est imposée qui s'appelle IPX/SPX.

Ce nom désigne un esnemble de fonctions mises à la disposition des programmes qui s'exécutent sur un poste de travail géré par NetWare. Les fonctions en question permettent aux postes de travail de communiquer entre eux, ce qui est indispensable pour réaliser des messageries électroniques ou pour commander les postes de travail à distance.

NetBios introduit par IBM poursuit le même objectif. Ce système est tellement répandu que NetWare comporte un émulateur NetBios qui convertit les appels à NetBios en autant d'appels à IPX/SPX, ce qui est possible car les deux interfaces reposent sur les mêmes principes.

Les programmes développés sou NetBios tournent aussi sous NetWare. C'est pourquoi je vous recommande d'utiliser NetBios si vous voulez monter dans les hautes sphères de la programmation sous réseau.

Réseau "Peer to peer"

La notion de réseau égalitaire ("peer to peer") connaît actuellement un important succès qui remet quelque peu en cause les principes traditionnels des réseaux car chaque poste de travail peut aussi jouer le rôle de serveur.

Cette configuration convient particulièrement aux réseaux de petite envergure, avec un trafic réduit, ce qui permet d'économiser l'investissement d'un serveur relativement cher. Chaque poste de travail est doté d'un sur-ensemble de DOS qui permet d'accéder aux autres postes de travail par les commandes de DOS et les instructions de traitement de fichiers habituelles.

Les programmes du type NetWare-Lite de Novell ou LANTastic de Microware en sont une illustration.

28.2. La programmation réseau sous DOS

Au centre de la programmation réseau sous DOS on trouve deux fonctions de l'API de DOS et le programme SHARE qui fait partie intégrante de DOS depuis la version 3.0.

SHARE contient les deux fonctions de l'API de DOS. Elles évitent tout conflit entre des programmes qui accéderaient simultanément aux mêmes fichiers. C'est bien là l'enjeu primaire de la programmation en DOS, à savoir le partage des fichiers. Très souvent d'ailleurs un même programme exécuté simultanément sur plusieurs postes de travail maintient ses fichiers sur le serveur de façon qu'ils puissent être exploités en même temps.

Prenons l'exemple d'un centre de tennis/squash. A la réception plusieurs employés gèrent la location des courts au moyen des différents postes de travail. Ils sont reliés par réseau à un serveur qui contient le fichier central des réservations pour les prochaines semaines. Sur l'ensemble des postes se déroule le même programme de réservation des courts. Comme les employés travaillent en même temps, différents problèmes vont se poser que nous allons étudier dans un moment.

Pour le moment restons-en à Share.

Test d'installation de Share

Comme la présence de SHARE est très importante pour l'exploitation des fonctions réseau de DOS, tout programme fonctionnant sous réseau devra en prendre connaissance au moment de son démarrage.

Notons à ce sujet que depuis la vresion 4.0 de DOS, le programme SHARE fait partie intégrante du système.

Si vous êtes sûr que votre programme ne tourne que sous DOS 4, 5 ou une version postérieure, vous pouvez vous passer du test d'installation de SHARE.

Ce test est en fait assez simple. Il est constitué par une unique instruction d'interruption. C'est la fonction 1000h du multiplexeur qu'il faut appeler car la partie résidente de Share se connecte au multiplexeur. Si cette fonction renvoie en AL la valeur FFh, Share est bien installé. Sur le mode de fonctionnement du multiplexeur, voyez le chapitre 27.

Verrouillage des enregistrements

Le verrouillage des enregistrements joue un rôle essentiel lors de l'utilisation simultanée d'un même fichier par plusieurs postes de travail. Reprenons l'exemple du centre de tennis. Plusieurs clients téléphonent en même temps pour réserver des courts. A la réception un certain nombre d'agents traitent les appels des clients. Ils affichent sur leurs postes de travail le planning des jours concernés. Cet affichage repose évidemment sur l'exploitation de fichiers mais la lecture simultanée des enregistrements ne pose aucun problème.

Les choses deviennent critiques à partir du moment où les enregistrements doivent subir des modifications. Supposons que l'un des agents prenant note d'une réservation reporte le nom du client dans l'enregistrement correspondant. Comme à cet instant précis le court était encore libre, un autre agent l'a peut-être également loué à un autre client. Un conflit se produit. Car le deuxième agent écrasera peut-être le nom de l'autre client parce que le logiciel n'était pas assez rapide pour lui indiquer à l'écran que le court venait d'être réservé.

Voilà le genre de situation que doit résoudre le verrouillage des enregistrements. Un programme désireux de modifier un enregistrement doit commencer par en prendre possession de façon exclusive, pour écarter toute tentative d'accès conflictuel. L'enregistrement est verrouillé pour les autres programmes ("Verrou enregistrement").

Techniquement on procède de la manière suivante. L'agent qui reçoit l'appel repère un court libre sur l'écran. Il actionne une touche de fonction pour indiquer au logiciel qu'il désire réserver ce court. L'enregistrement est alors verrouillé pour que le programme

puisse y opérer ses transformations. Un poste de travail extérieur ne peut plus accéder à l'enregistrement car toute nouvelle tentative de verrouillage serait vouée à l'échec.

Le programme peut y réagir en émettant un message à l'écran qui indique que le court en question est actuellement en cours de réservation. Mais il est vrai que le client au téléphone peut changer d'avis et préférer finalement une autre réservation. Il est donc recommandé au poste de travail dont la demande a été rejetée d'attendre un moment pour voir si l'enregistrement verrouillé ne va pas être libéré. Il répétera donc sa demande dans le cadre d'une boucle jusqu'à ce que l'enregistrement soit à nouveau libre ou jusqu'à ce qu'un délai limite soit dépassé.

L'enregistrement peut ensuite être relu et le programme pourra vérifier si le court est resté libre ou s'il a été loué à un autre client. A condition évidemment que l'enregistrement ait été libéré par le premier poste de travail, immédiatement après sa modification.

Pour partager les fichiers et leurs enregistrements entre plusieurs programmes sur autant de postes de travail, il faut respecter les règles suivantes:

- ✓ Avant d'être modifié, un enregistrement doit être verrouillé pour qu'aucun autre programme ne puisse y accéder
- ✓ Un enregistrement verrouillé doit être libéré aussitôt après sa modification pour que les autres programmes puissent l'exploiter à leur tour

Pour écarter toute possibilité de modification ou de lecture d'un enregistrement verrouillé, SHARE agit en liaison avec le système d'exploitation du réseau. Les appels aux fonctions inhibées sont rejetés par un signal d'erreur tant que l'enregistrement visé est verrouillé.

Si la théorie paraît simple, l'application se révèle complexe. Le verrouillage des enregistrements est exécuté par une seule fonction de DOS, à savoir la fonction 5Ch. Il faut lui fournir le handle du fichier ainsi que l'offset et la longueur de la zone à verrouiller à l'intérieur du fichier. Les fichiers étant souvent d'une taille supérieure à 64 Ko, ces paramètres sont communiqués sur 32 bits et répartis sur deux registres. Pour l'offset, le registre CX reçoit les 16 bits supérieurs, tandis que DX accueille les 16 bits inférieurs. Pour ce qui est de la taille de la zone, elle est répartie de la même façon sur les registres SI et DI.

Il n'est plus question ici d'enregistrements, mais ce n'est pas grave car une fois connue la longueur d'un enregistrement il est facile de trouver la zone où il débute à partir de son numéro, ce qui détermine aussi la longueur à verrouiller.

Paramètres communiqués à la fonction 5Ch	
Registre	**Signification**
Ah	Numéro de la fonction = 5Ch
AL	Mode: 0=Verrouiller 1=Déverrouiller
BX	Handle du fichier
CX:DX	Offset à l'intérieur du fichier (32 bits)
SI:DI	Longueur de la zone concernée (32 bits)

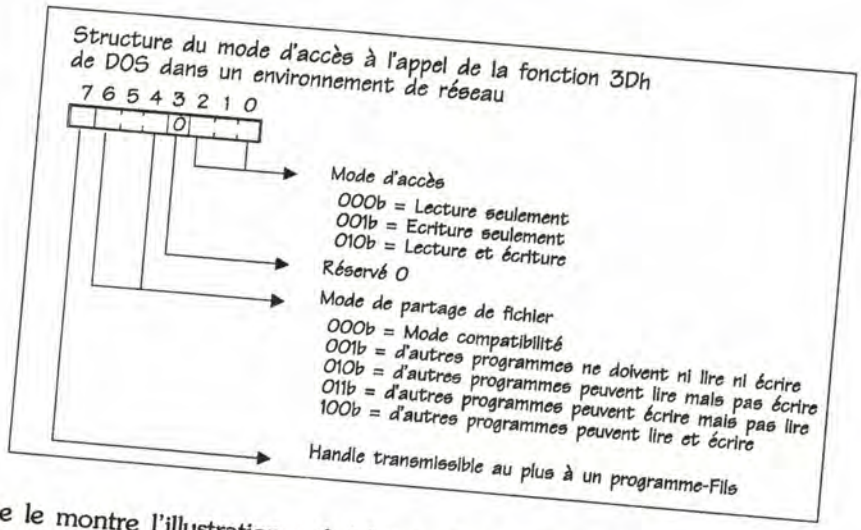
La même fonction servant aussi à libérer les zones verrouillées, le registre AL doit contenir le mode d'action souhaité: verrouillage ou déverrouillage. La fonction permet de verrouiller simultanément plusieurs zones de données, ce qui est très utile car les opérations de gestion impliquent souvent plusieurs enregistrements.

Avant de verrouiller un fichier il faut évidemment l'ouvrir. C'est cette ouverture qui fournit le handle attendu en BX par la fonction. Mais là aussi l'environnement du réseau impose des conditions particulières.

Partage de fichiers

Grâce à SHARE, DOS autorise non seulement le verrouillage des enregistrements mais aussi le verrouillage d'un fichier qui restreint l'accès des programmes externes à l'ensemble du fichier. Cette option est enclenchée à l'ouverture d'un fichier par la fonction 3Dh de DOS qui a un rôle général même en dehors des réseaux. Mais par rapport à une utilisation ordinaire, il faut fournir davantage d'informations en AL:

Paramètres communiqués à la fonction 3Dh	
Registre	**Signification**
Ah	Numéro de la fonction
AL	Mode d'accès (cf infra)
DS:DX	Pointeur sur le nom du fichier



Comme le montre l'illustration précédente, dans le cas d'un réseau les bits 4 à 6 définissent les opérations que les autres programmes ont le droit de faire sur le fichier. Vous pouvez ainsi éviter que "votre" fichier ne soit modifié, accorder aux autres programmes le droit d'y lire des informations ou même leur interdire tout accès pendant l'exécution de votre programme.

Les trois bits inférieurs prennent également une importance spéciale qu'on ne leur accorde généralement pas lorsqu'on programme simplement sous DOS. Ils définissent les opérations que le programme est susceptible d'exécuter lui-même sur le fichier. Par exemple si un programme indique le mode "Lecture seule", la fonction d'ouverture marchera même si le fichier a déjà été ouvert par un autre programme autorisé par le mode de partage.

Dans un environnement de réseau il faut éviter d'indiquer comme mode d'accès "Lecture et écriture" si on désire uniquement lire le fichier. Sinon l'accès peut être verrouillé parce qu'un autre programme a déjà ouvert le fichier et y interdit les écritures.

Si l'ouverture d'un fichier échoue, la fonction 3Dh retourne le code erreur 5, qui signifie " Accès refusé ". C'est ce code qui est également renvoyé par les fonctions de lecture et d'écriture lorsqu'une opération interdite a été demandée.

Si vous examinez les différents modes de partage de fichiers, vous serez frappé par la présence du mode dit de compatibilité. Ce mode est presque identique au mode 001b car il interdit la lecture et l'écriture du fichier à tous les autres programmes. Mais lorsqu'un de ces autres programmes tente une ouverture, il ne renvoie pas un code d'erreur ordinaire (5) mais appelle carrément le gestionnaire d'erreurs critiques de DOS. Sur l'écran s'affiche alors le message suivant :

```
Fonction SHARE interdite en lecture sur C:
(A)bandon, (R)eprise, (I)gnorer ?
```

Il est donc préférable lorsqu'on programme sous réseau de renoncer à ce mode et de choisir à la place le mode de partage des fichiers 001b si vous voulez interdire l'accès à tout autre programme.

Pratique de la programmation sous réseau

Pour montrer un exemple de programmation sous réseau en DOS, j'ai développé deux programmes qui exploitent les possibilités du verrouillage d'enregistrement et du verrouillage de fichier. Vous en trouverez les versions Pascal et C à la fin de cette section. Pour essayer ces programmes, vous n'avez pas besoin d'avoir un réseau. Windows 3.0 suffit. En effet si avant de lancer Windows 3.0 vous déclenchez SHARE, vous pouvez simuler autant de postes de travail que vous voulez en ouvrant des boîtes DOS et en y exécutant des programmes.

Dans le cas du verrouillage de fichier (programmes FLOCKP.PAS et FLOCKC.C), ce ne sera même pas nécessaire. Car le programme va ouvrir le même fichier deux fois de suite, ce qui revient à ouvrir ce programme par deux applications différentes.

Examinons d'abord les deux modules NETFILEP.PAS et NETFILEC.C qui contiennent les mêmes routines pour les deux programmes. Ils sont bâtis sur le même canevas mais ils se distinguent par la manière dont le fichier à traiter est transmis par l'appelant. Voici d'abord une liste des fonctions et procédures offertes:

Procédures et fonctions disponibles dans les modules NETFILEP.PAS et NETFILEC.C

Nom	Rôle
ShareInst	Teste si SHARE est installé
NetErrorMsg	Fournit le texte d'un message d'erreur
NetRewrite	Crée un fichier
NetReset	Ouvre un fichier existant
NetClose	Ferme un fichier
NetLock	Verrouille des enregistrements
NetUnLock	Libère des enregistrements verrouillés
Is_NetWRiteOK	Teste si les écritures sont autorisées
Is_NetReadOk	Teste si les lectures sont autorisées
Is_NetOpen	Teste si un fichier est ouvert
NetWrite	Ecrit des données dans un fichier
NetRead	Lit des données dans un fichier

Procédures et fonctions disponibles dans les modules NETFILEP.PAS et NETFILEC.C

Nom	Rôle
NetSeek	Positionne le pointeur du fichier

La version Pascal du module a été programmée sous forme d'unité. La version C est à inclure par une directive #INCLUDE, ce qui évite la compilation séparée du module et la constitution d'un fichier Make ou d'un projet.

Dans la version Pascal les différentes fonctions et procédures utilisent comme argument une variable fichier typée tout à fait ordinaire, semblable à celle des instructions ASSIGN READ, WRITE, etc...Malheureusement ces dernières ne peuvent être mises en service dans un réseau car elles ne fonctionnent pas si le fichier n'est pas ouvert à la fois en lecture et en écriture. C'est ainsi que l'unité NETFILEP est amenée à définir ses propres routines de lecture et d'écriture. L'utilisation des variables fichier reste pratique car on peut en déduire par exemple la taille des enregistrements. Elles contiennent aussi le handle ainsi que le mode d'accès du fichier qui y est habituellement enregistré par la procédure REWRITE du Pascal standard.

Dans la version C un fichier est défini par la structure de données suivante qui est transmise sous la forme d'une variable aux différentes fonctions du module NETFILEC.C:

```
typedef struct { unsigned int Handle, /* Handle du fichier */
                               RecS,   /* Taille de l'enregistrement */
} NFILE;                       Mode ; /* Mode d'accès */
```

Listing : NETFILEP.PAS

```
{*****************************************************************}
{*                                                               *}
{*                        N E T F I L E P . P A S                *}
{*---------------------------------------------------------------*}
{*  Fonction    : Propose différentes procédures et fonctions pour *}
{*                réaliser des programmes en réseau sous DOS     *}
{*---------------------------------------------------------------*}
{*  Auteur      : Michael Tischer                                *}
{*  Développé le : 10.02.1992                                    *}
{*  Dernière MAJ : 13.02.1991                                    *}
{*                                                               *}
{*****************************************************************}

unit NetFileP;

Interface

uses Crt, Dos;
                                  { Intègre les unités CRT et DOS }
const           {-- Types d'accès ------------------------------}

  FM_R  = 0;
  FM_W  = 1;                                  { Lecture seule }
  FM_RW = 2;                                  { Ecriture seule }
                    { Mode ordinaire en Pascal: lecture et écriture}
{-- Modes de partage ou types de protection ---------------------}

  SM_COMP = $00;
                                        { Mode de compatibilité }
  SM_RW   = $10;                { aucune protection des fichiers }
                  { Lecture et écriture externes interdites }
  SM_R    = $20;  { Lecture externe autorisée, écriture interdite }
  SM_W    = $30;  { Lecture externe interdite, écriture autorisée }
  SM_NO   = $40;                                { Tout est permis, protection}
                                        { par verrouillage d'enregistrement }
  {-- Erreurs possibles à l'appel des procédures ----------------}

  NE_OK             = $00;                              { Pas d'erreur }
  NE_FileNotFound   = $02;          { Erreur: Fichier non trouvé }
  NE_PathNotFound   = $03;          { Erreur: chemin d'accès non trouvé }
  NE_TooManyFiles   = $04;          { Erreur: trop de fichiers ouverts }
  NE_AccessDenied   = $05;          { Erreur: Accès au fichier refusé }
  NE_InvalidHandle  = $06;  { Erreur: Handle de fichier non valable }
  NE_AccessCode     = $07;          { Erreur: Type d'accès interdit }
  NE_Share = $20;                   { Non respect des règles de Share }
  NE_Lock           = $21;  { Erreur de (dé)verrouillage d'enreg. }
  NE_ShareBuffer    = $24;          { Débordement du buffer de Share }

var NetError : integer;       { Code d'erreur après interruption de DOS }

function ShareInst : boolean;                          { Share installé ? }

function NetErrorMsg( Numero : word ) : string;        { Message d'erreur }

procedure NetReset(     FNom  : string;     { Ouvre un fichier existant}
                        Mode  : integer;
                        RecS  : word;
                  var Fichier );
```

```
!
!procedure NetRewrite(      FNom  : string;   { Ouvre un nouveau fichier }
!                           Mode  : integer;
!                           RecS  : word;
!                       var Fichier );
!
!procedure NetClose( var Fichier );                    { Referme un fichier }
!
!function NetLock( var Fichier:          { Verrouille une zone d'un fichier }
!                      RecNo  : longint;
!                      Nombre : longint ) : boolean;
!
!function NetUnlock( var Fichier:          { Libère une zone d'un fichier }
!                      RecNo  : longint;
!                      Nombre : longint ) : boolean;
!
!function Is_NetOpen( var Fichier ) : boolean;          { Fichier ouvert ?}
!
!function Is_NetWriteOk( var Fichier ) : boolean;       { Enreg. autorisé ?}
!
!function Is_NetReadOk( var Fichier ) : boolean;        { Lecture permise ? }
!
!       { Les procédures Read, Write et Seek ne fonctionnent qu'avec }
!     { les fichiers dont le mode d'accès est Input-Output mais pour }
!   { l'utilisation en réseau les fichiers typés doivent être ouverts }
!     { dans d'autres modes, c'est pourquoi il faut faire appel aux }
!                          { procédures  de substitution qui suivent }
!
!procedure NetWrite( var Fichier;                       { Ecriture dans un fichier }
!                    var Donnees );
!
!procedure NetRead( var Fichier;                        { Lecture dans un fichier }
!                   var Donnees );
!
!procedure NetSeek( var Fichier;                        { Positionnement du pointeur }
!                       RecNo : longint );
!
!implementation
!
!const {-- Numéros des fonctions pour les appels à DOS ----------------}
!
!     FCT_OPEN     = $3D;  { Fonction: Ouvre un fichier avec un handle }
!     FCT_CLOSE    = $3E;{ Fonction: Referme un fichier avec un handle }
!     FCT_CREATE   = $3C;   { Fonction: Crée un fichier avec un handle }
!     FCT_WRITE    = $40;              { Fonction: Ecrit dans un fichier }
!     FCT_READ     = $3F;              { Fonction: Lit dans un fichier }
!     FCT_LSEEK    = $42;            { Fonction: Positionne le pointeur }
!     FCT_REC_LOCK = $5C;    { Fonction: Verrouille des enregistrements}
!
!              {-- Numéros des fonctions pour autres interruptions --}
!
!     MULTIPLEX    = $2F;                { Interruption du multiplexeur }
!     FCT_SHARE    = $1000;              { Test d'installation de Share }
!
!     {-- Marquage  des fichiers sous TP  ------------------------}
!
!     FMCLOSED     = $D7B0;                          { Fichier fermé }
!     FMINPUT      = $D7B1;              { Fichier ouvert en lecture }
!     FMOUTPUT     = $D7B2;           { Fichier ouvert en écriture }
!     FMINOUT      = $D7B3;  { Fichier ouvert en lecture et écriture }
!
!{*****************************************************************}
!{* ShareInst   : Test d'installation de Share                    *}
!{* Entrée      : néant                                           *}
!{* Sortie      : true si Share installé                          *}
!{* Var. globale NetError/W (Code erreur après appel)             *}
!{*****************************************************************}
!
!function ShareInst : boolean;
!
!var regs   : registers;          { Registres pour gérer les interruptions }
!
!begin
! regs.ax := FCT_SHARE;                          { Teste si Share est présent }
! intr( MULTIPLEX, regs );       { Appelle l'interruption du multiplexeur }
! ShareInst := ( regs.al = $FF );                { Exploite le résultat }
! NetError := NE_OK;                                        { pas d'erreur }
!end;
!
!{*****************************************************************}
!{* NetErrorMsg : Texte des messages d'erreur                     *}
!{* Entrée      : Code d'erreur                                   *}
!{* Sortie      : Explication                                     *}
!{*****************************************************************}
!
!function NetErrorMsg( Numero : word ) : string;
!
!var Sdummy : string;
!
!begin
!  case Numero of
!    NE_OK           : NetErrorMsg := 'Pas d''erreur                ';
!    NE_FileNotFound : NetErrorMsg := 'Fichier non trouvé           ';
!    NE_PathNotFound : NetErrorMsg := 'Chemin d''accès non trouvé   ';
!    NE_TooManyFiles : NetErrorMsg := 'Trop de fichiers ouverts     ';
!    NE_AccessDenied : NetErrorMsg := 'Accès au fichier refusé      ';
!    NE_InvalidHandle: NetErrorMsg := 'Handle de fichier non valide ';
!    NE_AccessCode   : NetErrorMsg := 'Type d''accès interdit       ';
!    NE_Share        : NetErrorMsg := 'Violation des règles de SHARE';
!    NE_Lock         : NetErrorMsg := 'Erreur de verrouillage       ';
!    NE_ShareBuffer  : NetErrorMsg := 'Débordement du buffer SHARE  ';
!    else              begin
!                        str( Numero, Sdummy );
!                NetErrorMsg := 'Erreur DOS: ' + Sdummy;
!                      end;
!
!  end;
!end;
!
!{*****************************************************************}
!{* NetRewrite    : Crée un fichier                               *}
!{* Entrées       : Nom fichier, mode ouverture, taille enregistrement *}
!{* Sortie        : Fichier ouvert                                *}
!{* Var. globales : NetError/W (code d'erreur à l'issue de l'appel *}
!{*****************************************************************}
!
!procedure NetRewrite(      FNom  : string;
!                           Mode  : integer;
!                           RecS  : word;
!                       var Fichier );
!
!var regs   : registers;          { Registres pour gérer les interruptions }
!    FNom2 : string;                          { Nom du fichier pour usage local }
!
!begin
!  FNom2 := FNom + #0;                { Copie et complète le nom du fichier }
!  with regs do
!    begin
!      ds := seg( FNom2[ 1 ] );                              { Affecte le nom }
!      dx := ofs( FNom2[ 1 ] );
!      ah := FCT_CREATE;                 { Numéro de la fonction "Créer fichier" }
!      cx := 0 ;                                          { Attribut du fichier }
!      msdos( regs );                                          { Interruption }
!      if ( ( flags and fcarry ) = 0 ) then          { Ouverture réussie ? }
!        begin
!        bx := ax;                                             { Handle en BX }
!        ah := FCT_CLOSE;          { Numéro de la fonction "Fermer fichier"}
!          msdos( regs );
!      if ( ( flags and fcarry ) = 0 ) then          { Fermeture réussie ? }
!        NetReset( FNom, Mode, Recs, Fichier )          { Rouvre le fichier }
!          else
!        NetError := ax;                                { Note le code d'erreur }
!        end
!      else
!    NetError := ax;                                  { Note le code d'erreur }
!    end;
!end;
!
!{*****************************************************************}
!{* NetReset      : Ouvre un fichier préexistant                  *}
!{* Entrées       : Nom fichier, mode ouverture, taille enregistrement *}
!{* Sortie        : Fichier ouvert                                *}
!{* Var. globales: NetError/W (Code d'erreur après appel)         *}
!{*****************************************************************}
!
!procedure NetReset(        FNom  : string;
!                           Mode  : integer;
!                           RecS  : word;
!                       var Fichier );
!
!var regs : registers;          { Registres pour gérer les interruptions }
!
!begin
!  FNom := FNom + #0;          { Le nom du fichier doit se terminer par #0}
!  with regs do
!    begin
!      ds := seg( FNom[ 1 ] );                       { Affecte le nom du fichier }
!      dx := ofs( FNom[ 1 ] );
!      ah := FCT_OPEN;                   { Numéro de la fonction "Ouvrir fichier" }
!      al := Mode;          { Octet d'état pour type d'accès et verrouillage }
!      msdos( regs );                                          { Interruption }
!      if ( ( flags and fcarry ) = 0 ) then          { Ouverture réussie ? }
!        begin                                                  { Pas d'erreur }
!      NetError := NE_OK;
!        with filerec( Fichier ) do
!          begin
!          move( FNom[ 1 ], filerec( Fichier ).Name,(Affecte nom fichier)
!            length( FNom ) );
!          Handle := ax;                                     { Handle du fichier }
!          RecSize := RecS;                                    { Fixe la taille }
!          case ( Mode and $0F ) of                { Fixe le mode en Pascal }
!                FM_R  : Mode := FMINPUT;
!                FM_W  : Mode := FMOUTPUT;
!                FM_RW : Mode := FMINOUT;
!            end;
!          end;
!      end
```

```
    else
      begin
        NetError := ax;                          { Mémorise le code d'erreur }
      filerec( Fichier ).Mode := FMCLOSED;       { Fichier non ouvert }
      end;
  end;
end;

{*********************************************************************}
{* NetClose : Ferme un fichier                                       *}
{* Entrée   : Fichier                                                *}
{* Sortie   : néant                                                  *}
{*********************************************************************}

procedure NetClose( var Fichier );

var regs : registers;              { Registres pour gérer  les interruptions }

begin
  if ( filerec( Fichier ).Mode <> FMCLOSED ) then   { Fichier ouvert ? }
    begin
      with regs do
        begin
      ah := FCT_CLOSE;          { Numéro de la fonction "Fermer fichier" }
      bx := FileRec( Fichier ).Handle;                { Handle du fichier }
      msdos( regs );                                  {  Interruption }
        end;
      FileRec( Fichier ).Mode := FMCLOSED;
      NetError := NE_OK;                              { Fichier fermé }
    end                                               { Pas d'erreur }
  else
    NetError := NE_InvalidHandle;
end;                                          { Fichier non ouvert }

{*********************************************************************}
{* Locking     : Verrouillage ou déverrouillage d'une zone du fichier *}
{* Entrée      : Handle fichier, opération, offset depuis le début du *}
{*               fichier, taille de la zone concernée                *}
{* Sortie      : true si réussi                                      *}
{* Var globales: NetError/W (Code d'erreur à l'issue de l'appel )    *}
{* Info        : Usage exclusivement interne , appel par NetLock     *}
{*               et NetUnlock                                        *}
{*********************************************************************}

function Locking( Handle    : word;
                  Operation : byte;
                  Offset    : longint;
                  Longueur  : longint ) : boolean;

var regs : registers;              { Registres pour gérer  les interruptions }

begin
  with regs do
    begin
      ah := FCT_REC_LOCK;  { Numéro de la fonction pour l'interruption }
      al := Operation;                                  { 0 = Lock, 1 = Unlock }
      bx := Handle;                                     { Handle du fichier }
      cx := offset shr 16;                              { Mot fort de l'offset }
      dx := offset and $FFFF;                           { Mot faible de l'offset }
      si := Longueur shr 16;                            { Mot fort de la longueur }
      di := Longueur and $FFFF;                         { Mot faible de la longueur }
      msdos( regs );                                    {  Interruption }
      if ( ( flags and fcarry ) = 0 ) then ( (Dé)verrouillage réussi ? }
        begin
      Locking := true;
        NetError := NE_OK;                              { Pas d'erreur }
        end
      else
        begin
          Locking := false;
      NetError := ax;
        end;                                  { Mémorise le code d'erreur }
    end;
end;

{*********************************************************************}
{* NetLock     : Verrouillage d'enregistrements                      *}
{* Entrée      : Fichier, numéro premier enregistrement, nb enreg.   *}
{* Sortie      : true si opération réussie                           *}
{* Var. globales : NetError/W (Code d'erreur à l'issue de l'appel)   *}
{*********************************************************************}

function NetLock( var Fichier;
                  RecNo  : longint;
                  Nombre : longint ) : boolean;

begin
  NetLock := Locking( filerec( Fichier ).Handle, 0,
                      filerec( Fichier ).Recsize * RecNo,
                      filerec( Fichier ).Recsize * Nombre );
end;

{*********************************************************************}
{* NetUnLock   : Libère des enregistrements verrouillés              *}
{* Entrée      : Fichier, numéro premier enregistrement, nb enreg.   *}
{* Sortie      : true si opération réussie                           *}
{* Var. globales : NetError/W (Code d'erreur à l'issue de l'appel)   *}
{*********************************************************************}

function NetUnlock( var Fichier;
                    RecNo  : longint;
                    Nombre : longint ) : boolean;

begin
  NetUnLock := Locking( filerec( Fichier).Handle, 1,
                        filerec( Fichier ).Recsize * RecNo,
                        filerec( Fichier ).Recsize * Nombre );
end;

{*********************************************************************}
{* Is_NetWriteOk : Teste si l'écriture dans un fichier est autorisée *}
{* Entrée        : Fichier                                           *}
{* Sortie        : true si écriture autorisée                        *}
{*********************************************************************}

function Is_NetWriteOk( var Fichier ) : boolean;

begin
  with filerec( Fichier ) do
    Is_NetWriteOk := ( Mode = FMOUTPUT ) or ( Mode = FMINOUT );
end;

{*********************************************************************}
{* Is_NetReadOk : Teste si la lecture d'un fichier est autorisée     *}
{* Entrée       : Fichier                                            *}
{* Sortie       : true si lecture autorisée                          *}
{*********************************************************************}

function Is_NetReadOk( var Fichier ) : boolean;

begin
  with filerec( Fichier ) do
    Is_NetReadOk := ( Mode = FMINPUT ) or ( Mode = FMINOUT );
end;

{*********************************************************************}
{* Is_NetOpen  : Teste si un fichier est ouvert                      *}
{* Entrée      : Fichier                                             *}
{* Sortie      : true si fichier ouvert                              *}
{*********************************************************************}

function Is_NetOpen( var Fichier ) : boolean;

begin
  with filerec( Fichier ) do
    Is_Netopen := ( Mode = FMINPUT ) or ( Mode = FMOUTPUT ) or
                  ( Mode = FMINOUT );
end;

{*********************************************************************}
{* NetWrite : Ecrit des données dans un fichier                      *}
{* Entrée   : Fichier, Donnees                                       *}
{* Sortie   : néant                                                  *}
{* Info     : L'écriture n'est possible avec les procédures de pascal *}
{*            que si le fichier a été ouvert en mode Input-Output    *}
{* Attention: pas de contrôle de type                                *}
{*********************************************************************}

procedure NetWrite( var Fichier;
                    var Donnees );

var regs : registers;              { Registres pour gérer  les interruptions }

begin
  with regs do
    begin
      ds := seg( Donnees );
      dx := ofs( Donnees );                         { Adresse des données }
      ah := FCT_MRITE;    { Numéro de la fonction "Ecrire dans fichier"}
      bx := filerec( Fichier ).Handle;              { Handle du fichier }
      cx := filerec( Fichier ).Recsize;             { Nombre d'octets }
      msdos( regs );                                {  Interruption }
      if ( ( flags and fcarry ) = 0 ) then          { Ecriture réussie ? }
        NetError := NE_OK
      else                                          { Pas d'erreur }
        NetError := ax;
    end;                                  { Mémorise le code d'erreur }
end;

{*********************************************************************}
{* NetRead  : Lit des données dans un fichier                        *}
{* Entrée   : Fichier, variable pour mémoriser les données           *}
{* Sortie   : Données                                                *}
{* Info     : La lecture n'est possible avec les procédures de Pascal *}
{*            que si le fichier a été ouvert en mode Input-Output    *}
{* Attention : Pas de contrôle de type                               *}
{*********************************************************************}
```

```
{
procedure NetRead( var Fichier:
                   var Donnees );

var regs : registers;          { Registres pour gérer  les interruptions }

begin
  with regs do
    begin
      ds := seg( Donnees );                          { Adresse des données }
      dx := ofs( Donnees );
      ah := FCT_READ;              { Numéro de la fonction "Lire fichier"}
      bx := filerec( Fichier ).Handle;              { Handle du fichier }
      cx := filerec( Fichier ).Recsize:             { Nombre d'octet }
      msdos( regs );                                {  Interruption }
      if ( ( flags and fcarry ) = 0 ) then          { Ecriture réussie ? }
                                                    { Pas d'erreur }
        NetError := NE_OK
      else                                          { Mémorise le code d'erreur }
        NetError := ax;
    end;
end;
{*****************************************************************}
{* NetSeek : Positionne le pointeur du fichier                   *}
{* Entrée  : Fichier, RecordNumero                               *}
{* Sortie  : néant                                               *}
{* Info    : La procédure Seek de Pascal n'est possible que si le*}
{*           fichier a été ouvert en mode Input-Output           *}
{*****************************************************************}

procedure NetSeek( var Fichier;
                   RecNo : longint );

var regs : registers;          { Registres pour gérer  les interruptions }

begin
  with regs do
    begin
      ah := FCT_LSEEK;{ Numéro de la fonction "Positionner le pointeur"}
      al := 0;       { Position absolue par rapport au début du fichier }
      bx := filerec( Fichier ).Handle;              { Handle du fichier }
      RecNo := RecNo * filerec( Fichier ).Recsize;  { Offset en octets }
      cx := RecNo shr 16;                           { Mot fort de l'offset }
      dx := RecNo and $FFFF;                        { Mot faible de l'offset }
      msdos( regs );                                {  Interruption }
      if ( ( flags and fcarry ) = 0 ) then          { Opération réussie ? }
                                                    { Pas d'erreur }
        NetError := NE_OK
      else                                          { Mémorise le code d'erreur }
        NetError := ax;
    end;
end;

{-- Partie Initialisation de l'unité ---------------------------------}

begin
end.
```

Listing : NETFILEC.C

```
/*****************************************************************/
/*                        N E T F I L E C . C                    */
/*-------------------------------------------------------------- */
/*   Fonction      : Propose différentes procédures et fonctions */
/*                   pour réaliser des programmes en réseau sous DOS */
/*-------------------------------------------------------------- */
/*   Modèle mémoire: SMALL                                       */
/*-------------------------------------------------------------- */
/*   Auteur        : Michael Tischer                             */
/*   Développé le  : 10.02.1992                                  */
/*   Dernière MAJ  : 13.02.1991                                  */
/*-------------------------------------------------------------- */
/*   Microsoft C   : Le message "Segment lost in conversation" est */
/*                   malheureusement inévitable mais il ne gêne  */
/*                   pas la compilation                          */
/*****************************************************************/

#include <dos.h>
#include <string.h>
#include <stdlib.h>

/*--- Macros ------------------------------------------------------*/

#ifdef FP_OFF
  #undef FP_OFF
  #undef FP_SEG
#endif

#define FP_OFF(fp)      ((unsigned)(fp))
#define FP_SEG(fp)      ((unsigned)((unsigned long)(fp) >> 16))

/*--- Constantes --------------------------------------------------*/

/*-- Valeurs booléennes-----------------------------------------*/

#define TRUE  ( 1 == 1 )
#define FALSE ( 0 == 1 )

/*-- Types d'accès aux fichiers---------------------------------*/

#define FM_R      0                          /* Lecture seule */
#define FM_W      1                          /* Ecriture seule */
#define FM_RW     2                          /* Lecture et écriture */

/*-- Modes de partage ou types de protection --------------------*/

#define SM_COMP 0x00 /* Mode de compatibilité, aucune protection  */
#define SM_RW   0x10   /* Lecture et écriture externes interdites */
#define SM_R    0x20 /* Lecture externe autorisée, écriture interdite */
#define SM_W    0x30 /* Lecture externe interdite, écriture autorisée */
#define SM_NO   0x40  /* Tout est permis, protection par verrouillage */
                                           /* des enregistrements */


/*-- Erreus possibles à l'appel des procédures ------------------*/

#define NE_OK             0x00                  /* Pas d'erreur */
#define NE_FileNotFound   0x02          /* Erreur : Fichier non trouvé */
#define NE_PathNotFound   0x03    /* Erreur : Chemi d'accès non trouvé */
#define NE_TooManyFiles   0x04     /* Erreur: Trop de fichiers ouverts */
#define NE_AccessDenied   0x05       /* Erreur: Accès au fichier refusé */
#define NE_InvalidHandle 0x06/* Erreur: Handle de fichier non valable */
#define NE_AccessCode     0x07        /* Erreur: Type d'accès interdit */
#define NE_Share          0x20      /* Violation des règles de Share */
#define NE_Lock           0x21/* Erreur de (dé)verrouillage d'un enreg.*/
#define NE_ShareBuffer    0x24      /* Débordement du buffer de Share */

/*-- Numéros des fonctions pour les appels à DOS ----------------*/

#define FCT_OPEN    0x3D/* Fonction: Ouvre un fichier avec un handle */
#define FCT_CLOSE   0x3E/* Fonction: Ferme un fichier avec un handle */
#define FCT_CREATE  0x3C /* Fonction: Crée un fichier avec un handle */
#define FCT_WRITE   0x40          /* Fonction: Ecrit dans un fichier */
#define FCT_READ    0x3F          /* Fonction: Lit dans un fichier */
#define FCT_LSEEK   0x42       /* Fonction: Positionne le pointeur */
#define FCT_REC_LOCK 0x5C /* Fonction: Verrouille des enregistrements */

/*-- Numéros des fonctions pour autres interruptions ------*/

#define MULTIPLEX   0x2F              /* Interruption du multiplexeur */
#define FCT_SHARE   0x1000            /* Test d'installation de Share */

/*-- Marquage des fichiers (valeurs semblables à Turbo-Pascal ) ------*/

#define FMCLOSED    0xD7B0                          /* Fichier fermé */
#define FMINPUT     0xD7B1              /* Fichier ouvert en lecture */
#define FMOUTPUT    0xD7B2             /* Fichier ouvert en écriture */
#define FMINOUT     0xD7B3  /* Fichier ouvert en lecture et écriture */

/*--- Déclarations de types ---------------------------------------*/

typedef struct { unsigned int Handle, RecS, Mode; } NFILE;

/*--- Variables globales ------------------------------------------*/

int         NetError;     /* Code d'erreur après interruption de DOS */
union  REGS regs;          /* Registres pour gérer les interruptions */
struct SREGS sregs;              /* Registres de segment même fonction */

/*****************************************************************/
/* ShareInst    : Test d'installation de Share                   */
/* Entrée       : néant                                          */
/* Sortie       : TRUE si Share installé                         */
/* Var. globale : NetError/W (Code d'erreur)                     */
/*****************************************************************/

int ShareInst( void )
```

```
!{
! regs.x.ax = FCT_SHARE;                    /* Teste si Share est présent */
! int86( MULTIPLEX, &regs, &regs );         /* Interruption multiplexeur */
! NetError = NE_OK;                                    /* Pas d'erreur */
! return ( regs.h.al == 0xFF );                  /* Détermine le retour */
!}
!
!/*****************************************************************/
!/* NetErrorMsg : Textes des messages d'erreur                    */
!/* Entrée      : cf infra                                        */
!/* Sortie      : cf infra                                        */
!/*****************************************************************/
!
!void NetErrorMsg( int  Numero,                    /* Code d'erreur */
!         char *Text )                          /* Texte du message */
!{
! char Sdummy[ 5 ];
!                                                 /* Code d'erreur */
! switch ( Numero )
! {
!  case NE_OK             : strcpy( Text, "Pas d'erreur                " );
!                           break;
!  case NE_FileNotFound   : strcpy( Text, "Fichier non trouvé          " );
!                           break;
!  case NE_PathNotFound   : strcpy( Text, "Chemin d'accès non trouvé   " );
!                           break;
!  case NE_TooManyFiles   : strcpy( Text, "Trop de fichiers ouverts    " );
!                           break;
!  case NE_AccessDenied   : strcpy( Text, "Accès au fichier refusé     " );
!                           break;
!  case NE_InvalidHandle: strcpy( Text, "Handle de fichier non valide " );
!                           break;
!  case NE_AccessCode     : strcpy( Text, "Type d'accès interdit       " );
!                           break;
!  case NE_Share          : strcpy( Text, "Violation des règles de Share " );
!                           break;
!  case NE_Lock           : strcpy( Text, "Erreur de verrouillage      " );
!                           break;
!  case NE_ShareBuffer    : strcpy( Text, "Débordement du buffer de Share" );
!                           break;
!  default                : {
!                            itoa( Numero, Sdummy, 2 );
!              strcpy( Text, "Erreur DOS:                 " );
!                            strcat( Text, Sdummy );
!                           }
! }
!}
!
!/*****************************************************************/
!/* NetReset    : Ouvre un fichier préexistant                    */
!/* Entrées     : cf infra                                        */
!/* Sortie      : cf infra                                        */
!/* Var. globale : NetError/W (Code d'erreur)                     */
!/*****************************************************************/
!
!void NetReset( char far     *FNom,
!          unsigned int Mode,                    /* Nom du fichier */
!          unsigned int RecS,                    /* Mode d'ouverture */
!          NFILE       *Fichier )          /* Taille d'enregistrement */
!{                                          /* Pointeur sur fichier */
! regs.x.dx = FP_OFF( FNom );
! regs.h.ah = FCT_OPEN;          /* Adresse du nom du fichier */
! regs.h.al = Mode;              /* Numéro de la fonction "Ouvrir fichier" */
! sregs.ds  = FP_SEG( FNom );                          /* Attribut */
! intdosx( &regs, &regs, &sregs );
! if ( !regs.x.cflag )                              /* Interruption */
! {                                        /* Ouverture réussie ? */
!  Fichier->Handle = regs.x.ax;
!  Fichier->RecS = RecS;                  /* Mémorise le handle */
!  switch ( Mode & 0x0F )                 /* et la taille de l'enreg. */
!  {                                      /* Fixe le mode du fichier */
!   case FM_R  : Fichier->Mode = FMINPUT;
!                break;
!   case FM_W  : Fichier->Mode = FMOUTPUT;
!                break;
!   case FM_RW : Fichier->Mode = FMINOUT;
!                break;
!  }
!  NetError = NE_OK;
! }
! else                                                 /* Pas d'erreur */
!  NetError = regs.x.ax;
!}                                    /* Mémorise le code d'erreur */
!
!/*****************************************************************/
!/* NetRewrite  : Crée un fichier                                 */
!/* Entrées     : cf infra                                        */
!/* Sortie      : cf infra                                        */
!/* Var. globale : NetError/W (Code d'erreur)                     */
!/*****************************************************************/
!
!void NetRewrite( char far     *FNom,
!          unsigned int Mode,                    /* Nom du fichier */
!                                                /* Mode d'ouverture*/
!          unsigned int RecS,                    /* Taille de l'enregistrement */
!          NFILE       *Fichier )          /* Pointeur sur le fichier */
!{
! regs.x.dx = FP_OFF( FNom );                  /* Adresse du nom du fichier */
! regs.h.ah = FCT_CREATE;        /* Numéro de la fonction "ouvrir fichier"*/
! regs.x.cx = 0 ;                                      /* Attribut */
! sregs.ds  = FP_SEG( FNom );
! intdosx( &regs, &regs, &sregs );
! if ( !regs.x.cflag )                              /* Interruption */
! {                                        /* Ouverture réussie */
!  regs.x.bx = regs.x.ax;
!  regs.h.ah = FCT_CLOSE;        /* Handle en BX */
!  intdos( &regs, &regs );       /* Numéro de la fonction "Fermer fichier " */
!  if ( !regs.x.cflag )
!   NetReset( FNom, Mode, RecS, Fichier );          /* Opération réussie ? */
!  else                                             /* Rouvre le fichier */
!   NetError = regs.x.ax;
!  }
! else
!  NetError = regs.x.ax;
!}                                    /* Mémorise le code d'erreur */
!
!/*****************************************************************/
!/* NetClose  : Ferme un fichier                                  */
!/* Entrées   : cf infra                                          */
!/* Sortie    : néant                                             */
!/*****************************************************************/
!
!void NetClose( NFILE *Fichier )
!{                                                /* Fichier à fermer */
! if ( Fichier->Mode != FMCLOSED )
! {                                                /* Fichier ouvert?*/
!  regs.x.bx = Fichier->Handle;
!  regs.h.ah = FCT_CLOSE;                        /* Affecte le handle */
!  intdos( &regs, &regs );       /* Numéro de la fonction "Ferme fichier"*/
!  if ( !regs.x.cflag )
!  {                                      /* Fermeture réussie  ? */
!   Fichier->Handle = 0;
!   Fichier->Mode = FMCLOSED;                    /* Efface le handle */
!   NetError = NE_OK;                            /* Fichier fermé    */
!  }                                                  /* Pas d'erreur */
!  else
!   NetError = regs.x.ax;
! }
! else
!  NetError = NE_InvalidHandle;
!}                                                /* Fichier non ouvert  */
!
!/*****************************************************************/
!/* Locking     : Verrouillage ou déverrouillage d'une zone de fichier */
!/* Entrées     : cf infra                                        */
!/* Sortie      : true si réussi                                  */
!/* Var. globale: NetError/W (Code d'erreur )                     */
!/* Info        : Utilisation exclusivement interne réservée à NetLock */
!/*               et NetUnlock                                    */
!/*****************************************************************/
!
!int Locking( int          Handle,                /* Handle du fichier */
!         int          Operation,                /* Type d'opération */
!         unsigned long Offset,/*Offset en octets depuis début fichier */
!         unsigned long Longueur )    /* Longueur de la zone en octets */
!{
! regs.h.ah = FCT_REC_LOCK;     /* Numéro de la fonction d'interruption */
! regs.h.al = Operation;        /* 0 = Verrouillage, 1 = Déverrouillage */
! regs.x.bx = Handle;                              /* Handle du fichier */
! regs.x.cx = Offset >> 16;                  /* Mot fort de l'offset */
! regs.x.dx = Offset & 0xFFFF;               /* Mot faible de l'offset */
! regs.x.si = Longueur >> 16;                /* Mot fort de la longueur */
! regs.x.di = Longueur & 0xFFFF;             /* Mot faible de la longueur */
! intdos( &regs, &regs );                                /* Interruption */
! if ( ! regs.x.cflag )                          /* Opération réussie ?*/
! {
!  NetError = NE_OK;
!  return TRUE;
! }
! else                                                 /* Pas d'erreur */
! {
!  NetError = regs.x.ax;
!  return FALSE;                          /* Mémorise le code d'erreur */
! }                                                     /* Erreur */
!}
!
!/*****************************************************************/
!/* NetUnLock   : Libère des enregistrements verrouillés          */
!/* Entrées     : cf infra                                        */
!/* Sortie      : true si réussi                                  */
!/* Var. globale : NetError/W (Code d'erreur )                    */
!/*****************************************************************/
!
!int NetUnLock( NFILE       *Fichier,                /* Fichier */
!          unsigned long RecNo,          /* Numéro d'enregistrement */
!          unsigned long Nombre )        /* Nombre d'enregistrements */
!{
```

```
| return Locking( Fichier->Handle, 1, Fichier->RecS * RecNo,
|                 Fichier->RecS * Nombre );
|}
|
|/*****************************************************************/
|/* NetLock      : Verrouille des enregistrements                 */
|/* Entrées      : cf infra                                       */
|/* Sortie       : true si réussi                                 */
|/* Var. globale : NetError/N (Code d'erreur )                    */
|/*****************************************************************/
|
|int NetLock( NFILE        *Fichier,                  /* Fichier */
|         unsigned long RecNo,             /* Numéro d'enregistrement */
|         unsigned long Nombre )           /* Nombre d'enregistrements */
|{
| return Locking( Fichier->Handle, 0, Fichier->RecS * RecNo,
|                 Fichier->RecS * Nombre );
|}
|
|/*****************************************************************/
|/* Is_NetReadOk : Teste si lecture autorisée                     */
|/* Entrée       : cf infra                                       */
|/* Sortie       : true si lecture autorisée                      */
|/*****************************************************************/
|
|int Is_NetReadOk( NFILE *Fichier )
|{
| return ( ( Fichier->Mode == FMINPUT ) ||
|          ( Fichier->Mode == FMINOUT ) );
|}
|
|/*****************************************************************/
|/* Is_NetOpen   : Teste si fichier ouvert                        */
|/* Entrée       : cf infra                                       */
|/* Sortie       : true si fichier ouvert                         */
|/*****************************************************************/
|
|int Is_NetOpen( NFILE *Fichier )
|{
| return ( ( Fichier->Mode == FMOUTPUT ) ||
|          ( Fichier->Mode == FMINPUT ) ||
|          ( Fichier->Mode == FMINOUT ) );
|}
|
|/*****************************************************************/
|/* Is_NetWriteOk : Teste si écriture autorisée                   */
|/* Entrée        : cf infra                                      */
|/* Sortie        : true si écriture autorisée                    */
|/*****************************************************************/
|
|int Is_NetWriteOk( NFILE *Fichier )
|{
| return ( ( Fichier->Mode == FMOUTPUT ) ||
|          ( Fichier->Mode == FMINOUT ) );
|}
|
|/*****************************************************************/
|/* NetWrite : Ecrit des données dans un fichier                  */
|/* Entrées  : cf infra                                           */
|/* Sortie   : néant                                              */
|/*****************************************************************/
|
|void NetWrite( NFILE     *Fichier,                     /* Fichier */
|           void  far *Donnees )            /* Pointeur sur données */
|{
| regs.x.dx = FP_OFF( Donnees );      /* Adresse de la zone de données */
| regs.h.ah = FCT_WRITE; /* Numéro de la fonction "Ecrire dans fichier"*/
| regs.x.bx = Fichier->Handle;                    /* Handle du fichier */
| regs.x.cx = Fichier->RecS;                      /* Nombre d'octets */
| sregs.ds  = FP_SEG( Donnees );
| intdosx( &regs, &regs, &sregs );
| if ( !regs.x.cflag )
|  NetError = NE_OK;                                      /* Pas d'erreur */
| else
|  NetError = regs.x.ax;                  /* Mémorise le code d'erreur */
|}
|
|/*****************************************************************/
|/* NetRead : Lit des données dans un fichier                     */
|/* Entrées : cf infra                                            */
|/* Sortie  : néant                                               */
|/*****************************************************************/
|
|void NetRead( NFILE     *Fichier,                      /* Fichier */
|          void  far *Donnees )             /* Pointeur sur données */
|{
| regs.x.dx = FP_OFF( Donnees );      /* Adresse de la zone des données */
| regs.h.ah = FCT_READ;         /* Numéro de la fonction "Lire fichier "*/
| regs.x.bx = Fichier->Handle;                    /* Handle de fichier */
| regs.x.cx = Fichier->RecS;                      /* Nombre d'octets */
| sregs.ds  = FP_SEG( Donnees );
| intdosx( &regs, &regs, &sregs );
| if ( !regs.x.cflag )
|  NetError = NE_OK;                                      /* Pas d'erreur */
| else
|  NetError = regs.x.ax;                  /* Mémorise le code d'erreur */
|}
|
|/*****************************************************************/
|/* NetSeek : Positionne le pointeur du fichier x                 */
|/* Entrées : cf infra                                            */
|/* Sortie  : néant                                               */
|/*****************************************************************/
|
|void NetSeek( NFILE        *Fichier,                    /* Fichier */
|          unsigned long RecNo )        /* Numéro d'enregistrement */
|{
| regs.h.ah = FCT_LSEEK;/* Numéro de la fonction "Positionner pointeur"*/
| regs.h.al = 0;                  /* Position absolue à partir du début */
| regs.x.bx = Fichier->Handle;                    /* Handle du fichier */
| RecNo = RecNo * Fichier->RecS;                  /* Offset en octets */
| regs.x.cx = RecNo >> 16;                        /* Mot fort de l'offset */
| regs.x.dx = RecNo & 0xFFFF;                     /* Mot faible de l'offset */
| intdos( &regs, &regs );
| if ( !regs.x.cflag )
|  NetError = NE_OK;                                      /* Pas d'erreur */
| else
|  NetError = regs.x.ax;                  /* Mémorise le code d'erreur */
|}
|
```

Les programmes FLOCKP.PAS et FLOCKC.PAS sont destinés à démontrer l'usage du verrouillage des fichiers. Ils demandent d'abord à l'utilisateur de définir le type d'accès et le mode de partage de deux fichiers qui seront ouverts par la suite. Il s'agit en fait d'un seul et même fichier appelé selon le cas FLOCKC.DAT ou FLOCKP.DAT. Créez ces fichiers en recopiant n'importe quel fichier texte, par exemple AUTOEXEC.BAT. Ce fichier est ouvert par deux appels OPEN pour être associé à deux handles différents, ce qui permet de simuler un environnement de réseau où deux programmes accèdent simultanément au même fichier.

```
FLOCKP Démo de verrouillage de fichiers sous DOS    (c) 1992 by Michael Tischer
================================================================================

Types d'accès possibles:          Modes de partage possibles:
 1: Lecture seule                  1: Mode de compatibilité ( pas de prot)
 2: Ecriture seule                 2: Tout accès étranger interdit
 3: Lecture et écriture            3: Lecture seule
                                   4: Ecriture seule
                                   5: Tout est permis (Record lock)

Type d'accès pour le fichier de test A:
```

Image d'écran du programme FLOCKP.PAS

Dès l'ouverture des deux handles, des messages d'erreur DOS s'affichent. Les handles servent aussi à entreprendre des opérations d'écriture et de lecture dont le statut est affiché. Le programme écrit "AAAA" dans le premier handle et "BBBB" dans le deuxième.

Puis le contenu du fichier est lu par les deux handles et affiché sur l'écran. Si l'accès en écriture a pu être mené à bien par le deuxième handle, les deux handles provoquent l'affichage du texte "BBBB", autrement dit le deuxième enregistrement a écrasé le premier, pour autant que celui-ci ait pu être effectué. Lorsque les deux écritures ont échoué, le fichier est vide et l'affichage donne deux chaînes vides. La dernière étape consiste à refermer à nouveau les handles.

L'intérêt des ces programmes est de permettre le test de toutes les combinaisons possibles de type d'accès et de mode de partage. Elles ne sont pas toutes raisonnables et certaines d'entre elles donnent des résultats curieux mais il ne se passera rien d'autre en cas d'accès simultané à un même fichier dans un réseau si l'on s'en tient aux règles du jeu.

Listing : FLOCKP.PAS

```
{*********************************************************************}
{*                        F L O C K P . P A S                        *}
{*-------------------------------------------------------------------*}
{*   Fonction       : Montre comment fonctionne le verrouillage des  *}
{*                    fichiers sous réseau à l'aide des fonctions et *}
{*                    procédures de l'unité NETFILEP                 *}
{*-------------------------------------------------------------------*}
{*   Auteur         : Michael Tischer                                *}
{*   Développé le   : 10.02.1992                                     *}
{*   Dernière MAJ   : 13.02.1992                                     *}
{*********************************************************************}

program FLockP;

uses Crt, Dos,                          { Intègre les unités CRT et DOS }
     NetFileP;                            { et aussi l'unité de Réseau }

const NomFichier = 'flockp.dat';              { Nom du fichier de test }

type  Test  = array[ 1..4 ] of char;{Type de données pour fichier test }

{*********************************************************************}
{* FMode   : Génère le mode du fichier à partir du type d'accès et du *}
{*           mode de partage                                         *}
{* Entrées : Type d'accès et mode de partage du fichier              *}
{* Sortie  : Mode du fichier                                         *}
{*********************************************************************}

function FMode( Acces, Prot : byte ) : byte;

{-- Types d'accès et modes de partage définis comme dans Netfile ------}

const Acces_Type : array[ 1..3 ] of byte = ( FM_R, FM_W, FM_RW );
      Prot_Type  : array[ 1..5 ] of byte = ( SM_COMP, SM_RW, SM_R,
                                             SM_W, SM_NO );

begin
  FMode := Acces_Type[ Acces ] or Prot_Type[ Prot ];
end;

{*********************************************************************}
{* FichierTest : Montre les conflits d'accès ou le jeu de protections *}
{*               avec ou sans verrouillage                           *}
{* Entrées     : Type d'accès et mode de partage des deux fichiers   *}
{*               concurrents                                         *}
{* Sortie      : néant                                               *}
{*********************************************************************}

procedure FichierTest( AccesA, ProtA, AccesB, ProtB : byte );
```

```
const TestAEcr : Test = 'AAAA';                    { Enregistrements de test }
      TestBEcr : Test = 'BBBB';

var TestALec,                                      { Données de lecture }
    TestBLec  : Test;
    FichierA,                                { Fichiers de test pour accès commun }
    FichierB    : file of Test;

begin
  window( 1, 11, 80, 25 );
  clrscr;
  writeln( 'Fichier A: Nom = ', NomFichier, ', Type d''accès = ',
       AccesA, ',  Partage    = ', ProtA );
  writeln( 'Fichier B: Nom = ', NomFichier, ', Type d''accès = ',
       AccesB, ',  Partage    = ', ProtB );

  {-- Ouvre les fichiers ------------------------------------------------}

  write( #13#10'Ouverture du fichier A:  ' );
  NetReset( NomFichier, FMode( AccesA, ProtA ),
        sizeof( Test ), FichierA );
  if ( NetError = NE_FileNotFound ) then
    NetRewrite( NomFichier, FMode( AccesA, ProtA ),
        sizeof( test ), FichierA );
  Writeln( 'Etat   ', NetError : 2, ' = ', NetErrorMsg( NetError ) );

  write( 'Ouverture du fichier B:  ' );
  NetReset( NomFichier, FMode( AccesB, ProtB ),
        sizeof( Test ), FichierB );
  Writeln( 'Etat   ', NetError : 2, ' = ', NetErrorMsg( NetError ) );

  {-- Ecrit dans les fichiers -------------------------------------------}

  write( #13#10'Ecriture dans fichier A:' );
  if ( Is_NetWriteOk( FichierA ) ) then          { Ecriture permise ? }
    begin
      Netwrite( FichierA, TestAEcr );
      writeln( ' Données "', TestAEcr, '" enregistrées ' );
    end
  else
    writeln( ' Fichier interdit en écriture     ' );

  write( 'Ecriture dans fichier B:' );
  if ( Is_NetWriteOk( FichierB ) ) then          { Ecriture permise ?}
    begin
      Netwrite( FichierB, TestBEcr );
      writeln( ' Données "', TestBEcr, '" enregistrées ' );
    end
  else
    writeln( ' Fichier interdit en écriture ' );

      {-- Remet les pointeurs des deux fichiers au début ------------}

  if Is_NetOpen( FichierA ) then                 { Fichier ouvert? }
    NetSeek( FichierA, 0 );
  if Is_NetOpen( FichierB ) then                 { Fichier ouvert ? }
    NetSeek( FichierB, 0 );

  {-- Lit les fichiers  -------------------------------------------------}

  write( #13#10'Lecture dans fichier A:' );
  if ( Is_NetReadOk( FichierA ) ) then           { Lecture permise ? }
    begin
      Netread( FichierA, TestALec );
      writeln( ' Enregistrement "', TestALec, '" lu ' );
    end
  else
    writeln( ' Fichier interdit en lecture ' );

  write( 'Lecture dans fichier B:' );            { Lecture permise ? }
  if ( Is_NetReadOk( FichierB ) ) then
    begin
      Netread( FichierB, TestBLec );
      writeln( ' Enregistrement "', TestBLec, '" lu ' );
    end
  else
    writeln( ' Fichier interdit en lecture ' );

  {-- Fermeture des fichiers --------------------------------------------}

  NetClose( FichierA );
  NetClose( FichierB );
end;

{*********************************************************************}
{*                      PROGRAMME PRINCIPAL                          *}
{*********************************************************************}

var AccesA,                                    { Types d'accès des fichiers }
    AccesB,
    ProtA,                                     { Mode de partage      }
    ProtB   : byte;

begin
  clrscr;
  writeln( 'FLOCKP Démo de verrouillage de fichiers sous DOS    ',
       '(c) 1992 by Michael Tischer' );
  writeln( '                                                    ',
       '=========================' );

  if ( ShareInst ) then                        { Share installé? }
    begin
      {-- Sélectionne le mode ---------------------------------------}

      writeln( #13#10'Types d''accès possibles:         ',
            'Modes de partage possibles:' );
      writeln( ' 1: Lecture seule                ',
           ' 1: Mode de compatibilité ( pas de prot)' );
      writeln( ' 2: Ecriture seule               ',
           ' 2: Tout accès étranger interdit ' );
      writeln( ' 3: Lecture et écriture          ',
           ' 3: Lecture seule          ' );
      writeln( '                                 ',
           ' 4: Ecriture seule         ' );
      writeln( '                                 ',
           ' 5: Tout est permis (Record lock)' );

      Write( #13#10'Type d''accès pour le fichier de test A: ' );
      read( AccesA );
      Write( 'Mode de partage pour le fichier de test A: ' );
      read( ProtA );
      Write( 'Type d''accès pour le fichier de test B: ' );
      read( AccesB );
      Write( 'Mode de partage pour le fichier de test B: ' );
      read( ProtB );

      FichierTest( AccesA, ProtA, AccesB, ProtB );
    end
  else
    writeln( #13#10'Test impossible, SHARE doit être installé ' );
end.
```

Listing : FLOCKC.C

```
/*********************************************************************/
/*                          F L O C K C . C                          */
/*-------------------------------------------------------------------*/
/*   Fonction       : Montre comment fonctionne le verrouillage des  */
/*                    fichiers sous réseau avec le module NETFILEC   */
/*-------------------------------------------------------------------*/
/*   Modèle mémoire : SMALL                                          */
/*-------------------------------------------------------------------*/
/*   Auteur         : Michael Tischer                                */
/*   Développé le   : 10.02.1992                                     */
/*   Dernière MAJ   : 13.02.1992                                     */
/*********************************************************************/

#include <conio.h>
#include <stdio.h>
#incluce "netfilec.c"        /* Inclut les routines de gestion réseau */

/*-- Constantes ---------------------------------------------------------*/

#define NOMFICHIER "flockc.dat"          /* Nom du fichier de test */

/*-- Définitions de types -----------------------------------------------*/

typedef char Test[ 5 ];                  /* Données d'enregistrement */
typedef unsigned char BYTE;

/*-- Routines d'écran pour Microsoft C ----------------------------------*/

#ifndef __TURBOC__                               /* Microsoft C? */
```

```
#define clrscr() clearwindow( 1, 1, 80, 25 )

/*****************************************************************/
/* Gotoxy       : Positionne le curseur                          */
/* Entrée       : Coordonnées du curseur                         */
/* Sortie       : néant                                          */
/*****************************************************************/

void gotoxy( int x, int y )
{
  regs.h.ah = 0x02;       /* Numéro de la fonction de l'interruption */
  regs.h.bh = 0;                                    /* Page d'écran */
  regs.h.dh = y - 1;
  regs.h.dl = x - 1;
  int86( 0x10, &regs, &regs );                      /* Interruption */
}
#endif

/*****************************************************************/
/* clearwindow  : Efface une partie de l'écran                   */
/* Entrées      : cf infra                                       */
/* Sortie       : néant                                          */
/*****************************************************************/

void clearwindow( int x1, int y1, int x2, int y2 )
{
 regs.h.ah = 0x07;        /* Numéro de la fonction de l'interruption */
 regs.h.al = 0x00;
 regs.h.bh = 0x07;
 regs.h.ch = y1 - 1;
 regs.h.cl = x1 - 1;
 regs.h.dh = y2 - 1;
 regs.h.dl = x2 - 1;
 int86( 0x10, &regs, &regs );                       /* Interruption*/
 gotoxy( x1, y1 );                          /* Positionne le curseur */
}

/*****************************************************************/
/* FMode   : Génère le mode du fichier à partir du type d'accès et du */
/*           mode de partage                                     */
/* Entrées : cf infra                                            */
/* Sortie  : Mode du fichier                                     */
/*****************************************************************/

int FMode( int Acces,                                /* Type d'accès */
           int Prot )                /* Mode de partage ou de protection */
{
 static BYTE Acces_Type[ 3 ] = { FM_R, FM_W, FM_RW };
 static BYTE Prot_Type[ 5 ]  = { SM_COMP, SM_RW, SM_R,
                                 SM_W, SM_NO };

 return Acces_Type[ Acces-1 ] | Prot_Type[ Prot-1 ];
}

/*****************************************************************/
/* FichierTest : Montre les conflits d'accès et le jeu des protections*/
/*               avec ou sans verrouillage                       */
/* Entrées     : cf infra                                        */
/* Sortie      : néant                                           */
/*****************************************************************/

void FichierTest( int AccesA,              /* Type d'accès 1er fichier */
          int ProtA,        /* Mode de partage (protection) 1er fichier */
          int AccesB,                    /* Type d'accès 2me fichier */
          int ProtB )       /* Mode de partage (protection) 2me fichier */
{
 Test  TestAEcr = "AAAA\0";                        /* Enreg de test */
 Test  TestBEcr = "BBBB\0";

 Test  TestALec,                                 /* Pour lire des données */
       TestBLec;

 NFILE FichierA;                    /* Fichiers de test pour accès commun */
 NFILE FichierB;
 char  SDummy[ 50 ];                             /* Etat après exécution */

 clearwindow( 1, 11, 80, 25 );
 printf( "Fichier A: Nom = %s Type d'accès = %2i Mode de partage    =
%2i\n\n",
      NOMFICHIER, AccesA, ProtA );
 printf( "Fichier B: Nom = %s Type d'accès = %2i Mode de partage    =
%2i\n\n",
      NOMFICHIER, AccesB, ProtB );

/*-- Ouvre les fichiers ------------------------------------------*/

 printf( "Ouverture du fichier A:  " );
 NetReset( NOMFICHIER, FMode( AccesA, ProtA ),
       sizeof( Test ), &FichierA );
 if ( NetError == NE_FileNotFound )
  NetRewrite( NOMFICHIER, FMode( AccesA, ProtA ),
         sizeof( Test ), &FichierA );
 NetErrorMsg( NetError, SDummy );
 printf( "Etat %2u = %s\n\n", NetError, SDummy );

 printf( "Ouverture du fichier B:  " );
 NetReset( NOMFICHIER, FMode( AccesB, ProtB ),
       sizeof( Test ), &FichierB );
 NetErrorMsg( NetError, SDummy );
 printf( "Etat %2u = %s\n\n", NetError, SDummy );

/*-- Ecrit dans les fichiers -------------------------------------*/

 printf( "Ecriture dans fichier A:" );
 if ( Is_NetWriteOk( &FichierA ) )                /* Ecriture permise ? */
 {
  NetWrite( &FichierA, TestAEcr );
  printf( " Données '%s' enregistrées \n", TestAEcr );
 }
 else
  printf( " Fichier interdit à l'écriture \n" );

 printf( "Ecriture dans fichier B:" );
 if ( Is_NetWriteOk( &FichierB ) )                /* Ecriture permise ? */
 {
  NetWrite( &FichierB, TestBEcr );
  printf( " Données '%s' enregistrées\n\n", TestBEcr );
 }
 else
  printf( " Fichier interdit à l'écriture \n\n" );

/*-- Repositionne au début les deux pointeurs des fichiers -------*/

 if ( Is_NetOpen( &FichierA ) )                     /* Fichier ouvert? */
  NetSeek( &FichierA, 0L );
 if ( Is_NetOpen( &FichierB ) )                     /* Fichier ouvert? */
  NetSeek( &FichierB, 0L );

/*-- Lit les fichiers  -------------------------------------------*/

 TestALec[0] = TestBLec[0] = '\0';
 printf( "Lecture dans fichier A:" );
 if ( Is_NetReadOk( &FichierA ) )                 /* Lecture permise? */
 {
  NetRead( &FichierA, TestALec );
  printf( " Enregistrement '%s' lu\n", TestALec );
 }
 else
  printf( " Fichier interdit à la lecture \n" );

 printf( "Lecture dans fichier B:" );
 if ( Is_NetReadOk( &FichierB ) )                 /* Lecture permise? */
 {
  NetRead( &FichierB, TestBLec );
  printf( " Enregistrement '%s' lu\n\n", TestBLec );
 }
 else
  printf( " Fichier interdit à la lecture \n\n" );

/*-- Ferme les fichiers ------------------------------------------*/

 NetClose( &FichierA );
 NetClose( &FichierB );
}

/*****************************************************************/
/*                        PROGRAMME PRINCIPAL                    */
/*****************************************************************/

void main( void )
{
 int AccesA;                              /* Types d'accès des fichiers */
 int AccesB;
 int ProtA;                              /* Modes de partage (protection) */
 int ProtB;

 clrscr();
 gotoxy( 1, 1 );
 printf( "FLOCKC Démo de verrouillage de fichiers sous DOS   " \
     "(c) 1992 by Michael Tischer\n" );
 printf( "                                                  " \
     "                          \n\n" );

 if ( ShareInst() )                                 /* SHARE installé? */
 {
/*-- Sélectionne le mode  ----------------------------------------*/

  printf( "Types d'accès possibles          Modes de partage possibles\n"
 );
  printf( " 1: Lecture seule              " );
  printf( " 1: Mode de compatibilité (pas de protection)\n");
  printf( " 2: Ecriture seule             " );
  printf( " 2: Tout accès étranger interdit \n" );
  printf( " 3: Lecture et écriture        " );
  printf( " 3: Lecture seule\n" );
```

```
  printf( "                                  " );
  printf( " 4: Ecriture seule\n" );
  printf( "                                  " );
  printf( " 5: Tout est permis (record lock)\n" );

  printf( "\nType d'accès pour le fichier de test A: " );
  scanf( "%i", &AccesA );
  printf( "Mode de partage pour le fichier de test A: " );
  scanf( "%i", &ProtA );
  printf( "Type d'accès pour le fichier de test B: " );
  scanf( "%i", &AccesB );
  printf( "Mode de partage pour le fichier de test B: " );
  scanf( "%i", &ProtB );

  FichierTest( AccesA, ProtA, AccesB, ProtB );
 }
 else
  printf( "\nTest impossible, SHARE doit être chargé \n" );
}
```

Contrairement aux programmes FLOCK, les programmes RECLOCK (RECLOCKP.PAS et RECLOCKC.C) doivent être exécutés sur deux ordinateurs reliés par réseau ou sous Windows dans deux boîtes DOS séparées. La demonstration porte sur le verrouillage des enregistrements d'un fichier appelé REC.DAT pour les versions Pascal et C. Le fichier en question est créé au début du programme et comporte dix enregistrements de 160 octets chacun. On mémorise dans chacun de ces enregistrements un code ASCII unique, "A" majuscule pour le premier, "B" pour le second, "C" pour le troisième, etc...

```
Démo des fonctions de verrouillage de fichiers DOS  (C) 1991 by Michael Tischer
================================================================================

Fonctions possibles
  1: Positionner le pointeur de fichier.
  2: Verrouiller l'enregistrement
  3: Lire l'enregistrement
  4: Modifier l'enregistrement
  5: Ecrire dans l'enregistrement
  6: Lever l'interdiction
  7: Fin

Choix:


Enregistrement actuel:    0
Etat           : Libre
Statut réseau  :    0  = Pas d'erreur


Enregistrement actuel:
```

Image d'écran du programme RECLOCKP.PAS

Les deux programmes offrent la possibilité d'accéder aux différents enregistrements, de les lire, d'y écrire des données et surtout de les verrouiller individuellement. La partie droite de l'écran indique quels sont les enregistrements verrouillés dans chaque instance du programme.

Essayez de verrouiller quelques enregistrements dans la première instance du programme et d'y accéder dans la seconde. DOS va protester et retourner le code d'erreur 5 qui apparaît sur l'écran avec le message d'erreur associé.

Listing : RECLOCKP.PAS

```
{*********************************************************************}
{*                        R E C L O C K P                            *}
{*-------------------------------------------------------------------*}
{*   Fonction     : Démonstration des fonctions de verrouillage      *}
{*                  d'enregistrements DOS                            *}
{*-------------------------------------------------------------------*}
{*   Auteur       : Michael Tischer                                  *}
{*   Développé le : 19.09.1991                                       *}
{*   Dernière MAJ : 24.09.1991                                       *}
{*********************************************************************}

program RecLockP;

uses Crt, Dos,                          { Intégrer les unités CRT et DOS }
     NetFileP;                             { Intégrer l'unité de réseau }

const NomFichier = 'Rec.dat';          { Nom de fichier du fichier test }

type  Test     = array[ 1..160 ] of char;    { Type de données du test }
      TestFile = file of Test;

var Fichier : TestFile;                                { Fichier test }

{*********************************************************************}
{* CreerEnreg : Créer des enregistrements pour un test               *}
{* Entrée     : Caractères pour l'enregistrement                     *}
{* Sortie     : Données de test                                      *}
{*********************************************************************}

procedure CreerEnreg(     Caractere : char;
                      var Enreg     : test );

var i : word;                                      { Compteur de boucle }

begin
  for i := 1 to 160 do
    Enreg[ i ] := Caractere;
end;

{*********************************************************************}
{* OuvrirFichier : Ouvrir un fichier réseau existant. Sinon créer un *}
{*                 nouveau fichier test et compléter le fichier      *}
{*                 avec des données de test                          *}
{* Entrée        : Fichier                                           *}
{* Sortie        : Fichier                                           *}
{*********************************************************************}

function OuvrirFichier( var Fichier : testfile ) : boolean;

var i        : word;                               { Compteur de boucle }
    TestChaine : Test;{ Nécessaire pour la création de données de test }

begin
{-- Ouvrir un fichier pour l'entrée et la sortie en mode Deny-None ----}

  NetReset( NomFichier, fm_rw or sm_no, sizeof( Test ), Fichier );
  if ( NetError = NE_FileNotFound ) then       { Fichier inexistant? }
    begin

{-- Créer un fichier et compléter par des données de test ------------}

      NetRewrite( NomFichier, fm_rw or sm_no, sizeof( Test ), Fichier );
      if ( NetError = 0 ) then                   { Créer sans erreur? }
        begin
          if NetLock( Fichier, 0, 26 ) then    { Verrouiller 26 enreg. }
            begin
              NetSeek( Fichier, 0 );{ Pointeur sur le début du fichier }
              for i := 1 to 26 do
                begin
                  CreerEnreg( chr( ord( 'Z' ) + 1 - i ), TestChaine );
                  NetWrite( Fichier, TestChaine );{Ecrire données test }
                end;
              OuvrirFichier := NetUnlock( Fichier, 0, 26 );
            end
          else
            OuvrirFichier := false;      { Erreur lors du verrouillage }
        end
      else
        OuvrirFichier := false;{ Erreur lors de la création du fichier }
    end
  else
    OuvrirFichier := ( NetError = 0 );           { Ouvrir ou erreur? }
end;

{*********************************************************************}
{* NetzEdit   : Démonstration des fonctions de réseau                *}
{* Entrée     : Fichier                                              *}
{* Sortie     : Fichier                                              *}
{*********************************************************************}

procedure NetzEdit( var Fichier : TestFile );

var NumEnreg : longint;              { Numéro d'enregistrement actuel }
    EnregAct : Test;                          { Enregistrement actuel }
    Action   : byte;                            { Action souhaitée }
    Etat     : boolean;                 { Enregistrement verrouillé? }
    Caractere : char;

begin
{-- Afficher le menu -------------------------------------------------}

  writeln( #13#10'Fonctions possibles' );
  writeln( '  1: Positionner le pointeur de fichier.' );
  writeln( '  2: Verrouiller l''enregistrement' );
  writeln( '  3: Lire l''enregistrement' );
  writeln( '  4: Modifier l''enregistrement' );
  writeln( '  5: Ecrire dans l''enregistrement' );
  writeln( '  6: Lever l''interdiction' );
  writeln( '  7: Fin' );

  NumEnreg := 0;                          { Enregistrement actuel }
  Etat := false;                     { Enregistrement non verrouillé }
  CreerEnreg( #32, EnregAct );        { Créer un enregistrement vide }

  repeat
  {-- Afficher les informations ---------------------------------------}

    gotoxy( 1, 16 );     { Afficher la position du pointeur de fichier }
    writeln( 'Enregistrement actuel: ', NumEnreg : 4 );
    write( 'Etat              : ' );
    if Etat then
      writeln( 'Verrouillé' )
    else
      writeln( 'Libre    ' );
    Writeln( 'Statut réseau  : ', NetError: 4, ' = ',
       copy( NetErrorMsg( NetError ) + '                    ', 1, 30 ) );
    gotoxy( 1, 21 );              { Afficher l'enregistrement de test }
    writeln( 'Enregistrement actuel:' );
    writeln( EnregAct );

    NetSeek( Fichier, NumEnreg ); { Positionner le pointeur de fichier }

    gotoxy( 1, 13 );
    write( 'Choix:                              ' );
    gotoxy( 10, 13 );
    readln( Action );
    case Action of
      1 : begin
            gotoxy( 1, 13 );
            write( 'Nouveau numéro d''enregistrement: ' );
            readln( NumEnreg );
            Etat := false;          { Enregistrement non verrouillé }
            CreerEnreg( #32, EnregAct )
          end;
      2 : Etat := Etat or NetLock( Fichier, NumEnreg, 1 );
      3 : NetRead( Fichier, EnregAct );          { Lire l'enregistrement }
      4 : begin
            gotoxy( 1, 13 );
            write( 'Nouveau caractère: ' );
            readln( Caractere );
            CreerEnreg( Caractere, EnregAct );
          end;
      5 : NetWrite( Fichier, EnregAct );{ Ecrire dans l'enregistrement }
      6 : Etat := Etat and not NetUnlock( Fichier, NumEnreg, 1 );
    end;
  until ( Action = 7 );
end;

{*********************************************************************}
{*              P R O G R A M M E  P R I N C I P A L                 *}
{*********************************************************************}

begin
  clrscr;
  writeln( 'Démo des fonctions de verrouillage de fichiers DOS',
           ' (C) 1991 by Michael Tischer' + paramstr( 1 ) );
  writeln( '-------------------------------------',
           '-------------' );

  if ( ShareInst ) then                  { Programme Share installé? }
    begin
      if OuvrirFichier( Fichier ) then       { Fichier ouvert ou créé? }
        begin
          NetzEdit( Fichier ); { Démonstration des fonctions de réseau }
          NetClose( Fichier );                      { Fermer le fichier }
        end
```

```
    else
      writeln( #13#10'Erreur lors de l''ouverture du fichier réseau, ' +
               'Numéro d''erreur: ', NetError );
    ClrScr;
  end
  else
    writeln( #13#10'Test non possible, Share doit être installé' );
end.
```

Listing : RECLOCKC.C

```
/*****************************************************************/
/*                      R E C L O C K C . C                      */
/*---------------------------------------------------------------*/
/*   Fonction       : Montre comment fonctionne le verrouillage des */
/*                    enregistrements avec les fonctions du module  */
/*                    NETFILEC                                      */
/*---------------------------------------------------------------*/
/*   Modèle mémoire : SMALL                                       */
/*---------------------------------------------------------------*/
/*   Auteur         : Michael Tischer                             */
/*   Développé le   : 10.02.1992                                  */
/*   Dernière MAJ   : 13.02.1991                                  */
/*****************************************************************/

#include "netfilec.c"        /* Intègre les routines orientées réseau */
#include <stdio.h>
#include <conio.h>

/*-- Constantes ---------------------------------------------------*/

#define NOMFICHIER "rec.dat"                 /* Nom du fichier de test */
#define ENREGIS    10                        /* Nombre d'enrg . de test */

/*-- Définitions de types -----------------------------------------*/

typedef char Test[ 160 ];              /*Type de données pour le test */
typedef char TestString[ 161 ];    /* Type des données pour affichage */

#ifndef __TURBOC__                                /* Microsoft C? */

  #define clrscr() clearwindow( 1, 1, 80, 25 )

  /*****************************************************************/
  /* Gotoxy       : Positionne le curseur                          */
  /* Entrées      : Coordonnées du curseur                         */
  /* Sortie       : néant                                          */
  /*****************************************************************/

  void gotoxy( int x, int y )

  {
    regs.h.ah = 0x02;          /* Numéro de la fonction d'interruption */
    regs.h.bh = 0;                                      /* Couleur */
    regs.h.dh = y - 1;
    regs.h.dl = x - 1;
    int86( 0x10, &regs, &regs );                  /* Interruption   */
  }
#endif

/*****************************************************************/
/* clearwindow  : Efface une partie de l'écran                   */
/* Entrées      : cf infra                                       */
/* Sortie       : néant                                          */
/*****************************************************************/

void clearwindow( int x1, int y1, int x2, int y2 )
{
 regs.h.ah = 0x07;             /* Numéro de la fonction d'interruption*/
 regs.h.al = 0x00;
 regs.h.bh = 0x07;
 regs.h.ch = y1 - 1;
 regs.h.cl = x1 - 1;
 regs.h.dh = y2 - 1;
 regs.h.dl = x2 - 1;
 int86( 0x10, &regs, &regs );                       /* Interruption*/
 gotoxy( x1, y1 );                            /* Refixe le curseur */
}

/*****************************************************************/
/* OuvreFichier : Ouvre un fichier préexistant. Sinon créer un nouveau*/
/*                fichier de test et le remplit avec les données  */
/*                prévues à cet effet                             */
/* Entrées      : cf infra                                       */
/* Sortie       : Fichier                                        */
/*****************************************************************/

int OuvreFichier( NFILE *Fichier )              /* Fichier réseau */
{
 int i;                                                 /* Compteur*/
 Test TestEnreg;      /* Indispensable pour créer les données de test */

 /*-- Ouvre un fichier pour entrée/sortie en mode Deny-None ---------*/

 NetReset( NOMFICHIER, FM_RW | SM_NO, sizeof( Test ), Fichier );
 if ( NetError == NE_FileNotFound )             /* Fichier inconnu ? */
 {
                 /*-- Crée le fichier et le remplit ----------------*/

  NetRewrite( NOMFICHIER, FM_RW | SM_NO, sizeof( Test ), Fichier );
  if ( NetError == NE_OK )                      /* Création correcte ? */
  {
   if ( NetLock( Fichier, 0L, (long) ENREGIS ))/* Verrouille tous les */
                                                /* enregistrement */
   {
    NetSeek( Fichier, 0L );         /* Pointe sur le début du fichier */
    for ( i = 0; i < ENREGIS; i++ )
    {
     memset( TestEnreg, 'A' + i, 160 );
     NetWrite( Fichier, TestEnreg );     /* Ecrit les données de test */
    }
    return NetUnLock( Fichier, 0L, (long) ENREGIS );
   }
   else
    return FALSE;                              /* Erreur de verrouillage */
  }
  else
   return FALSE;                                /* Erreur de création */
 }
 else
  return ( NetError == 0 );                    /* Ouverture sans erreur ? */
}

/*****************************************************************/
/* ResoEdit   : Démonstration des fonctions de réseau            */
/* Entrées    : voir plus loin                                   */
/* Sortie     : Fichier                                          */
/*****************************************************************/

void ResoEdit( NFILE *TestFile )                /* Fichier réseau */
{
 unsigned long ActRecord;                  /* Numéro d'enreg. courant */
 TestString    ActEnreg;                      /* Données courantes */
 int           Action;                         /* Action souhaitée */
 int           Status;                       /* Enreg. verrouillé ? */
 char          Caractere[ 10 ];
 char          SDummy[ 50 ];                    /* Etat du réseau */
 int           Verrouille[ ENREGIS ];
 int           i;                                       /* Compteur */

 /*-- Affiche le menu ----------------------------------------------*/

 printf( "Fonctions proposées\n" );
 printf( "  1: Positionne le pointeur\n" );
 printf( "  2: Verrouille un enregistrement\n" );
 printf( "  3: Lit un enregistrement\n" );
 printf( "  4: Modifie les données \n" );
 printf( "  5: Ecrit un enregistrement\n" );
 printf( "  6: Déverrouille un enregistrement\n" );
 printf( "  7: Fin\n" );

 /*-- Insitialise et affiche l'état des enregistrements ------------*/

 gotoxy( 58, 4 );
 printf( "Etat des enreg:" );
 for ( i = 0; i < ENREGIS; ++i )
 {
  Verrouille[i] = FALSE;
  gotoxy( 64, i+5 );
  printf( "%2d   libre", i );
 }

 ActRecord = 0;                              /* Enregistrement courant */
 Status = FALSE;                                  /* Non verrouillé */
 memset( ActEnreg, 32, 160 );               /* Génère des données vides */
```

```
do
{
/*-- Affiche les informations ----------------------------------------*/

 gotoxy( 1, 16 );                          /* Position du pointeur */
 printf( "Enregistrement courant: %4ld\n",  ActRecord );
 printf( "Etat            :    %s\n",
      Verrouille[ActRecord] ? "verrouillé" : "libre      " );
 NetErrorMsg( NetError, SDummy );
 printf( "Etat du réseau : %4d = %s", NetError, SDummy );
 gotoxy( 1, 21 );                          /* Affiche les données */
 printf( "Données courantes :\n" );
 ActEnreg[ 160 ] = 0;
 printf( "%s", ActEnreg );

 NetSeek( TestFile, ActRecord );           /* Positionne le pointeur */
 gotoxy( 1, 13 );
 printf( "Sélection:                    " );
 gotoxy( 12, 13 );
 Action = 0;
 scanf( "%i", &Action );
 switch( Action )
 {
  case 1 : gotoxy( 1, 13 );
       printf( "Nouveau numéro: " );
           do
            {
             gotoxy( 22, 13 );
             printf( "                  " );
             gotoxy( 22, 13 );
             scanf( "%li", &ActRecord );
            }
           while (( ActRecord >= 0  &&  ActRecord < ENREGIS ));
           break;

  case 2 : Status = NetLock( TestFile, ActRecord, 1L );
           if ( Status )
            {
             Verrouille[ ActRecord ] = TRUE;
             gotoxy( 64, (int) ActRecord +5 );
         printf( "%2d   verrouillé", ActRecord );
            }
           break;

  case 3 : NetRead( TestFile, ActEnreg );        /* Lit les données */
           break;

  case 4 : gotoxy( 1, 13 );
       printf( "Nouveau caractère:" );
           scanf( "%s", Caractere );
           memset( ActEnreg, Caractere[ 0 ], 160 );
           break;

  case 5 : NetWrite( TestFile, ActEnreg ); /* Enregistre les données */
           break;

  case 6 : Status = NetUnLock( TestFile, ActRecord, 1L );
           if ( Status )
            {
             Verrouille[ ActRecord ] = FALSE;
             gotoxy( 64, (int) ActRecord+5 );
             printf( "%2d   libre      ", ActRecord);
            }
           break;
 }

}
while ( Action != 7 );
}

/*********************************************************************/
/*                          Programme principal                      */
/*********************************************************************/

void main( )

{
 NFILE Fichier;                                /* Fichier de test */

 clrscr();
 printf( "RECLOCKC Démo du verrouillage d'enregistrements DOS" \
         " (c) 1992 by Michael Tischer\n" );
 printf( "-------------------------------------------------------" \
         "-----------\n\n" );

 if ( ShareInst() )                            /* Share installé? */
 {
  if ( OuvreFichier( &Fichier ) )        /* Fichier ouvert ou créé ? */
  {
   ResoEdit( &Fichier );                 /* C'est parti pour la démo */
   NetClose( &Fichier );                        /* Ferme le fichier */
   clrscr( );
  }
  else
   printf( "\nErreur %i à l'ouverture du fichier" , NetError );
 }
 else
  printf( "\nTest impossible, SHARE doit être installé" );
}
```

29. DOS et Windows

L'utilisateur qui crée des programmes DOS à l'heure actuelle doit toujours envisager de les faire tourner également sous Windows. Bien que Windows simule parfaitement l'environnement normal du DOS, il arrive toutefois que toutes les routines des programmes DOS ne peuvent pas être exécutées sous Windows. Il faut notamment faire attention aux utilitaires tels que Speed Disk ou Calibrate de Norton qui risquent de se planter sous Windows. Des problèmes surviennent également avec les programmes TSR qui accèdent essentiellement au noyau de DOS pour se protéger contre d'autres programmes TSR.

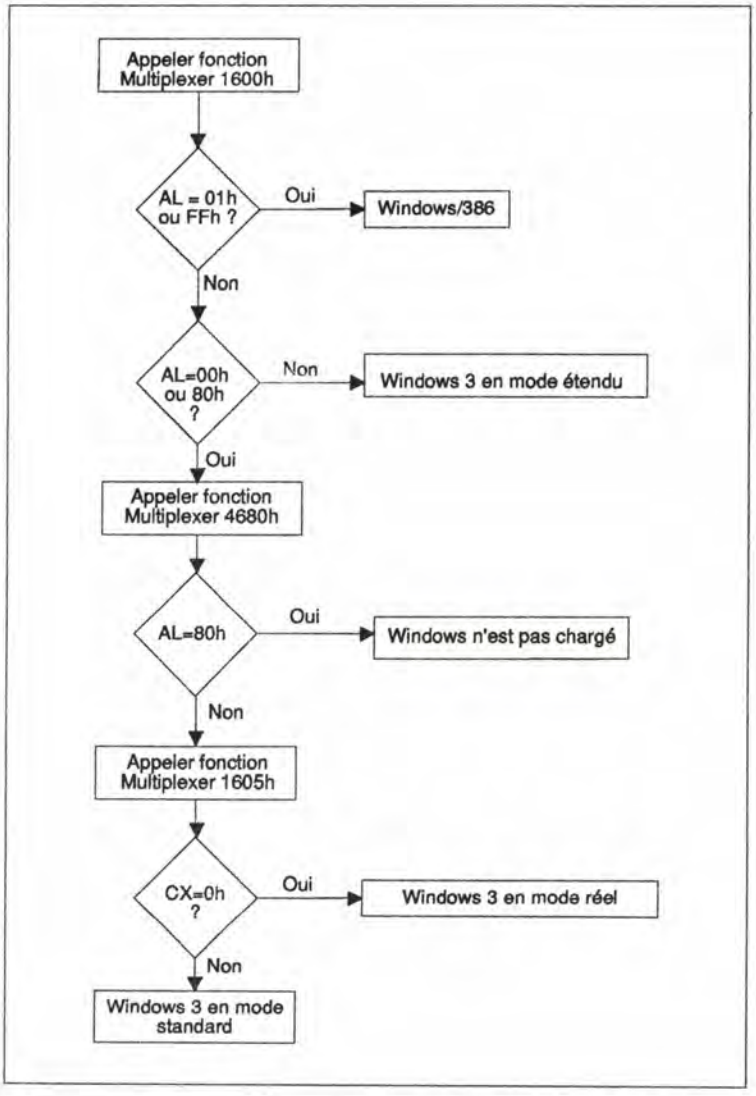
Ce n'est pas sans raison que les guides d'utilisation informent le programmeur que certains utilitaires disque et programmes TSR ne doivent pas être exploités sous Windows. Mais des utilisateurs veulent néanmoins élucider ce mystère.

Des programmes considérés potentiellement comme dangereux commencent alors par une sorte de test destiné à signaler la présence de Windows. Dans ce cas, le programme s'interrompt après avoir affiché un message d'erreur à l'écran. Ce chapitre montre comment réaliser un test de ce genre.

29.1. Découvrir Windows

Avant tout, notez qu'il n'existe aucune fonction DOS capable de communiquer si Windows est activé et indiquer sa version et son mode. Mais au contraire, il faut associer plusieurs appels de fonctions qui n'ont aucune relation apparente entre elles pour obtenir progressivement les informations souhaitées.

Comme le montre la figure suivante illustrant le déroulement du test Windows, l'interruption Multiplexer joue un rôle important dans la détermination de la présence de Windows. Il s'agit de l'interruption 2Fh contenant de nombreux programmes DOS et utilitaires (SHARE et PRINT) résidents offrant leurs services à d'autres programmes. Windows insère également des fonctions dans cette interruption.



Algorithme de contrôle de la version Windows et du mode d'exploitation

D'après la figure précédente, le Multiplexer appelle d'abord la fonction 1600h. Windows la tient prête pour déterminer le mode d'exploitation de Windows 3.x. Si l'interruption Multiplexer ne contient aucune fonction portant ce numéro, le contenu du registre n'est pas alors modifié par l'appel de la fonction et le registre AL contient la valeur 0. Ainsi, on sait d'ores et déjà que Windows 3.x n'est pas activé. Les tests suivants révèlent si une autre version de Windows est active.

Après l'appel de la fonction, si on rencontre la valeur 01h ou FFh dans le registre AL, cela confirme l'absence de Windows 3.x mais la présence de la version spéciale installée

auparavant, c'est-à-dire la version 386 de Windows 2.x. On ne peut certes pas spécifier avec précision de quelle version il s'agit, mais ce qui est sûr c'est que Windows est activé.

Après cet appel, une autre valeur permet de répondre à la question : 80h. Si cette valeur se trouve dans le registre AL, cela signifie que Windows 3 est bel et bien activé mais pas en mode étendu. Tout comme avec la valeur 00h, d'autres tests s'avèrent nécessaires.

Cependant, il convient d'examiner d'abord ces tests au cas où ils pourraient apporter une réponse à la question. Si le registre AL contient une valeur différente de 00h, 01h, 80h ou FFh, cela confirme la présence de Windows 3 (3.1 ou peut-être 4) en mode étendu. La valeur du registre AL représente le numéro de version principal de la version Windows qui est en train de tourner. Quant au numéro secondaire, il se trouve dans le registre AH.

Revenons au test à effectuer en cas de réception de la valeur 00h ou 80h. Pour spécifier si Windows 3 est réellement activé, on peut se servir de la fonction 4680h. Elle introduit Windows 3 dans l'interruption Multiplexer. Si elle retourne la valeur 80h dans le registre AL, cela indique que cette fonction n'est pas du tout implantée. On en déduit que Windows n'est pas non plus activé.

Mais si on rencontre la valeur 00h, cela signifie que Windows est activé. Il ne reste plus qu'à déterminer le mode. En fin de compte, Windows 3 peut toujours fonctionner en mode standard ou réel une fois que le mode étendu a été exclu.

Une autre fonction ayant inséré Windows dans l'interruption Multiplexer fait son apparition. Il s'agit de la fonction 1605h. Elle sert à commuter Windows depuis le mode d'exploitation en cours vers le mode standard et subit un échec si le mode standard est déjà activé. On reconnaît cet état de fait par la valeur 0 retournée dans le registre CX.

Si la fonction peut toutefois être exécutée, Windows passe alors du mode réel au mode standard. Cette commutation peut être annulée aussitôt en appelant la fonction Multiplexer 1606h.

En tout cas, ce dernier test confirme si Windows 3 a été dernièrement exécuté en mode standard ou réel. Ainsi se termine le test.

Programmes d'exemples

Nous avons écrit trois programmes en Basic, Pascal et C qui se chargent de convertir à votre place l'algorithme décrit plus haut en code programme. Ils portent les noms WINDAB.BAS, WINDAP.PAS et WINDAC.C et sont construits de manière identique.

Le point fort de ces programmes est une routine portant le nom WINDOWS. Comme résultat de fonction, elle retourne l'une des constantes NO_WIN, WIN_386_X,

WIN_REAL, WIN_STANDARD ou WIN_ENHANCED déclarées au début des programmes. Grâce à leurs noms, elles reproduisent fidèlement la version Windows et le mode d'exploitation en présence.

La fonction Windows retourne également le numéro exact de la version Windows, du moins lorsque Windows 3 est exploité en mode étendu. A cet effet, il faut transmettre deux variables integer (ou leurs adresses dans la version C) à cette fonction. La fonction Windows fournit ensuite le numéro de version de Windows dans ces variables.

Listing : WINDAP.PAS

```
{*********************************************************************}
{*                        W I N D A P . P A S                        *}
{*-------------------------------------------------------------------*}
{*   Fonction       : Confirmer si Windows est actif et dans         *}
{*                    quel mode.                                     *}
{*-------------------------------------------------------------------*}
{*   Auteur         : Michael Tischer                                *}
{*   Développé le   : 22.08.1991                                     *}
{*   Dernière modif. : 22.03.1992                                    *}
{*********************************************************************}

uses Dos;                                          { Insérer l'unité DOS }

const MULTIPLEX  = $2F;                  { N° de l'interruption Multiplex }
      NO_WIN     = $00;                          { Windows non actif }
      W_386_X    = $01;                   { Windows /386 V2.X en cours }
      W_REAL     = $81;                 { Windows fonctionne en mode réel }
      W_STANDARD = $82;               { Windows fonctionne en mode standard }
      W_ENHANCED = $83;                { Windows fonctionne en mode étendu }

{*********************************************************************}
{* WINDOWS : Confirme si Windows est actif                           *}
{* Entrée : HVERSION = Variable Integer devant contenir le numéro    *}
{*                     de version principal                          *}
{*          NVERSION = Integer devant contenir le numéro de version  *}
{*                     secondaire                                    *}
{* Sortie : Statut Windows, une constante parmi NO_WIN, W_386_X,     *}
{*          W_REAL, W_STANDARD ou W_ENHANCED                         *}
{* Infos  : Le numéro de version ne peut être obtenu que dans le     *}
{*          mode étendu de Windows 3                                 *}
{*********************************************************************}

function windows( var HVersion, NVersion : integer ) : integer;

var regs : registers;          (* Registre pour l'appel d'interruption *)
    Res  : integer;

{-- Cette fonction remplace l'appel de intr( $2F, Regs ) -------------}
{-- Regs.ax = $1600 (Test d'installation du mode étendu), ------------}
{-- l'appel avec la fonction Pascal renvois des valeurs erronées   ---}

function int2fcall : integer;

begin
  inline( $b8 / $00 / $16 /                { mov   ax,1600h          *}
          $cd / $2f /                      { int   2Fh               *}
          $89 / $46 / $FE );               { mov   [bp-2], ax        *}
  { A cet endroit, le compilateur rajoute "mov ax, [bp-2]" pour      *}
  { charger la variable de fonction locale dans le registre de retour *}
end;

begin
  HVersion := 0;                          { Initialise le numéro de version }
  NVersion := 0;

            {-- Identifie Windows x.y en mode étendu ----------------}

  res := int2fcall;                  { Test d'installation du mode étendu }

  case ( lo(Res) ) of
    $01,
    $FF: begin
           HVersion := 2;                          { Version principale }
           NVersion := 0;                 { Version secondaire inconnue }
           Windows := W_386_X;
         end;
    $00,
    $80: begin
           regs.ax := $4680;  { Identifier les modes Réel et Standard }
           intr( MULTIPLEX, regs );
           if ( regs.al = $80 ) then
             Windows := NO_WIN                { Windows ne fonctionne pas }
           else
             begin
                  {-- Windows en mode Réel ou Standard ---------------}

               regs.ax := $1605;   { Simule l'inst. d'un DOS-Extender }
               regs.bx := $0000;
               regs.si := $0000;
               regs.cx := $0000;
               regs.es := $0000;
               regs.ds := $0000;
               regs.dx := $0001;
               intr( MULTIPLEX, regs );
               if ( regs.cx = $0000 ) then
                 begin
                    {-- Windows en mode Réel -------------------------}

                   regs.ax := $1606;
                   intr( MULTIPLEX, regs );
                   Windows := W_REAL;
                 end
               else
                 Windows := W_STANDARD;
             end;
         end;

    {-- Windows en mode Etendu, ax contient le numéro de version -------}

    else
      begin
        HVersion := lo(Res);           { Afficher la version de Windows }
        NVersion := hi(Res);
        Windows := W_ENHANCED;                  { Windows en mode Etendu }
      end;
  end;
end;

{*********************************************************************}
{*                  P R O G R A M M E   P R I N C I P A L            *}
{*********************************************************************}

var WindowsActif,                                      { Mode de Windows }
    HVer,                                { Version principale de Windows }
    NVer         : integer;              { Version secondaire de Windows }

begin
  writeln( ' WINDAP - (c) 1991, 92 by Michael Tischer ' );
  writeln;
  WindowsActif := windows( HVer, NVer );
  case ( WindowsActif ) of
    NO_WIN     : writeln( 'Windows non actif ' );
    W_REAL     : writeln( 'Windows en mode Réel' );
    W_STANDARD: writeln( 'Windows activé en mode Standard' );
    W_386_X    : writeln( 'Windows/386 V 2.x actif' );
    W_ENHANCED: writeln( 'Windows V ', HVer, '.', NVer,
                         ' actif en mode Etendu' );
  end;
  halt( WindowsActif );
end.
```

Listing : WINDAC.C

```
/*****************************************************************
*                          W I N D A C . C                       *
*----------------------------------------------------------------*
*   Fonction       : Confirme si Windows est actif et dans       *
*                    quel mode.                                  *
*----------------------------------------------------------------*
*   Auteur         : Michael Tischer                             *
*   Développé le   : 22.08.1991                                  *
*   Dernière modif.: 22.03.1992                                  *
*----------------------------------------------------------------*
*   Modèle de mémoire : SMALL                                    *
*****************************************************************/

/*--- Insérer les fichiers Include ---------------------------*/

#include <dos.h>
#include <stdio.h>

/*--- Constantes ---------------------------------------------*/

#define MULTIPLEX    0x2F                    /* N° d'int. Multiplex */

#define NO_WIN       0x00                    /* Windows non actif */
#define W_386_X      0x01                 /* Windows /386 V2.x en cours */
#define W_REAL       0x81                 /* Windows fonctionne en mode Réel*/
#define W_STANDARD   0x82              /* Windows fonctionne en mode Standard */
#define W_ENHANCED   0x83               /* Windows fonctionne en mode Etendu */

/*****************************************************************
* windows : Confirme si Windows est actif                        *
* Entrée : HVERSION  = Ptr sur INT devant contenir le numéro de  *
*                      version principale                        *
*          NVERSION = Ptr sur INT devant contenir le numéro de   *
*                     version secondaire                         *
* Sortie : Statut Windows, une constante parmi NO_WIN, W_386_X,  *
*          W_STANDARD, W_STANDARD ou W_ENHANCED                  *
* Infos  : Le numéro de version peut être obtenu qu'en mode Etendu *
*          de Windows 3                                          *
*****************************************************************/

int windows( int *HVersion, int *NVersion )
{
 union  REGS  regs;              /* Registres pour l'appel d'interruption */
 struct SREGS sregs;              /* Segment pour l'appel d'interruption */

 *HVersion = 0;                        /* Initialise le numéro de version */
 *NVersion = 0;

          /*-- Identifie Windows x.y en mode Etendu ----------------*/

 regs.x.ax = 0x1600;                  /* Test d'installation de Windows */
 segread( &sregs );                    /* Lire les registres de segment */
 int86x( MULTIPLEX, &regs, &regs, &sregs );

 switch ( regs.h.al )
 {
  case 0x01:
  case 0xFF:  *HVersion = 2;                          /* Hauptversion */
              *NVersion = 0;              /* Version secondaire inconnue */
              return W_386_X;               /* Windows /386 Version 2.x */

  case 0x00:
  case 0x80:  regs.x.ax = 0x4680;              /* Modes Réel et Standard */
              int86x( MULTIPLEX, &regs, &regs, &sregs );
              if( regs.h.al == 0x80 )
                return NO_WIN;                /* Windows ne fonctionne pas */
              else
              {
               /*-- Windows en mode Réel ou Standard ----------------*/

                regs.x.ax = 0x1605;           /* Simuler l'initialisation */
                regs.x.bx = regs.x.si = regs.x.cx =
                            sregs.es = sregs.ds = 0x0000;
                regs.x.dx = 0x0001;
                int86x( MULTIPLEX, &regs, &regs, &sregs );
                if( regs.x.cx == 0x0000 )
                {
                 /*-- Windows en mode Réel ----------------------------*/

                  regs.x.ax = 0x1606;
                  int86x( MULTIPLEX, &regs, &regs, &sregs );
                  return W_REAL;
                }
                else
                  return W_STANDARD;
              }

 /*-- Windows en mode Etendu, ax contient le numéro de version -------*/

  default:  *HVersion = regs.h.al;  /* Afficher la version de Windows */
            *NVersion = regs.h.ah;
            return W_ENHANCED;                /* Windows en mode Etendu */
 }
}

/*****************************************************************
*                P R O G R A M M E   P R I N C I P A L           *
*****************************************************************/

int main( void )
{
 int WindowsActif,                                 /* Mode de Windows */
     HVer,                              /* Version principale de Windows */
     NVer;                              /* Version secondaire de Windows */

 printf(" WINDAC  -  (c) 1991, 92 by Michael TISCHER \n\n" );
 WindowsActif = windows( &HVer, &NVer );
 switch ( WindowsActif )
 {
  case NO_WIN:     printf( "Windows non actif\n" );
                   break;
  case W_REAL:     printf( "Windows actif en mode Réel\n" );
                   break;
  case W_STANDARD: printf( "Windows actif en mode Standard\n" );
                   break;
  case W_386_X:    printf( "Windows/386 V 2.x actif" );
                   break;
  case W_ENHANCED: printf( "Windows V %d.%d actif en %s\n",
                           HVer, NVer, "mode Etendu" );
                   break;
 }
 return( WindowsActif );
}
```

Listing : WINDAB.BAS

```
'*****************************************************************
'*                          W I N D A B . B A S                  *
'*---------------------------------------------------------------*
'* Fonction       : Confirmer si Windows est actif et dans       *
'*                  quel mode                                    *
'*---------------------------------------------------------------*
'* Auteur         : Michael TISCHER                              *
'* Développé le   : 22.08.1991                                   *
'* Dernière MAJ.  : 22.03.1992                                   *
'*****************************************************************

DECLARE FUNCTION Windows% (HVersion AS INTEGER, NVersion AS INTEGER)

'$INCLUDE: 'Qb.bi'                    'Fichier Include pour appels d'int.

CONST MULTIPLEX = &H2F                'Numéro d'int. de l'int. Multiplex
CONST NOWIN = &H0                                    'Windows non actif
CONST WIN386X = &H1                          'Windows/386 V2.x fonctionne
CONST WINREAL = &H81                  'Windows fonctionne en mode réel
CONST WINSTANDARD = &H82            'Windows fonctionne en mode standard
CONST WINENHANCED = &H83             'Windows fonctionne en mode étendu

DIM WindowsActif AS INTEGER                                  'Mode Windows
DIM HVer AS INTEGER                        'Version principale de Windows
DIM NVer AS INTEGER                        'Version secondaire de Windows

PRINT " WINDAB  -  (c) 1991, 92 by Michael TISCHER "
PRINT                                              'Insérer une ligne vierge
WindowsActif = Windows(HVer, NVer)      'Déterminer mode/version Windows
SELECT CASE WindowsActif
```

```
!  CASE NOWIN
!     PRINT "Windows non actif"
!  CASE WIN2X
!     PRINT "Windows /386 V 2.x actif"
!  CASE WINREAL
!     PRINT "Windows en mode réel"
!  CASE WINSTANDARD
!     PRINT "Windows actif en mode standard "
!  CASE WINENHANCED
!     PRINT "Windows V "; LTRIM$(STR$(HVer)); ".";
!     PRINT LTRIM$(STR$(NVer)); " actif en mode Etendu"
!END SELECT
!
!'*********************************************************************
!'* Windows : Confirme si Windows est actif                           *
!'* Entrée  : HVERSION = Integer pour recevoir version principale     *
!'*           NVERSION = Integer pour recevoir version secondaire     *
!'* Sortie  : Statut Windows, constante parmi NOWIN, WIN386X, WINREAL,*
!'*           WINSTANDARD ou WINENAHNCED                              *
!'* Info    : Le numéro de version ne peut être lu et retourné que dans *
!'*           le mode étendu de Windows 3.0                           *
!'*********************************************************************
!'
!FUNCTION Windows% (HVersion AS INTEGER, NVersion AS INTEGER)
!DIM Regs AS RegTypeX                  ' Registres proc. pour appel d'int.
!DIM Res AS INTEGER                      'Mémoire tampon pour la version
!
!HVersion = 0                          'Initialise les numéros de version
!NVersion = 0
!
!Regs.ax = &H1600              'N°fonc. Test d'installation du mode étendu
!CALL INTERRUPTX(MULTIPLEX, Regs, Regs)
!Res = Regs.ax                                          'Ranger Regs.AX
!
!SELECT CASE Res MOD 256                 'Evaluer octet de poids faible
!
!  CASE &H1, &HFF                                      'Windows /386 actif
!    HVersion = 2                                      'Version principale
!    NVersion = 0                          'Version secondaire inconnue
!    Windows = WIN386X
!
!  CASE &HD, &H80            'Windows inactif ou en mode réel ou standard
!    Regs.ax = &H4680                'Identifier le mode réel ou standard
!    CALL INTERRUPTX(MULTIPLEX, Regs, Regs)
!    IF (Regs.ax MOD 256) = &H80 THEN                    'Windows actif ?
!      Windows = NOWIN                                          ' Non !
!    ELSE                    'Windows fonctionne en mode réel ou standard
!      Regs.ax = &H1605          'Simuler l'installation du DOS Extender
!      Regs.bx = &H0
!      Regs.si = &H0
!      Regs.cx = &H0
!      Regs.es = &H0
!      Regs.ds = &H0
!      Regs.dx = &H1
!      CALL INTERRUPTX(MULTIPLEX, Regs, Regs)
!      IF Regs.cx = &H0 THEN                      'Windows en mode réel ?
!        ELSE                                                      'Oui
!      Windows = WINREAL                  'Non, Windows fonctionne en mode standard
!      Windows = WINSTANDARD
!      END IF
!    END IF
!  CASE ELSE    'Windows en mode étendu. AX contient le numéro de version
!    HVersion = Res AND &HF                ' LSB pour la version principale
!    NVersion = Res \ 256                  ' MSB pour la version secondaire
!    Windows = WINENHANCED
!END SELECT
!
!END FUNCTION
!
```

30. Préserver la compatibilité

Au cours des premières années du PC, la compatibilité jouait un rôle important du point de vue de l'électronique. En effet, seuls les ordinateurs portant l'étiquette "compatibles IBM" pouvaient espérer une exécution sans encombre des logiciels PC. Le Flight simulator de Microsoft, qui n'existait encore que dans la version 1.0, représentait l'outil idéal pour tester la compatibilité.

A l'heure actuelle, le problème lié à la compatibilité électronique ne joue plus aucun rôle puisque les ordinateurs proposés par les innombrables fournisseurs dépendent d'un petit groupe de constructeurs dont le souci essentiel est d'assurer la compatibilité vis-à-vis de leurs concurrents.

Cependant, du point de vue logiciel, le problème de la compatibilité reste toujours aussi important qu'auparavant. Prenons un exemple : l'utilisateur qui accède directement et exclusivement à une carte VGA dans ses programmes ne doit pas s'étonner lorsque son programme ne fonctionne pas sur des ordinateurs avec une carte graphique Hercules. Ce chapitre décrit d'ailleurs les règles à respecter pour garantir la compatibilité logicielle.

30.1. Problèmes de compatibilité lors de la programmation DOS

Nous étudierons dans cet ouvrage trois possibilités d'accéder à l'électronique du PC. Nous pouvons utiliser à cet effet les fonctions DOS ou BIOS déjà disponibles ou bien développer également des fonctions ou routines personnelles qui nous permettent de contrôler l'électronique directement. Cette seconde méthode ne présente guère d'avantage en ce qui concerne les mémoires de masse (disquette et disque dur) et le clavier. Des routines spéciales de sortie sur écran peuvent par contre être beaucoup plus rapides, et donc beaucoup plus efficaces, que les routines offertes par le BIOS et le DOS dans ce domaine.

Du point de vue de la compatibilité, ce sont cependant les fonctions du DOS qui l'emportent haut la main dans toute comparaison. Il convient donc de respecter un certain nombre de règles pour tous ceux qui veulent développer des programmes qui ne soient pas limités aux ordinateurs dont le BIOS ou l'électronique sont compatibles avec une machine bien précise mais qui puissent, au contraire, tourner sans problème sur tous les ordinateurs fonctionnant sous le DOS. Les précautions à prendre sont également rendues nécessaires pour préserver la compatibilité avec les versions futures du DOS, qui soumettront les programmes à de très rudes épreuves avec l'introduction du multi-tâches et d'autres innovations. Si vous souhaitez, par conséquent, développer des programmes sous la version actuelle du DOS de façon à ce qu'ils puissent encore tourner sans problème sous les versions 4, 5 ou 6, nous vous recommandons de suivre les conseils suivants :

- ✓ Pour accéder aux fichiers, utilisez uniquement les fonctions Handle introduites dans la version 2. Seules ces dernières garantissent la compatibilité avec les versions futures de DOS. Evitez de faire appel aux fonctions FCB.

- ✓ Si vous êtes cependant parfois contraint d'avoir recours aux anciennes fonctions FCB (par exemple pour accéder aux fichiers dotés de certains attributs ou bien pour accéder aux sous-répertoires), veillez bien à ne pas ouvrir de FCB déjà ouvert et à ne pas refermer de FCB déjà refermé. Cela pourrait en effet entraîner des problèmes dans le cadre d'un réseau

- ✓ Testez le numéro de version du DOS au début de votre programme et terminez le programme par un message d'erreur s'il ne peut fonctionner sous cette version.

- ✓ Sauvegardez autant que possible dans le bloc d'environnement les constantes nécessaires au bon déroulement de votre programme (par exemple le chemin des programmes ou fichiers à charger à la suite du programme principal). Allez ensuite rechercher ces constantes dans le bloc d'environnement.

- ✓ Libérez à l'aide de la fonction appropriée du DOS toute la mémoire que votre programme n'utilise pas. C'est surtout important pour les programmes COM car ceux-ci se voient attribuer, après avoir été chargés, la totalité de la mémoire restant disponible.

- ✓ Si vous avez besoin de mémoire supplémentaire, ne vous "servez" pas directement mais réclamez la poliment à l'aide de la fonction appropriée du DOS.

- ✓ Ne poussez pas la familiarité avec les vecteurs d'interruption jusqu'à les manipuler directement par des accès à la mémoire mais utilisez plutôt les fonctions du DOS prévues à cet effet.

- ✓ Lorsque vous êtes contraint de modifier le contenu de différents vecteurs d'interruption dans le cadre de votre programme, n'oubliez pas de sauvegarder tout d'abord l'ancienne valeur de ces vecteurs pour pouvoir la rétablir avant de mettre fin à votre programme.

- ✓ Pour mettre fin à votre programme, appelez une des fonctions du DOS (31h ou 4Ch) qui permettent de transmettre au programme d'appel une valeur signalant si le programme a pu ou non être correctement exécuté. Il est préférable de ne plus utiliser les autres fonctions servant à terminer un programme (interruption 20h et fonction 0 de l'interruption 21h).

- ✓ Pour pouvoir localiser plus précisément l'origine d'une erreur lorsqu'il s'en produit une, nous vous conseillons d'utiliser la fonction 59h de l'interruption 21h, dont vous disposez à partir de la version 3 du DOS.

Voici à nouveau, pour terminer, une liste de fonctions du DOS très anciennes qu'il vaut mieux éviter d'utiliser chaque fois que c'est possible. Nous vous indiquons en regard leurs équivalents plus "modernes".

Ancienne fonction		Nouvelle fonction	
00h	Fin du programme	4Ch	Fin de l'opération
0Fh	Ouvrir fichier	3Dh	Ouvrir handle
10h	Fermer fichier	3Eh	Fermer handle
11h	Rechercher première entrée	4Eh	Rechercher première entrée
12h	Rechercher entrée suivante	4Fh	Rechercher entrée suivante
13h	Supprimer fichier	41h	Supprimer entrée de répertoire
14h	Lecture séquentielle	3Fh	Lecture (à travers handle)
15h	Ecriture séquentielle	40h	Ecriture (à travers handle)
16h	Créer fichier	3Ch 5Ah 5Bh	Créer handle ou créer fichier temporaire ou créer nouveau fichier
17h	Renommer fichier	56h	Renommer entrée du répertoire
21h	Lecture sélective	3Fh	Lecture (à travers handle)
22h	Ecriture sélective	40h	Ecriture (à travers handle)
23h	Déterminer taille du fichier	42h	Déplacer pointeur de fichier
24h	Fixer numéro d'enregistrement	42h	Déplacer pointeur de fichier
26h	Créer nouveau PSP	4Bh	Chargement et exécution de fichiers
27h	Lecture sélective	3Fh	Lecture (à travers handle)
28h	Ecriture sélective	40h	Ecriture (à travers handle)

Si vous suivez tous ces conseils, vos programmes pourront tourner également sur d'autres machines que celle prévue au départ, ainsi que sous les versions futures du DOS, sans qu'il soit nécessaire de les modifier ou du moins au prix de modifications minimes.

31. Structures secrètes de DOS

Il existe des ouvrages entièrement consacrés aux aspects peu connus, non documentés ou non publiés du bon vieux DOS. Il satisfont une sorte de voyeurisme dont tout programmeur engagé dans ce domaine peut devenir la victime. Je n'ai pas l'intention ici de suivre ce penchant. Si on estime correctement l'utilité de ces informations pour la programmation en DOS, on s'aperçoit bien vite qu'elle est en général fort minime. Mais on ne peut pas nier non plus qu'il existe plusieurs variables non documentées de DOS qui donnent des informations de première importance que l'on ne peut pas obtenir autrement. Ce sont elles qui vont attirer notre attention dans ce chapitre.

31.1. Secret ou public ?

Pour gérer les ressources d'exploitation (mémoire vive, mémoire de masse), DOS a recours à de nombreuses structures de données. Pour certaines de ces structures il existe une documentation officielle complète et nous avons déjà pu les décrire de façon détaillée dans le cadre de cet ouvrage. Il s'agit :

- ✓ du préfixe du segment de programme (PSP) qui précède chaque programme dans la mémoire,
- ✓ des blocs de contrôle des fichiers (FCB), qui gèrent l'accès aux fichiers,
- ✓ des blocs de contrôle de la mémoire (MCB), qui servent à gérer la mémoire vive,
- ✓ des structures d'en-tête des drivers de périphériques,
- ✓ des blocs d'environnement qui associent à chaque programme en mémoire une série de chaînes de caractères et
- ✓ des nombreuses structures que DOS met en place dans les mémoires de masse (secteur de boot, table d'allocation des fichiers, répertoire racine, etc.).

En plus des structures citées il en existe toute une série d'autres uniquement connues de quelques initiés car elles ne sont pas évoquées par Microsoft dans les documentations techniques fournies avec DOS. Microsoft justifiait cette attitude en alléguant que ces structures n'avaient pas d'utilité pour les programmes et que leur définition risquait d'être modifiée dans les versions futures de DOS. Ces deux arguments ne sont pas très pertinents car certains programmes ont véritablement besoin de ces structures et ils ne pourraient absolument pas être réalisés si elles n'existaient pas.

Les nombreux utilitaires de gestion des disquettes et des disques durs en sont des exemples criants. Si on examine par exemple avec un débogueur le représentant le plus célèbre de cette catégorie, les Norton Utilities, on se rend compte que ces programmes font même un usage très intensif de ces structures et qu'ils y sont contraints. Mais ils ont eu à souffrir de l'instabilité de ces structures. Beaucoup d'entre eux ne fonctionnaient plus lorsqu'on passait de la version 3.3 à la version 4.0 de DOS car Microsoft avait opéré des changements internes sur certaines des structures exploitées de façon illégitime.

Maintenant que l'avertissement a été compris, jetons un coup d'oeil sur les plus importantes des structures internes cachées de DOS, le bloc d'information de DOS ou DIB.

31.2. Le bloc d'informations de DOS

La clé d'accès aux principales structures de DOS est le bloc d'informations de DOS (DOS Info Block =DIB). Il s'agit d'une structure qui contient des pointeurs sur d'autres structures de DOS et quelques autres informations. Savoir que le DIB existe et connaître sa structure ne sert à rien si on ignore son adresse en mémoire. Or cette adresse n'est pas inscrite dans un emplacement de mémoire défini une fois pour toutes et elle ne peut pas non plus être obtenue à l'aide de l'une des fonctions documentées de l'API de DOS. Il faut donc avoir recours à une fonction non documentée, la fonction 52h, qui sert uniquement à cette fin et qui renvoie l'adresse du bloc d'informations de DOS dans les registres ES:BX.

Contrairement à toutes les autres fonctions de DOS qui renvoient des pointeurs sur une structure ou sur une zone de données, le contenu des registres ES:BX, à l'issue de l'appel, ne référence pas le premier champ du DIB mais directement le second.

Structure du DIB

Le premier champ du DIB contient un pointeur FAR sur le bloc de contrôle MCB de la première zone de mémoire allouée. Vous trouverez des informations détaillées sur cette structure et sa fonction à l'intérieur du système dans le chapitre consacré à la gestion de la mémoire vive sous DOS.

Structure du bloc d'informations de DOS (DIB)		
Adresse	**Contenu**	**Type**
-04h	Pointeur sur le premier MCB	1 PTR
ES:BX	Pointeur sur le premier bloc de paramètres du lecteur (DPB)	1 PTR
+04h	Pointeur sur le dernier buffer de DOS	1 PTR
+08h	Pointeur sur le driver de l'horloge ($CLOCK)	1 PTR
+0Ch	Pointeur sur le driver de console (CON)	1 PTR
+10h	Longueur maximale d'un secteur (pour toutes les mémoires de masse connectées)	1 WORD
+12h	Pointeur sur le premier buffer de DOS	1 PTR
+16h	Pointeur sur la table des chemins d'accès	1 PTR

Structure du bloc d'informations de DOS (DIB)		
Adresse	Contenu	Type
+1Ah	Pointeur sur la table des fichiers système (SFT)	1 PTR
Longueur : 1Eh (30 octets)		

Le bloc de paramètres du lecteur

Le pointeur qui forme le second champ du DIB permet d'accéder à quantité d'informations impossibles à obtenir autrement. Il référence le premier Bloc de paramètres d'un périphérique (Drive Parameters Block = DPB), une structure que DOS met en place pour chaque mémoire de masse connectée (disquette, disque dur, streamer etc.).

Structure du bloc de paramètres d'un périphérique		
Adresse	Contenu	Type
+00h	Numéro ou nom du périphérique (0 = A, 1 = B etc.)	1 BYTE
+01h	Numéro de sous-unité attribué par le driver du périphérique	1 BYTE
+02h	Octets par secteur	1 WORD
+04h	Facteur Interleave	1 BYTE
+05h	Secteurs par cluster	1 BYTE
+06h	Secteurs réservés (pour le secteur Boot)	1 WORD
+08h	Nombre de tables d'allocation des fichiers (FATs)	1 BYTE
+09h	Nombre d'entrées du répertoire racine	1 WORD
+0Bh	Premier secteur occupé	1 WORD
+0Dh	Dernier cluster occupé	1 WORD
+0Fh	Secteurs par table d'allocation	1 BYTE
+10h	Premier secteur de données	1 WORD
+12h	Pointeur sur l'en-tête du driver de périphérique associé	1 PTR
+16h	Descripteur du support	1 BYTE
+17h	Indicateur d'exploitation (0FFh = le périph. n'a pas encore été appelé)	1 BYTE
+18h	Pointeur sur le prochain DPB (xxxx:FFFF = plus de DPB)	1 PTR
Longueur : 1Ch (28 octets)		

Le premier champ du DPB indique la désignation du périphérique concerné. 0 représente l'unité A, 1 l'unité B, 2 l'unité C etc. La signification du second champ qui indique le numéro de sous-unité n'est pas aussi évidente. Pour la comprendre, il faut se

rappeler que l'accès aux différents périphériques s'effectue à travers ce qu'on appelle des drivers de périphériques. Pour que DOS ne soit pas obligé de se préoccuper des caractéristiques physiques d'une mémoire de masse, DOS n'accède pas directement à une unité de disquette ou à un disque dur mais fait appel à un driver de périphérique qui sert d'intermédiaire entre DOS et le matériel.

Il n'existe pas de driver de périphérique séparé pour chaque périphérique car un seul driver peut fort bien supporter plusieurs périphériques. C'est ainsi par exemple qu'un driver intégré dans DOS gère la ou les unités de disquette et le premier disque dur s'il en existe un. DOS met donc en place un DPB pour chacun de ces périphériques : 3 DPB seront ainsi installés automatiquement dans le cas d'un système à disque dur. (Un DPB est systématiquement prévu pour l'unité de disquette B, même si elle n'est pas montée). Pour que le driver puisse distinguer les périphériques qu'il gère, un numéro d'ordre compris entre 0 et le nombre de périphériques moins 1 leur est attribué. C'est ce numéro qui figure dans le champ de sous-unité (Sub Unit).

Le champ suivant indique le nombre d'octets par secteur. Sous DOS, c'est en fait toujours 512 octets. Vient ensuite ce qu'on appelle le facteur Interleave qui exprime le décalage des secteurs logiques par rapport aux secteurs physiques lors du formatage du support. Il ne faut pas prendre cette valeur trop au sérieux car les systèmes de disques modernes trompent DOS en mettant ici des valeurs fictives.

Ces deux champs sont suivis de toute une série de champs qui ont déjà été cités à propos de la gestion des mémoires de masse par DOS (Chapitre 6). Ils décrivent la situation et la taille des structures que DOS met en place dans les mémoires de masse pour les gérer.

Ils contiennent notamment un pointeur sur l'en-tête du driver auquel DOS a recours pour accéder au périphérique. Ce pointeur permet d'obtenir d'autres informations intéressantes car l'en-tête du driver de périphérique indique par exemple l'attribut du driver de périphérique.

On repère ensuite le champ du descripteur de support (Media Descriptor) et de l'indicateur d'exploitation. Ce dernier contient la valeur 0FFh tant qu'aucun accès au périphérique n'a été effectué. Après le premier accès, il est mis à 0 et ne se modifie plus par la suite.

Le DPB se termine par un pointeur servant de lien avec le DPB suivant. Chaque DPB se terminant par un pointeur de ce type, l'ensemble forme une liste chaînée. Pour signaler la fin de la chaîne, l'adresse d'offset de ce pointeur contient la valeur 0FFFFh dans le dernier DPB.

Accès aux DPB

Lorsque dans un programme vous avez besoin d'informations contenues dans le DPB, vous avez plusieurs possibilités pour déterminer l'adresse du DBP recherché. La

première consiste à suivre la méthode que nous venons de décrire, c'est-à-dire à déterminer d'abord l'adresse du DIB. Cette adresse vous permet alors de lire le pointeur réénfrençant le premier DPB et de suivre la chaîne des DPB jusqu'à ce que vous ayez atteint le DPB recherché. Une seconde possibilité, un peu plus simple et moins exposée aux modifications du DIB, consiste à recourir à deux fonctions non documentées de DOS. Il s'agit des fonctions 1Fh et 32h qui font partie du jeu de fonctions de DOS depuis la version 2.0. Quand elles sont invoquées, ces deux fonctions fournissent un pointeur référençant un DPB dans les registres DS:BX. Alors que la fonction 1Fh fournit systématiquement un pointeur sur le DPB de l'unité de disque actuelle, l'adresse renvoyée par la fonction 32h se rapporte au périphérique dont le numéro a été transmis à la fonction dans le registre DL lors de l'appel. (0 représentant l'unité de disque courante, 1 l'unité A, 2 l'unité B, etc.) Cette fonction est donc plus souple et devrait pour cette raison être préférée à la fonction 1Fh.

L'accès aux différents DPB par les fonctions 1Fh et 32h présente un autre avantage car il oblige DOS à rechercher certaines informations comme le facteur Interleave et l'octet de descripteur de support (Media Descriptor) par exemple, qui ne sont disponibles pour les unités de disquette qu'après un premier accès. Si par contre vous accédez au DPB par le pointeur du bloc DIB, il peut arriver que certains champs n'aient pas encore été initialisés et que des valeurs erronées vous soient fournies.

Le buffer de DOS

A coté du pointeur sur le premier DPB, le DIB comporte à l'adresse 12h un pointeur sur le premier buffer DOS. Les buffers ainsi nommés existent indépendamment du nombre de périphériques. Ils servent à stocker des secteurs en mémoire centrale. Il faut savoir en effet que les secteurs sont toujours lus intégralement par le système alors que les programmes d'application par le moyen de l'API de DOS ne réclament souvent que des parties de secteurs. Certains secteurs sont mémorisé de façon permanente parce qu'ils sont sans cesse relus par le programme en cours d'exécution, par exemple la table d'allocation des fichiers ou le catalogue du répertoire principal.

Le nombre de buffers peut être fixé par l'utilisateur à l'intérieur du fichier de configuration CONFIG.SYS. Si ces buffers offrent plus de place mémoire qu'il n'est nécessaire pour la table d'allocation des fichiers, le répertoire racine et les sous-répertoires, des secteurs ordinaires y seront également sauvegardés dans l'espoir qu'ils soient rechargés sous peu, auquel cas ils pourront être lus directement dans le buffer.

Les secteurs sont reliés par une liste chaînée pour permettre à DOS, lors de chaque opération de lecture, d'examiner tout d'abord si le secteur à lire ne figure pas par hasard dans l'un des buffers.

Comme pour les DPB, la liste chaînée est construite à l'aide d'un pointeur placé au tout début de chaque buffer. Ici aussi, le dernier buffer est atteint lorsque l'offset présenté par le pointeur est 0FFFFh.

Structure d'un buffer de DOS		
Adresse	**Contenu**	**Type**
+00h	Pointeur sur le prochain buffer de DOS	1 PTR
+04h	Numéro de l'unité de disque (0=A, 1=B etc.)	1 BYTE
+05h	Indicateurs	1 BYTE
+06h	Numéro de secteur	1 WORD
+08h	Réservé	2 BYTE
+0Ah	Contenu du secteur mémorisé	512 BYTES
Longueur : 210h (528 octets)		

Le champ de liaison avec le buffer suivant est suivi du numéro de l'unité de disque dont provient le secteur dans le buffer. La valeur 0 représente ici l'unité A, 1 l'unité B, 2 l'unité C etc. Outre le code de l'unité de disque, le numéro du secteur est également indispensable pour identifier précisément le secteur. Ce numéro se présente dans la mémoire 06h du buffer. Pour que DOS puisse obtenir des informations sur le périphérique à partir duquel le secteur a été chargé dans le buffer, le dernier champ de l'en-tête du buffer contient un pointeur sur le DPB correspondant. Bien que ce champ soit le dernier champ de l'en-tête du buffer de DOS, il n'est pas directement suivi du secteur stocké dans le buffer car deux octets viennent encore s'intercaler avant ce secteur. Il s'agit probablement d'une place réservée à des extensions futures.

La table des chemins d'accès

Ce n'est pas la dernière fois que nous rencontrons le DPB dans l'en-tête d'un buffer de DOS. Il réapparaît en effet dans ce qu'on appelle la table des chemins d'accès qui commence à l'adresse 16h du DIB. Cette table contient pour chaque unité de disque le chemin d'accès courant ainsi qu'un pointeur sur le DPB de l'unité de disque.

```
       0  1  2  3  4  5  6  7  8  9  A  B  C  D  E  F
0000: 41 3A 5C 43 41 43 48 45-00 00 00 00 00 00 00 00   A:\CACHE........
0010: 00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00   ................
0020: 00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00   ................
0030: 00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00   ................
0040: 00 00 00 00 40 20 74 80-02 27 03 FF FF FF FF 02   ....@ t..'......
0050: 00 42 3A 5C 00 00 00 00-00 00 00 00 00 00 00 00   .B:\............
0060: 00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00   ................
0070: 00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00   ................
0080: 00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00   ................
0090: 00 00 00 00 00 40 40 74-80 02 00 00 FF FF FF FF   .....@@t........
00A0: 02 00 43 3A 5C 54 43 5C-42 41 55 53 5C 41 53 4D   ..C:\FONREV_\ASM
00B0: 5C 48 45 52 43 4D 4F 4E-4F 00 00 00 00 00 00 00   \HERCMONO.......
00C0: 00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00   ................
00D0: 00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00   ................
00E0: 00 00 00 00 00 00 40 60-74 80 02 65 05 FF FF FF   ......@'t..e....
00F0: FF 02 00 44 3A 5C 4D 53-43 5C 42 49 4E 00 00 00   ...D:\MSC\BIN...
0100: 00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00   ................
0110: 00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00   ................
0120: 00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00   ................
0130: 00 00 00 00 00 00 00 40-00 00 80 0D 17 00 FF FF   .......@........
0140: FF FF 02 00
```

Extrait de mémoire montrant le contenu de la table des chemins d'accès

Si l'instruction LASTDRIVE= figure dans le fichier de configuration du système, la table contient des chemins pour toutes les unités de A jusqu'à la lettre spécifiée. Si par contre cette instruction fait défaut, la table contiendra juste le nombre de chemins correspondant aux périphériques supportés par les drivers de périphériques installés.

En modifiant les éléments de cette table, on peut rediriger une unité de disque vers une autre. C'est cette possibilité qu'exploitent les instructions JOIN et SUBSTitute de DOS car elles manipulent à l'intérieur de la table le chemin associé à l'unité à rediriger.

32. Programmes résidents

Depuis sa "naissance", DOS souffre de ce que les fées qui se sont penchées sur son berceau ne l'aient pas doté du don de "multitâche", c'est-à-dire de la faculté d'exécuter plus d'un programme à la fois. Ce n'est que maintenant, après plus de dix ans de règne, que DOS est complété par deux systèmes d'exploitation qui comblent cette lacune : OS/2 et Windows.

Les programmes dits résidents ou TSR (TSR = terminate and stay resident) apportent comme un semblant de multitâche dans l'univers du DOS. Ces programmes, une fois lancés, sommeillent à l'arrière-plan en attendant d'être activés. Ils se sont surtout imposés depuis l'apparition de SideKick.

Un programme résident ne fait pas un véritable travail multitâche puisqu'il n'y a pas d'exécution simultanée de plusieurs programmes. Mais l'utilisateur peut à tout moment frapper une "touche d'activation" pour appeler le programme. Le travail en cours est alors interrompu, l'écran sauvegardé, et l'utilisateur se voit offrir différents services. Lorsqu'il met fin au programme résident, l'écran est restauré et le travail interrompu se poursuit sans qu'il se soit aperçu de rien.

Les programmes résidents permettent notamment d'accéder à tout moment à des outils aussi utiles qu'une calculatrice, un agenda ou un bloc-notes. Par exemple avec SideKick et ses émules l'utilisateur peut rapidement prendre ou lire des notes sans sortir du programme en cours.

En dehors de ces applications typiques, les programmes résidents sont également employés comme générateurs de macrocommandes, formateurs d'écran et même comme correcteurs orthographiques.

Certains programmes peuvent même entrer en interaction avec les programmes interrompus et échanger des données avec eux. C'est ainsi par exemple qu'un programme résident peut transmettre une page d'agenda à un traitement de texte.

Bien que les applications les plus diverses puissent être transformées en programmes résidents, ces derniers présentent tous un mode de fonctionnement et une structure identiques que nous allons étudier dans ce chapitre.

Avant de jeter un coup d'oeil dans les coulisses des programmes résidents, nous ne vous cacherons pas toutefois qu'il s'agit d'un sujet extrêmement complexe dont la compréhension réclame une parfaite connaissance du fonctionnement interne du système. Il ne faut pas commencer la lecture de cet ouvrage par le présent chapitre : il est indispensable avant de vous y attaquer d'acquérir une certaine expérience de la programmation système.

Cette exigence est notamment due au fait que par définition les programmes résidents sortent du cadre classique des systèmes monotâches, dans lesquels chaque programme

peut en principe accéder tout seul à l'ensemble des ressources du système (mémoire vive, écran, disque dur etc.). Ici en plus du programme de premier plan il existe un programme résident qui surveille son activité en arrière-plan. A l'intérieur du système, le programme résident doit donc affronter de nombreux concurrents, tels que le BIOS, DOS, le programme interrompu, mais aussi les autres programmes résidents. Tâcher d'y parvenir n'est pas une mince affaire, même si cela représente un défi très stimulant à relever. De toute façon, la solution passe par l'usage de l'assembleur.

Mais ce langage ne convient pas très bien pour programmer des sujets typiquement résidents comme des calculatrices ou des blocs-notes pour lequels des langages évolués comme Pascal ou C sont plus appropriés. C'est pourquoi dans ce chapitre, nous travaillerons avec deux programmes en assembleur permettant de "convertir" en programmes résidents des programmes ordinaires écrits en Pascal ou en C. Mais avant de nous pencher sur ces programmes, nous allons tout d'abord étudier comment activer des programmes résidents.

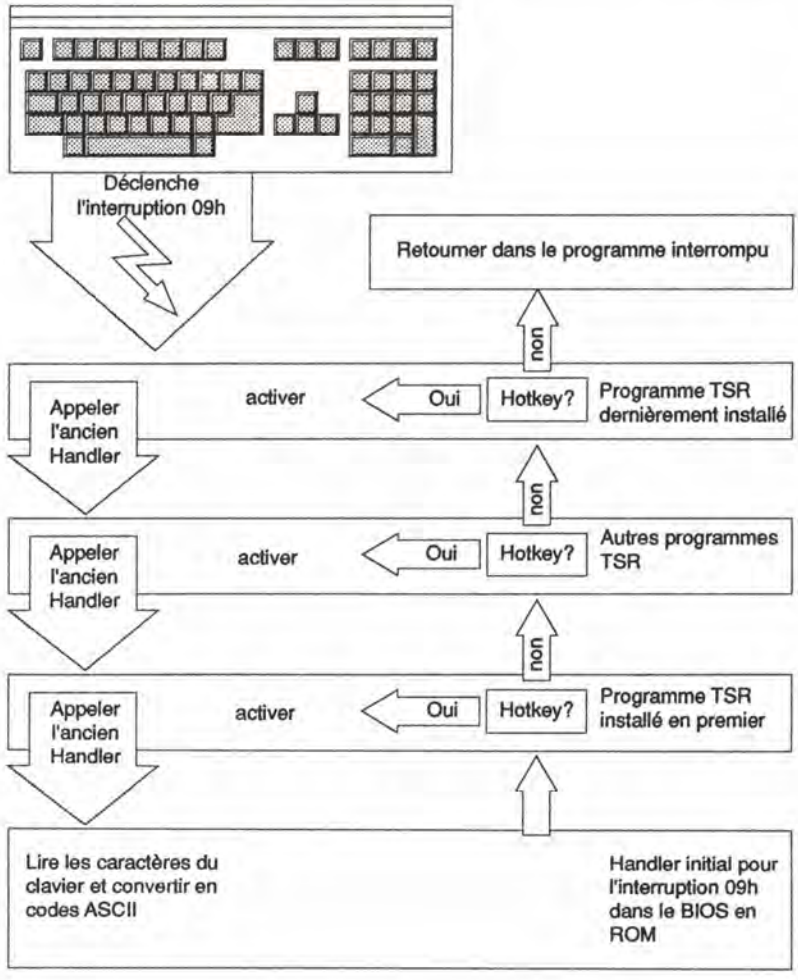
32.1. Activation de programmes résidents

Si nous exigeons de notre programme résident qu'il passe au premier plan dès qu'une certaine combinaison de touches est actionnée (= hot Key ou touche d'activation), il nous faut naturellement installer une sorte de mécanisme de déclenchement qui soit d'une manière ou d'une autre relié au clavier. On peut avoir recours à cet effet aux interruptions 09h et 16h, qui sont toutes deux appelées par le système pour la gestion du clavier. L'interruption 16h est la fameuse interruption clavier du BIOS qui permet aux programmes de lire les caractères et l'état du clavier. Si nous utilisions cette interruption, notre programme résident ne pourrait être activé que lorsque cette interruption est appelée par un programme de lecture du clavier.

L'interruption 09h convient donc mieux à notre objectif car elle est déclenchée par le processeur du clavier chaque fois qu'une touche est actionnée ou relâchée. Il nous faut donc rediriger cette interruption vers une routine de notre cru qui vérifiera si le programme résident doit être activé ou non.

Avant de procéder à cette activation, elle devra appeler la routine qui était responsable de l'interruption 09h avant que le programme résident ne soit installé. Il y a deux raisons à cela. La première tient au rôle même de l'interruption 09h. Cette interruption est chargée d'informer le système que le clavier requiert son attention pour lui transmettre des informations sur le fait qu'une touche a été actionnée ou relâchée. L'interruption 09h est donc normalement dirigée sur une routine du BIOS chargée de recevoir et d'exploiter les informations en provenance du clavier. Concrètement, elle reçoit le code d'une touche et le convertit en un code ASCII qu'elle place dans le buffer clavier du BIOS. Comme notre programme résident ne veut et ne peut pas se charger de cette tâche, la routine d'origine doit être appelée, faute de quoi plus aucune entrée au clavier ne serait possible.

La seconde raison est qu'il est possible que d'autres programmes résidents aient déjà été installés avant le nôtre et que ces programmes aient déjà redirigé l'interruption 09h vers une routine particulière. Comme notre programme figure maintenant, dans la chaîne des gestionnaires d'interruption, avant ces programmes, leurs routines d'interruption ne seront plus appelées automatiquement si nous empêchons l'appel de l'ancien gestionnaire d'interruption. Il en résulterait que ces programmes résidents ne pourraient plus être invoqués. C'est pourquoi tous les programmes résidents doivent appeler l'ancien gestionnaire d'interruption avant ou après leur propre traitement d'interruption.



Chaîne des programmes résidents (exemple de l'interruption 09h)

Cet appel ne doit pas s'effectuer avec l'instruction INT du langage machine car c'est notre gestionnaire d'interruption qui risquerait ainsi de se déclencher à nouveau. Dans

la plupart des cas, il en résulterait un empilage infini d'appels récursifs qui entraînerait un débordement de la pile, d'où un plantage du système. Pour éviter cela, l'adresse de l'ancien gestionnaire d'interruption doit être recherchée lors de l'installation du programme résident et être stockée dans une variable interne au programme résident. A l'aide de l'adresse ainsi sauvegardée, l'ancien gestionnaire d'interruption pourra alors être appelé avec une instruction FAR CALL. Pour simuler l'appel de ce gestionnaire avec l'instruction INT, le contenu du registre des indicateurs doit être placé sur la pile à l'aide de l'instruction PUSHF, avant l'instruction CALL, comme cela se fait automatiquement lorsque l'instruction INT est exécutée.

L'endroit exact où l'ancien gestionnaire d'interruption doit être appelé à l'intérieur du nouveau dépend toujours de la logique de cette routine. Dans les deux modules en assembleurs présentés à la fin de ce chapitre on prend soin d'effectuer d'abord d'autres tâches. Dans ce cas précis il faudra commencer par lire le contenu du port 60h. C'est le port où le clavier dépose le numéro de la touche tapée, autrement dit son scan code. L'ancien gestionnaire d'interruption qui va être appelé par la suite lira également ce port pour prendre connaissance de la touche frappée. Mais cette lecture ne sera pas influencée par la lecture précédente effectuée par le nouveau gestionnaire car le port conserve son contenu jusqu'à ce qu'une autre touche soit actionnée.

En lisant le contenu du porte 60h du clavier, le nouveau gestionnaire d'interruption du programme résident peut voir si l'utilisateur a tapé la touche d'activation. En fait ce test ne porte sur l'instant que sur une partie de "la" touche d'activation car celle-ci est en général constituée par une combinaison d'une lettre ou d'un chiffre avec une touche de commande du type <Majuscule> <Ctrl>, <Alt> etc...Si le gestionnaire d'interruption trouve le code de la lettre ou du chiffre impliqués dans l'activation, il n'a fait que la moitié du chemin. Car il devra encore tester si l'utilisateur a frappé en même temps la touche de commande associée. C'est ainsi que le contenu de l'indicateur d'état du clavier de DOS est soigneusement inspecté. Rappelons qu'il se trouve à l'adresse 17h du segment de variables du BIOS (segment 0040h). Chaque touche de commande s'y trouve représentée par un bit que l'on repère en appliquant le masque adéquat. Une fois la touche d'activation détectée, le programme résident ne peut pas se déclencher sans précaution car il se trouve confronté à un sévère problème évoqué sous le nom de "Non-réentrance de DOS".

DOS n'est pas réentrant

Comme le programme résident peut être activé à tout moment à l'aide du clavier, c'est-à-dire sans tenir compte des autres opérations en cours à l'intérieur du système, il est possible qu'il interrompe l'exécution d'une fonction de DOS. Cela ne pose d'ailleurs pas de problème en soi, pourvu que le programme résident, une fois terminé, revienne comme il faut à la fonction de DOS interrompue et que l'exécution de celle-ci puisse se poursuivre. Des problèmes peuvent cependant survenir si le programme résident appelle à son tour des fonctions de DOS, ce qu'il est pratiquement impossible d'éviter si on programme avec un langage évolué. C'est là le problème de la réentrance. Le terme

désigne la faculté d'un système de faire appeler et exécuter son code programme par plusieurs programmes en même temps. Or DOS n'est justement pas réentrant car, en tant que système monotâche, il considère a priori que ses différentes fonctions ne peuvent pas être appelées simultanément (en parallèle) mais seulement l'une après l'autre (en série).

L'appel d'une fonction de DOS par un programme résident lors de l'exécution d'une autre fonction pose des problèmes à DOS car au moment du déclenchement de l'interruption 21h il charge l'adresse de l'une des trois piles de DOS dans les registres SS:SP du processeur. Laquelle des trois piles est utilisée dépend du groupe de fonctions dans lequel le DOS range la fonction concernée, et cela n'est pas identifiable par l'utilisateur. Pendant l'exécution de la fonction de DOS appelée, des données temporaires sont placées sur cette pile, mais aussi et surtout les adresses de retour des sous-programmes. Si l'exécution de la fonction est alors interrompue par l'activation d'un programme résident qui appelle à son tour une fonction de DOS, DOS va à nouveau charger l'adresse de départ d'une pile interne dans les registres SS:SP. S'il s'agit de la même pile déjà employée lors de l'exécution de la fonction interrompue (ce qui ne peut jamais être exclu), tout accès à la pile détruira alors irrémédiablement les informations que la fonction de DOS interrompue y avait placées. Les appels de fonctions de DOS par le programme résident s'exécuteront sans problème mais les difficultés commenceront lorsque le programme résident sera terminé et que le contrôle sera rendu à la fonction de DOS interrompue par le lancement du programme. Le contenu de la pile ayant été modifié par les appels de fonction de DOS intervenus entre-temps, un comportement imprévisible de la fonction en résultera, qui finira par planter le système.

Pour résoudre ce problème de réentrance, la seule possibilité consiste, pour un programme résident, à renoncer à appeler les fonctions de DOS ou à ne permettre l'activation du programme résident que lorsqu'aucune fonction de DOS n'est en cours d'exécution. La première possibilité étant exclue pour les raisons indiquées plus haut, il faut recourir à la seconde. Le système DOS nous aide en cela, en mettant à notre disposition un indicateur INDOS qui n'est normalement utilisé qu'à l'intérieur de DOS mais qui peut être très utile pour le but que nous poursuivons. Il s'agit d'un compteur du nombre des imbrications des appels à DOS. S'il contient la valeur 0, c'est qu'aucune fonction de DOS n'est actuellement en train d'être exécutée. La valeur 1 indique au contraire qu'une fonction de DOS est actuellement en cours d'exécution. Dans certaines situations bien précises, cet indicateur peut aussi contenir des valeurs plus élevées, lorsqu'une fonction a appelé une autre fonction DOS au cours de son exécution, ce qui n'est toutefois permis que dans des cas très particuliers.

Comme le contenu de cet indicateur ne peut naturellement pas être testé à l'aide d'une fonction de DOS (car elle risquerait d'être un obstacle pour son propre appel), son contenu doit être lu directement dans la mémoire. L'adresse en question n'étant plus modifiée après le lancement du système, il suffit pour cela de la rechercher au cours de l'installation du programme résident et de la sauvegarder dans une variable. On peut se servir pour cela de la fonction 34h de DOS (non documentée) qui renvoie l'adresse du indicateur INDOS dans les registres ES:BX.

Le test de cet indicateur doit ensuite être intégré dans le gestionnaire de l'interruption 09h. Il faut vérifier si la touche d'activation a été actionnée. Si tel est le cas, l'activation du programme résident ne doit être autorisée qu'à condition que l'indicateur INDOS contienne la valeur 0.

Tous les problèmes ne sont pas réglés pour autant. Si l'activation du programme résident est maintenant coordonnée avec les appels de fonction de DOS par un programme transitoire exécuté au premier plan, le programme résident ne pourra néammoins pas être activé à partir de l'environnement DOS. L'interpréteur de commandes DOS (COMMAND.COM) a en effet lui-même recours à quelques fonctions de DOS pour afficher le message d'attente de DOS et pour recevoir les saisies de l'utilisateur. L'indicateur INDOS contient donc en permanence la valeur 1. Dans ce cas particulier, une interruption de la fonction de DOS en cours d'exécution serait sans grand risque, à condition toutefois que l'on puisse être sûr que l'indicateur INDOS contient la valeur 1 à cause de l'interpréteur de commandes du DOS et non par suite des activités d'un programme transitoire.

Il existe heureusement une solution à ce problème. Elle est liée au fait que DOS se trouve dans une sorte d'état d'attente lorsqu'il attend des frappes de l'utilisateur à l'intérieur de l'interpréteur de commandes. Pour ne pas gaspiller inutilement le temps d'exécution du processeur, DOS appelle à intervalles réguliers l'interruption 28h qui est chargée de l'activation d'opérations d'arrière-plan telles que le spooling de l'imprimante (instruction PRINT du DOS) et d'autres tâches semblables. Si cette interruption (appelée Idle = inoccupé) a été appelée, on peut considérer qu'il n'y a pas grand risque à interrompre DOS et à activer le programme résident.

Pour tirer parti de ce comportement du DOS, on installera un nouveau gestionnaire pour l'interruption 28h lors du lancement du programme résident. Ce gestionnaire appellera tout d'abord l'ancien gestionnaire et examinera alors si l'utilisateur a actionné la touche d'activation. Si tel est le cas, le programme résident pourra être activé même si dans certains cas le flag INDOS contient une valeur différente de 0.

Une autre condition doit cependant être respectée (qui vaut également à l'intérieur du gestionnaire de l'interruption clavier 09h) : un programme résident ne doit pas être activé si des interruptions critiques sont en cours d'exécution.

Interruptions critiques

Il s'agit d'actions qui, pour différentes raisons, ne peuvent être interrompues et qui doivent donc se terminer en un temps relativement court. Sur un PC, il s'agit surtout des accès à la disquette ou au disque dur qui sont gérés au niveau le plus bas par l'interruption 13h du BIOS. Si l'accès à ces périphériques ne se termine pas dans un délai extrêmement bref, des perturbations très sérieuses peuvent en résulter dans le fonctionnement du système. La situation peut même devenir dramatique si le programme résident exécute un accès à ces périphériques alors qu'un autre accès (déclenché

par le programme qui vient d'être interrompu) n'est pas encore terminé. Il se peut que le système ne se plante pas mais il est certain que des données seront perdues.

La solution consiste, ici aussi, à installer un gestionnaire d'interruption de notre cru pour l'interruption 13h du BIOS. Lorsqu'il sera appelé, ce gestionnaire mettre à 1 un indicateur interne signalant que l'interruption disque du BIOS est actuellement activée. Il appellera ensuite l'ancien gestionnaire de cette interruption, qui exécutera l'accès à la disquette ou au disque dur. Lorsqu'il reviendra au gestionnaire du programme résident, ce dernier annulera alors l'indicateur qui avait été mis à 1 et signalera ainsi la fin de l'activité de l'interruption disque du BIOS.

Pour éviter que cette interruption ne soit interrompue, les autres gestionnaires d'interruption à l'intérieur du programme résident tiendront compte de cet indicateur et n'activeront le programme résident que lorsque le contenu de l'indicateur signalera que l'interruption disque du BIOS n'est pas activée.

Récursion

Une dernière condition à remplir pour activer le programme résident est de s'assurer qu'aucun appel récursif ne puisse avoir lieu. Rien n'empêchant en effet l'utilisateur d'actionner la touche d'activation après que le programme résident ait été activé, il faut interdire que le programme résident puisse être à nouveau être activé avant d'être arrivé à son terme. La solution est très simple puisqu'il suffit d'installer ici encore un indicateur qui sera mis à 1 lors de l'activation du programme résident et sera remis à 0 lorsque le programme sera terminé. Lorsque la touche d'activation sera actionnée, si un des gestionnaires d'interruption constate au vu de cet indicateur que le programme résident a déjà été activé, il ignorera tout simplement la touche d'activation.

Si toutes les conditions indiquées sont remplies, plus rien ne s'oppose à ce que le programme résident soit activé.

Activation différée

Compte tenu des activités de DOS ou du BIOS un programme résident ne peut pas être exécuté au premier plan à n'importe quel moment. C'est pourquoi la plupart des programmes résidents réaménagent également le gestionnaire de l'interruption 08h du timer pour pouvoir différer leur activation. Lorsque la touche d'activation est détectée par le gestionnaire d'interruption du clavier, un indicateur spécial est initialisé pour signaler que le programme résident ne peut pas être déclenché sur le champ.

Cet indicateur est testé à l'intérieur du nouveau gestionnaire de l'interruption du timer qui est appelé 18.2 fois par seconde, à moins que le programme en cours d'exécution n'ait modifié ce rythme. Si le gestionnaire du timer constate que le programme résident

est en attente de déclenchement et que ni DOS ni le BIOS ne sont actifs, plus rien ne s'oppose au déclenchement attendu.

Il faut cependant limiter le délai qui sépare la frappe de la touche d'activation du déclenchement effectif du programme résident. Si l'opération de DOS en cours est particulièrement longue, l'attente peut théoriquement durer plusieurs secondes. Mais l'utilisateur ne sait pas si la touche qu'il a frappée a été prise en compte.

L'indicateur initialisé pour demander le différé d'activation va également servir à décompter le temps et être décrémenté par le timer. Le déclenchement ne reste potentiel que tant que cet indicateur reste positif. Si le gestionnaire d'interruption initialise l'indicateur avec la valeur 6, le programme résident devra être appelé dans l'intervalle des 6 prochains tops d'horloge, ce qui correspond à environ 1/3 de seconde. Une fois ce délai dépassé, la touche d'activation n'est plus prise en compte.

Changement de contexte

Les opérations d'activation d'un programme résident représentent ce qu'on appelle en informatique un changement de contexte. Le contexte (ou en d'autres termes l'environnement) est constitué par toutes les informations nécessaires à l'exploitation d'un programme. Il s'agit par exemple du contenu des registres du processeur, d'informations importantes du système d'exploitation, et aussi de la mémoire occupée par le programme. Nous n'avons toutefois pas à nous préoccuper de ce dernier point lors du changement de contexte car notre programme résident a été marqué comme résident lors de son installation et la mémoire qu'il occupe lui est donc réservée et ne peut être attribuée par le système d'exploitation à d'autres programmes.

Mais les registres du processeur, et en premier lieu les registres de segment doivent recevoir les valeurs que le programme résident attend lors de son exécution. Elles ont été sauvegardées à cet effet dans des variables internes lors de l'installation du programme résident où nous pouvons maintenant les lire. Comme le contenu de ces registres et de tous les autres va être modifié lors de l'exécution du programme résident, il faut s'occuper de les sauvegarder car ils font partie du contexte du programme interrompu et ne devront paraître inchangés lorsque l'exécution de ce programme reprendra.

Il en va de même pour les informations du système d'exploitation dépendant du contexte. Pour DOS seul le PSP (Segment de préfixe du programme) ainsi que la zone appelée DTA (Disk Transfer Area) sont importants. Les adresses de ces deux structures doivent donc être déterminées lors de l'installation du programme résident et être sauvegardées pour pouvoir être restaurées lors du changement de contexte. Il ne faut pas oublier non plus de sauvegarder l'adresse du PSP et de la DTA du programme interrompu avant de passer au contexte du programme résident, de façon à ce que ces adresses puissent être rétablies à leurs valeurs d'origine une fois le programme résident terminé.

L'adresse de la DTA peut être fixée et lue à l'aide de deux fonctions de DOS (respectivement les fonctions 1Ah et 2Fh) mais il n'existe pas de fonctions documentées équivalentes pour le PSP dans toutes les versions de DOS. A partir de la version 3.0 de DOS, la fonction documentée numéro 62h permet de déterminer l'adresse du PSP courant mais il manque toujours une fonction permettant de fixer cette adresse. Ces fonctions existent cependant depuis la version 2.0 de DOS en tant que fonctions non documentées. Il s'agit de la fonction 50h ("fixer l adresse du PSP") et de la fonction 51h ("lire l'adresse du PSP"). Ces deux fonctions sont utilisées à l'intérieur de notre programme résident.

Une dernière précaution s'impose lorsque le programme est activé à l'aide de l'interruption 28h. Lorsque l'activation provient de cette interruption, on interrompt du même coup une fonction DOS active dont le contenu de la pile doit absolument être préservé. C'est pourquoi on va généralement rechercher les 64 mots supérieurs de la pile courante pour les sauvegarder dans la pile du programme résident. Le changement de contexte pour passer au programme résident est ainsi achevé et ce dernier peut être lancé.

A partir de cet instant, le programme résident peut être considéré comme un programme tout à fait normal, capable d'appeler n'importe quelle fonction de DOS ou du BIOS. De tous les concurrents à l'intérieur du système, seul subsiste encore le programme de premier plan, que le programme résident doit respecter en ce sens qu'il ne doit pas détruire le contenu de son écran (ou alors il doit le rétablir à l'issue de son exécution)

Sauvegarde du contexte d'écran

Si les tâches décrites jusqu'ici incombaient obligatoirement à l'interface en assembleur (approfondie par la suite), la sauvegarde du contexte d'écran entre dans le cadre des attributions du programme en C ou en Pascal constituant le véritable programme résident. Le contexte d'écran comprend le mode vidéo courant, la position du curseur et le contenu de l'écran. De plus, pour les cartes graphiques, il convient de sauvegarder également le contenu des registres de sélection des couleurs et des autres registres de la carte vidéo, si des modifications sont susceptibles d'y être apportées.

Comme nous l'avons expliqué au chapitre 4, le mode vidéo en cours peut être lu facilement avec la fonction 00h de l'interruption vidéo 16h du BIOS. S'il apparaît alors que l'écran se trouve en mode texte (ces modes portent les numéros 0, 1, 2, 3 et 7), le programme résident doit seulement sauvegarder les 4000 premiers octets de la mémoire d'écran. Pour cela il peut encore recourir au BIOS vidéo (voyez le chapitre 4) ou accéder directement à la mémoire d'écran (les techniques appropriées sont décrites au chapitre 4).

Si le test du mode vidéo indique par contre que c'est un mode graphique qui est activé, la sauvegarde du mode vidéo est nettement plus compliquée, ne serait-ce que parce que la mémoire d'écran peut atteindre, sur certaines cartes EGA et VGA et dans certains

modes, une taille allant jusqu'à 256 Ko. Or si le programme résident a interrompu le travail d'un programme transitoire, il ne srea guère possible d'allouer un buffer d'une telle taille.

C'est pourquoi beaucoup de programmes résidents renoncent carrément à s'activer en mode graphique. Ils n'autorisent leur lancement que dans le cadre des différents modes texte. Comme sur le PC on travaille généralement en mode texte, cet inconvénient est normalement sans grande portée. Font toutefois exception à cet égard les environnements utilisateur comme GEM et Windows, qui travaillent exclusivement en mode graphique. Ces logiciels disposent d'ailleurs de leurs propres mécanismes pour permettre une exécution parallèle d'autres programmes ou pour intégrer des calculatrices, des blocs-notes etc., de sorte qu'il n'est de toute façon pas très intéressant d'utiliser les programmes résidents en liaison avec ces environnements utilisateur.

32.2. Les programmes résidents en Pascal et C

Avec les réflexions précédentes, nous avons défini la base sur laquelle reposera le travail des deux interfaces en assembleur que nous allons maintenant vous présenter (TSRPA.ASM pour les programmes en Pascal et TSRCA.ASM pour les programmes en C). L'un et l'autre de ces deux programmes fonctionnent selon les mêmes principes, ils ne se distinguent qu'à cause des différences de structure existant entre les programmes compilés en C ou en Pascal. Nous pouvons donc examiner tout d'abord les points communs entre les deux programmes.

Les deux interfaces considèrent que le programme résident est à installer lorsqu'on l'appelle pour la première fois et à désinstaller lors de tout nouvel appel.

Par ailleurs il est possible de rentrer à nouveau en contact avec le programme résident en l'appelant à partir de la ligne de commande, par exemple pour changer la touche d'activation sans être obligé de procéder à une réinstallation. Mais on peut aussi exploiter d'autres possibilités, car n'importe quelle routine en C ou en Pascal peut être appelée à partir du programme résident comme nous le verrons dans un instant.

32.2.1. Les fonctions de l'interface en assembleur

Pour faire fonctionner les mécanismes cités, l'interface en assembleur met à la disposition du programme en langage évolué sept routines présentées par le tableau suivant :

Lecture de l'état d'installation	
Nom	**Rôle**
TsrInit	Transforme le programme en programme résident, installe le gestionnaire d'interruption, termine le programme en le laissant en mémoire

Lecture de l'état d'installation	
Nom	**Rôle**
TsrIsInst	Teste si un exemplaire du programme résident est déjà présent en mémoire centrale
TsrCanUninst	Teste si l'exemplaire installé peut être désinstallé
TsrUnInst	Procède à la désinstallation de l'exemplaire résident en mémoire centrale
TsrSetptr	Mémorise l'adresse de la routine qui doit être appelée dans l'exemplaire déjà installé du programme
TsrCall	Déclenche la routine fixée par TsrSetptr
TsrSetHotKey	Fixe la touche d'activation du programme

Parmi toutes ces routines, le programme en langage évolué doit d'abord appeler TsrInInst qui permet de savoir s'il existe déjà un représentant du programme résident en mémoire centrale. Cette routine se sert à cet effet de l'interruption 2Fh du multiplexeur DOS (MUX) car au moment de l'installation du programme résident un gestionnaire de l'interruption 2Fh a été aménagé.

Mais ce gestionnaire ne réagit qu'à une fonction particulière dont le numéro est fixé à l'appel de TsrIsInst. Si l'interruption est appelée avec un autre numéro de fonction, le programme résident transmet simplement l'appel à l'ancien gestionaire. Le multiplexeur supporte deux options pour la fonction désignée par son numéro : elles portent elles-mêmes les numéros AAh et BBh. La première sert à rechercher un exemplaire existant du programme résident.

Comme il est d'usage pour les fonctions du multiplexeur, les numéros de fonction et d'option sont soumis à une permutation dans les registres AH/AL. Si le programme résident n'est pas installé, la permutation n'a pas lieu, puisqu'il n'existe pas de gestionnaire MUX qui se sente concerné par la fonction indiquée. Le registre AX garde donc sa valeur.

En appelant cette fonction MUX, TsrIsInst peut très facilement détecter si le programme résident a déjà été installé. Si tel est le cas, TsrIsInst appelle dans la foulée la deuxième fonction MUX du programme résident qui retourne le segment mémoire où se trouve le programme résident. Cette valeur est mémorisée dans une variable du programme résident. Comme toutes les autres variables du module en assembleur, elle est hébergée dans le segment de code. C'est ainsi qu'elles peuvent être accédées de l'intérieur du gestionnaire d'interruption même quand le segment de données du programme est hors de portée.

Fixer de la touche d'activation et installation

Si à l'aide de TsrIsInst le programme détecte qu'il n'est pas encore installé, il va procéder en règle générale à son installation. L'opération se fait en deux étapes. Il faut d'abord choisir la touche d'activation par TsrSetHotKey et ensuite ancrer le programme en mémoire par TsrInit.

Mais restons en pour le moment à TsrSetHotkey. Il faut communiquer à cette routine les deux paramètres qui déterminent la touche d'activation : le masque binaire de la touche de commande et le scan code de la lettre ou du chiffre associés. Ces paramètres peuvent être fournis sous forme de constantes qui se trouvent en tête des programmes en Pascal ou en C.

Les touches de commande portent les noms LSHIFT (majuscule gauche), RSHIFT (majuscule droite), ALT, CTRL, etc... Plusieurs de ces constantes peuvent être combinées par un opérateur OU si vous voulez que l'activation se fasse par plusieurs touches de commande actionnées simultanément. Sous cet angle un OU binaire correspond à un ET logique pour l'utilisateur.

Les différentes constantes définissant les scan codes des touches commencent par le préfixe SC_ suivi du nom de la touche (par exemple SC_5, SC_X, ou SC_SPACE). Ces constantes sont faciles à reconnaître car elles forment un bloc important dans les programmes en langage évolué. Si votre touche d'activation se limite à une ou plusieurs touches de commande sans aucun caractère additionnel, vous pouvez utiliser la constante SC_NOKEY.

Une fois la touche d'activation fixée, le programme est transformé en un programme résident par l'action de TsrInit. Il faut communiquer deux paramètres à TsrInit : l'offset de la routine proprement dite et une information sur la taille mémoire nécessaire. Cette dernière n'est pas traitée de la même façon dans les versions destinée aux programmes en C et en Pascal : c'est pourquoi nous reviendrons sur ce point un peu plus tard.

Retenons pour l'instant que le premier paramètre fourni à TsrInit est l'offset de la routine en langage évolué qui est déclenchée lors de l'activation du programme résident. Il s'agit bien de la routine qui représente la fonctionnalité même du programme résident et qui peut ainsi exploiter toutes les possibilités du langage utilisé. Tout y esy permis : accès aux fichiers, lecture de catalogues, actions diverses et variées impliquant des fonctions de DOS. Lorsque cette routine s'achève, le programme résident se replace à l'arrière-plan et laisse le champ libre au programme interrompu.

La fonction de TsrInit consiste tout d'abord à rechercher et à sauvegarder les adresses des gestionnaires des interruptions 08h, 09h, 13h, 28h et 2Fh. TsrInit doit ensuite prendre connaissance des données appartenant au contexte du programme en langage évolué. Ces données sont également sauvegardées dans des variables à l'intérieur du segment de code de façon à rester accessibles ultérieurement aux gestionnaires d'interruption et à l'action du programme résident. L'étape suivante consiste à installer

les gestionnaires d'interruption pour les interruptions 08h, 09h, 13h 28h et 2Fh. Le numéro de fonction communiqué au multiplexeur 2Fh est celui indiqué précédemment lors de l'appel à TsrInit.

Avant que le programme puisse être ancré en mémoire comme programme résident, il faut calculer sa taille c'est-à-dire le nombre de paragraphes qui resteront occupés à l'issue de l'instalation. Comem nous l'avons déjà dit, les interfaces en C et en Pascal diffèrent sensiblement sur ce point. Vous trouverez donc les explications nécessaires sur ce calcul dans la description de chacune des interfaces, dans les pages suivantes.

L'installation proprement dite est alors terminée et le programme est laissé à lui-même comme programme résident. Notez bien à ce propos que la routine d'installation ne retourne pas au programme en langage évolué et que toutes les tâches d'initialisation telles que la réservation de mémoire ou l'initialisation de variables doivent donc être terminées avant que cette routine ne soit appelée.

Désinstallation

Les programmes résidents appliquent souvent le principe suivant : au premier appel ils s'installent, au second ils se retirent à nouveau de la mémoire. Si TsrIsInst détecte la présence en mémoire d'un exemplaire du programme, il faudra généralement désinstaller cet exemplaire.

Pour cela il faut d'abord appeler la fonction TsrCanUninst qui vérifie si la désinstallation est possible. Il existe en effet des cas d'impossibilité, lorsque l'installation du programme résident a été suivie par une autre installation, le nouveau programme résident détournant également les vecteurs d'interruption du timer, du clavier, etc...Ce faisant, il enlève toute possibilité d'accès à l'ancien gestionnaire du programme à désinstaller sans que ce dernier soit prévenu de sa disparition en mémoire. le résultat est un plantage du système.

TsrCanUninst prend la précaution de tester si toutes les interruptions détournées renvoient bien sur les gestionnaires des interruptions de l'exemplaire déjà installé du programme résident. Il répond True ou False. Seule la valeur True ouvre la voie à l'appel de TsrUninst pour désinstaller le programme.

Les anciens gestionnaires des interruptions 08h, 09h 13h, 28h et 2Fh sont rétablis. La mémoire occupée par le programme est ensuite libérée pour que DOS puisse l'allouer à d'autres programmes. Le programme désinstallé ne laisse ainsi aucune trace en mémoire.

Appel de routines à l'intérieur du programme résident

C'est au plus tard au moment de la désinstallation d'un programme résident qu'il faudra faire appel à des routines depuis l'intérieur du programme. Car la partie en langage évolué d'un programme résident consommera souvent des ressources d'exploitation (mémoire, vecteurs d'interruption, fichiers) qu'il faudra restituer lors de la désinstallation.

Comme les adresses de segment de l'exemplaire installé peuvent être lues facilement par l'intermédiaire du multiplexeur et comme les offsets sont les mêmes que pour l'exemplaire en cours d'exécution, il n'est pas difficile de réaliser un pointeur qui référence une routine à appeler à l'intérieur du programme résident installé. Il faut simplement supposer que la routine en question est du type FAR.

Il apparaît cependant une difficulté car comme on appelle directement la routine, aucun changement de contexte ne se produit en direction de l'exemplaire installé. Le segment de données de l'exemplaire en cours d'exécution restera actif, tout comme ses PSP et DTA. La routine appelée ne peut donc pas accéder à ses variables du segment de données, ce sont les variables de l'exemplaire en cours d'exécution qui sont exploitées.

Il faut donc mettre en place un moyen qui effectue un changement de contexte avant l'appel de la fonction puis rétablit le contexte à l'issue de l'appel.

La routine en question ne nécessiterait comme argument que le seul offset de la routine à exécuter dans l'exemplaire installé du programme résident. Ce mécanisme fonctionne mais il n'est pas particulièrement confortable car aucun argument ne peut être transmis à la routine appelée et inversement aucune valeur de retour n'est exploitable.

Pour rendre possible la transmission d'arguments et le retour d'un résultat, on a choisi une autre voie qui se compose de deux routines de l'interface en assembleur. Ces routines s'appellent TsrSetPtr et TsrCall

TsrSetPtr va intervenir en premier pour recueillir l'adresse de la routine à appeler et la mémoriser dans une variable de l'interface en assembleur. Puis TsrCall exécute le changement de contexte et se sert de l'adresse mémorisée précédemment pour déclencher la routine.

Le seul problème est de déclarer TsrCall dans le module en langage évolué avec la liste des arguments transmis à la routine appelée. Mais le nombre et le type de ces arguments varie selon la nature de la routine. Mais ce problème se résoud également comme on le verra dans la description de la partie en langage évolué.

32.2.2. Les gestionnaires d'interruption

Les gestionnaires d'interruption de l'interface en assembleur fonctionnent selon le principe décrit plus haut. Au centre se trouve le gestionnaire de l'interruption 09h du

clavier. La collaboration avec les gestionnaires des interruptions 08h (timer), 13h (disque BIOS) et 28h (Idle DOS) se fait par trois indicateurs situés dans le segment de code de l'interface en assembleur : in_bios, tsractif et tsrnow.

Les indicateurs

L'indicateur TsrActif signale si le programme résident est actuellement actif. Il ne peut prendre que les états 0 ou 1. L'indicateur INBIOS est incrémenté lorsque le gestionnaire INT 13h se déclenche et décrémenté à son issue. Partant de la valeur 0 comme tous les autres indicateurs, il présente donc la valeur 1 en cours d'exécution de INT 13h, et 0 dans toute autre circonstance. En fait INBIOS peut avoir une valeur supérieure à 1 lorsqu'une fonction de l'interruption 13h effectue un appel récursif à cette même interruption. Pour nous il sera important de savoir que l'indicateur est nul lorsqu'aucune fonction disque du BIOS n'est en cours d'exécution, ce qui ouvre la voie à l'activation du programme résident.

Le troisième indicateur du groupe mentionné est TsrNow. Il est fixé à l'intérieur du gestionnaire d'interruption du clavier lorsque la touche d'activation a été frappée mais que le déclenchement du progarmme résident n'est pas possible pour l'une des raisons déjà décrites. Les gestionnaires de l'interruption du timer et de l'interruption Idle de DOS se servent de cet indicateur pour savoir si le programme résident est en attente de déclenchement.

Le gestionnaire d'interruption du timer décrémente l'indicateur tsrnow à chaque appel de façon qu'à l'issue d'un certain délai il atteigne la valeur 0, ce qui interdira le déclenchement du programme résident parce qu'il sera trop tard.

Après la théorie, jetons un coup d'oeil sur le code des différents gestionnaires d'interruption. Nous y verrons quelques détails fort intéressants pour la programmation de ce type de routine. Ce sont surtout les interruptions 08h (timer) et 09h (clavier) que nous examinerons. Les explications valent pour les deux versions, qu'elles soient écrites en C ou en Pascal.

Le gestionnaire d'interruption du timer

L'extrait de listing suivant montre la partie de l'interface en assembleur consacrée au nouveau gestionnaire d'interruption du timer 08h.

L'indicateur tsrnow est testé dès le début de la routine. Notez l'absence d'indication explicite du segment (par exemple cmp cs:tsrnow,0) car la variable en question se trouve dans le segment de code. Le segment n'a pas besoin d'être précisé dans le programme source car l'assembleur en tient compte automatiquement dans la génération du code machine. Une directive ASSUME a précisé antérieurement que le registre de segment

CS référençait le segement de code et que le contenu de DS et des autres registres n'était pas défini à l'avance :

```
assume cs:code, ds:nothing, es:nothing, ss:nothing
```

Tous les gestionnaires d'interruption pourront ainsi accéder directement aux différentes variables de l'interface en assembleur car elles sont hébergées dans le segement de code, ce qui évite de charger à chaque fois le segment de données du programme correspondant en langage évolué.

Mais revenons à l'interruption du timer. Si le gestionnaire trouve que TsrNow est nul, c'est-à-dire que le programme résident n'attend pas son déclenchement immédiat, il se branche à l'étiquette i8_end où se trouve l'appel à l'ancienne version du gestionnaire. L'instruction correspondante s'écrit :

```
jmp [int8_ptr]
```

int8_ptr n'est pas une adresse de branchement mais une variable qui a mémorisé l'adresse de l'ancien gestionnaire sous l'effet de la routine TsrInst lors de l'installation du programme résident. Comme int8_ptr est une variable de type DWORD, l'assembleur sait qu'il doit exécuter un branchement long (FAR JMP) et non pas un saut de proximité. Il est clair en effet que l'ancien gestionnaire se trouve dans un tout autre segment de code que l'actuel (qui est celui du programme résident).

Le branchement vers l'ancien gestionnaire termine l'exécution du nouveau gestionnaire car l'instruction IRET située à la fin de l'ancien gestionnaire rend la main au programme interrompu par l'appel du timer.

```
;-- Nouveau gestionnaire de l'interruption 8h (Timer)------------------

int08       proc far

            cmp  tsrnow,0          ;Faut-il activer le programme résident
            je   i8_end               ;Non, passe à l'ancien gestionnaire

            dec  tsrnow         ;Oui, décrémente l'indicateur d'activation

            ;-- TSR doit être activé mais est-ce possible ? -------------

            cmp  in_bios, 0         ;Interruption disque du BIOS active ?
            jne  i8_end                ;OUI --> pas d'activation possible

            call dosactif                  ;DOS peut-il être interrompu ?
            je   i8_tsr                ;Oui, appelle le programme résident

i8_end:     jmp  [int8_ptr]              ;retourne à l'ancien gestionnaire

            ;-- Active le programme résident ---------------------------

i8_tsr:     mov  tsrnow,0             ;Le TSR n'attend plus son activation
            mov  tsractif,1   ;le programme résident est maintenant actif
            pushf                ;Simule l'appel de l'ancien gestionnaire
```

```
        call [int8_ptr]                          ; par INT 8h
        call start_tsr                  ;Lance le programme résident
        iret                          ;retourne au programme interrompu

int08   endp
```

Mais que se passe-t-il si l'indicateur tsrnow n'est pas nul, c'est-à-dire si le programme résident est en différé d'activation ? Dans ce cas l'indicateur est décrémenté dans l'espoir qu'il devienne nul et puisse ainsi au prochain appel du timer conduire à l'activation effective du programme résident.

Supposons que le programme résident doive être activé. Il faut d'autres tests pour en démontrer la possibilité. Le premier porte sur IN_BIOS. Si cet indicateur n'est pas nul, l'interruption du timer est survenue pendant une opération sur disque gérée par le BIOS. Le contrôle est alors transmis à l'ancien gestionnaire d'interruption du timer. Sinon c'est au tour de l'indicateur INDOS d'être testé. Rappelons qu'il sert à détecter si une fonction de DOS a été interrompue.

Le test est géré par la routine en assembleur dosactif . Elle lit l'adresse de l'indicateur INDOS dans une variable de l'interface en assembleur et compare le contenu mémorisé à cette adresse avec la valeur 0. Si DOS ne peut pas être interrompu (INDOS:1), l'ancien gestionnaire est appelé et le programme résident n'est pas activé.

Sinon plus rien ne s'oppose au déclenchement de notre programme résident. Il faut naturellement exécuter au préalable l'ancien gestionnaire d'interruption du timer car il effectue des tâches importantes se rapportant à l'avancement de l'horloge et à la gestion des lecteurs de disquette. Par ailleurs ce même gestionnaire contient une instruction machine fort importante pour gérer les interruptions matérielles et qui s'énonce :

```
out 20h,20h
```

Cette instruction indique au contrôleur d'interruption que l'exécution de l'interruption est achevée. Tant qu'il ne la réceptionne pas, le contrôleur d'interruption ne genère plus d'interruption. Donc les appels au timer ne s'effectuent plus, l'horloge du PC n'avance plus , il est même impossible de faire des saisies au clavier puisque l'interruption 09h du clavier ne marche plus.

Pour appeler l'ancien gestionnaire, on se sert de l'adresse stockée en int8_ptr mais avec une instruction CALL pour que le processeur en rencontrant un IRET dans l'ancien gestionnaire revienne au nouveau gestionnaire du programme résident.

Ce retour ne s'effectuerait d'ailleurs pas correctement si le contenu du registre des indicateurs n'était pas empilé par PUSHF préalablement à l'instruction CALL. Cette dernière instruction en effet ne dépose sur la pile que l'adresse de retour, sans ajouter le registre des indicateurs comme le fait une interruption. En conséquence, si on ne prend pas de précaution, l'instruction IRET prend comme contenu du registre des indicateurs une partie de l'adresse de retour et comme adresse de retour des informations toutes autres de la pile, ce qui plante évidemment le système.

Bien que le programme résident ne soit lancé qu'après l'appel de l'ancien gestionnaire, il faut déjà mettre à 0 l'indicateur TsrNow et à 1 l'indicateur TsrActif. On fait donc comme si le programme était déjà déclenché. En effet l'ancien gestionnaire informe le contrôleur d'interruption que l'interruption est terminée (cf plus haut). Donc pendant l'exécution du nouveau gestionnaire d'interruption du timer d'autres interruptions peuvent survenir, et notamment des interruptions clavier parce que l'utilisateur a frappé une touche.

Supposons que cette touche soit par malheur la touche d'activation. Le programme résident serait activé à partir du gestionnaire d'interruption du clavier si on n'avait pris la précaution de mettre à 1 l'indicateur tsractif. Le nouveau gestionnaire d'interruption du timer ne pourrait se poursuivre qu'au terme de l'exécution du programme résident.

Mais ce risque n'existe pas si on positionne correctement les indicateurs avant même d'appeler l'ancien gestionnaire : le programme résident ne peut plus être invoqué par les autres gestionnaires d'interruption.

Le programme résident se déclenche en fait par Start_Tsr, une routine de l'interface en assembleur qui commence par sauvegarder le contexte d'écran du programme interrompu. Cette routine établit ensuite le contexte du programme résident et appelle la partie résidente qui a été installée par TsrInit. A l'issue du programme résident, le contexte du programme interrompu sera rétabli.

Ce n'est qu'à ce moment que l'on revient à l'exécution du nouveau gestionnaire d'interruption du timer, qui rend la main par une instruction IRET au programme interrompu à l'origine. Il est remarquable de penser que le nouveau gestionnaire du timer est sans cesse invoqué pendant l'exécution du programme résident alors même que son exécution n'est pas achevée. Mais peu importe, pourvu que l'adresse du programme interrompu reste sur la pile au retour de Start_Tsr. Du point de vue du système, l'exécution du nouveau gestionnaire du timer est considérée comme achevée à la réception de l'instruction out 20h,20h.

Le gestionnaire d'interruption du clavier

Beaucoup d'observations faites à propos du gestionnaire d'interruption du timer sont aussi valables pour le gestionnaire d'interruption du clavier relié à l'interruption 09h. On commence par y sauvegarder sur la pile le contenu de AX qui sera modifié. D'une manière générale tous les gestionnaires d'interruption doivent restituer les registres dans l'état où ils les ont trouvés. Pour le gestionnaire d'interruption du timer, cette règle ne posait pas de problème car il ne modifie aucun registre.

L'étape suivante consiste à lire le contenu du port 60h. C'est le port où le contrôleur du clavier dépose le scan code de la touche tapée et c'est ce scan code qui devra être examiné pour détecter la touche d'activation.

Mais le nouveau gestionnaire du clavier teste d'abord si le programme résident est déjà actif. Si tel est le cas, les autres tests sont inutiles et un branchement vers i9_end passe le contrôle à l'ancien gestionnaire, après restauration de AX.

Les mêmes événements se produisent si TsrActif est nul sans que TsrNow le soit. Cette configuration signifie que la touche d'activation a été tapée mais que le programme résident est en instance de déclenchement. A lors à quoi bon retester la touche d'activation ?

Ce test reste cependant nécessaire si le programme résident n'est ni actif ni en instance de déclenchement. Le scan code est comparé à la variable sc_code. Si elle contient la valeur 128, c'est qu'il n'existe ps à proprement parler de touche d'activation, il faut alors tester l'état des touches de commande à l'adresse de branchement i9_ks.

```
;-- Nouveau gestionnaire de l'interruption 09h (clavier)---------------

int09      proc far

           push ax
           in   al,60h                              ;Lit le port du clavier

           cmp  tsractif,0    ;Le programme résident est-il déjà actif ?
           jne  i9_end   ;OUI: appelle l'ancien gestionnaire puis retour

           cmp  tsrnow,0  ;Le programme est-il en attente d'activation ?
           jne  i9_end   ;OUI: appelle l'ancien gestionnaire puis retour

           ;-- Teste la touche d'activation   -------------------------

           cmp  sc_code,128                        ;Y a-t-il un scan code ?
           je   i9_ks         ;Non, ne teste que les touches de commande

           cmp  al,128                      ;Oui est-ce un code release ?
           jae i9_end                             ;Oui, pas d'activation

           cmp  sc_code,al           ;Code make à comparer avec le modèle
           jne  i9_end                  ;Pas d'activation si pas le même

i9_ks:     ;-- Teste l'état des touches de commande --------------------

           push ds
           mov  ax,040h                  ;DS sur le segment des variables
           mov  ds,ax                                          ; du BIOS
           mov  ax,word ptr ds:[17h]  ;lit indic. d'état clavier en BIOS
           and  ax,key_mask              ;bits de la touche d'activation
           cmp  ax,key_mask            ;bits de la touche d'activation ?
           pop  ds
           jne  i9_end    ;Touche d'activation détectée ? NON --> retour

           cmp  in_bios, 0       ;Interruption disque du BIOS en cours ?
           jne  i9_el                 ;OUI --> pas d'activation possible

           call dosactif             ;Est-il possible d'interrompre DOS ?
           je   i9_tsr                  ;Oui, lance le programme résident
```

```
i9_e1:      mov  tsrnow,TIME_OUT            ;TSR en attente d'activation

i9_end:     pop   ax                                          ;Reprend AX
            jmp   [int9_ptr]        ;Se branche sur l'ancien gestionnaire

i9_tsr:     mov   tsractif,1       ;Le TSR va être actif (dans un moment)
            mov   tsrnow,0                    ;Pas de délai de lancement
            pushf
            call  [int9_ptr]              ;Appelle l'ancien gestionnaire
            pop   ax                                        ;Récupère AX
            call  start_tsr                 ;Lance le programme résident
            iret                          ;retourne au programme interrompu

int09       endp
```

S'il existe une vraie touche d'activation (sc_code différent de 128) le scan code livré par le port 60h est examiné. S'il est supérieur à 128, il s'agit d'un code Release qui signale le relâchement d'une touche et non son enfoncement. Dans ce cas le programme se branche sans tarder sur l'ancien gestionnaire, car ce qui nous intéresse ici c'est de détecter une pression sur la touche.

Si une touche est effectivement enfoncée, son code est comparé à sc_code. S'il n'y a pas identité, il ne s'agit pas de la touche d'activation et le contrôle est rendu à l'ancien gestionnaire. Mais si le code correspond, il faut tester l'état des touches de commande.

Nous voilà donc revenus à l'étiquette i9_ks déjà mentionnée précédemment. L'état courant des touches de commande lu à l'adresse 0040h:0017h du BIOS est comparé à l'état de référence stocké dans la variable key_mask. S'il n'y a pas de correspondance, on retourne à l'ancien gestionnaire du clavier qui est chargé de la suite du traitement.

Mais si les touches de commandes attendues ont été activées, c'est le signal du déclenchement du programme résident. Auparavant il faut cependant tester si aucune fonction disque du BIOS ou aucune fonction de l'API de DOS n'est en cours. Si cet obstacle existe, le programme résident n'est pas activé mais le contrôle est rendu à l'ancien gestionnaire du clavier. Mais on fixe auparavant à l'indicateur tsrnow la valeur représentée par la constante TIME_OUT. Ainsi aux prochains appels du timer, le programme aura une nouvelle possibilité de se déclencher.

TIME_OUT possède a priori la valeur 9 qui correspond à une demi seconde (le timer est appelé 18,2 fois par seconde). Ainsi vous pouvez y reporter toute autre valeur à votre convenance.

Si rien du côté du BIOS ou de DOS ne s'oppose au déclenchement du programme résident, il faut d'abord appeler l'ancien gestionnaire du clavier (tout comme dans le cas du timer nous avions invoqué son ancienne version). Les deux indicateurs TsrNow et TsrActif sont initialisés de façon qu'aucun autre gestionnaire d'interruption n'ait la malencontreuse idée de déclencher le programme résident. Ce déclenchement se fait en effet tout seul à l'intérieur du gestionnaire de clavier.

32.2.3. Les programmes en langage évolué

Après toutes ces considérations sur les modules en assembleur, je voudrais attirer votre attention sur les programmes en langage évolué TSRP.PAS et TSRC.C qui montrent leur interfaçage. Les deux programmes ont la même structure et ne diffèrent que sur des points de détail auxquels nous nous consacrerons un peu plus tard. Pour l'instant je parlerai des points communs.

Après leur lancement, les programmes exploitent les paramètres de la ligne de commande avec la fonction ParmGetHotKey. Les touches d'activation sont introduites par le préfixe "/t". Ce préfixe doit être suivi du nom d'une touche de commande (lshift, rshift, alt, ctrl, etc...) ou d'un numéro de scan code sous forme décimale.

Par exemple pour définir <Majuscule gauche> et <Espace> comme touche d'activation, il faut mettre comme paramètres :

```
/tlshift /t57
```

Si on y ajoute la touche <Alt>, l'ensemble devient :

```
/tlshift /talt /t57
```

L'ordre des paramètres est sans importance.

Comme résultat, ParamGetHotKey reporte l'état attendu des touches de commande et le scan code de la touche d'activation dans les deux variables keymask et ScCode qui lui sont transmises à cet effet.

Si une erreur est détectée à l'exploitation de la ligne de commande, l'exécution du programme est arrêtée avec émission d'un message approprié. Sinon la fonction TsrIsInst du module en assembleur teste si le programme est déjà installé. En même temps son code de reconnaissance par le multiplexeur est fixé. La constante I2F_CODE est a priori égale à C4h mais vous pouvez choisir un autre code. Ce sera nécessaire si vous développez plusieurs programmes résidents avec le module en assembleur. Faute de cette précaution, leur code serait le même et il y aurait confusion au niveau du multiplexage.

Si TsrIsInst indique que le programme n'est pas encore installé, les variables Keymask et ScCode sont examinées. Si aucun paramètre /t n'a été fourni sur la ligne de commande, ces variables reçoivent des valeurs par défaut et la routine TsrSetHotKey décide de prendre comme touche d'activation <Alt><H>. Sinon la même routine donne la valeur prévue à la touche d'activation.

Il reste alors à appeler la routine TsrInit qui rend le programme résident. La procédure résidente communiquée est le programme de haut niveau mais nous verrons cela un peu plus loin.

Si l'appel de TsrInit a révélé que le programme était déjà installé, la suite des opérations dépend de l'utilisateur. Si au lancement du programme il a indiqué une touche d'activation, celle-ci est répercutée par TsrSetHotkey sur l'exemplaire déjà installé du programme et il n'y a pas de désinstallation. Si aucune nouvelle touche d'activation n'a été communiquée, TsrCanUninst teste si l'exemplaire déjà installé peut être retiré de la mémoire. Si tel est le cas, la procédure TsrUninst se charge de la désinstallation.

Auparavant une routine en langage évolué est appelée à l'intérieur de l'exemplaire installé pour libérer les ressources internes et afficher un message. Cette routine s'appelle endfct (en C) ou Endprc (en Pascal).

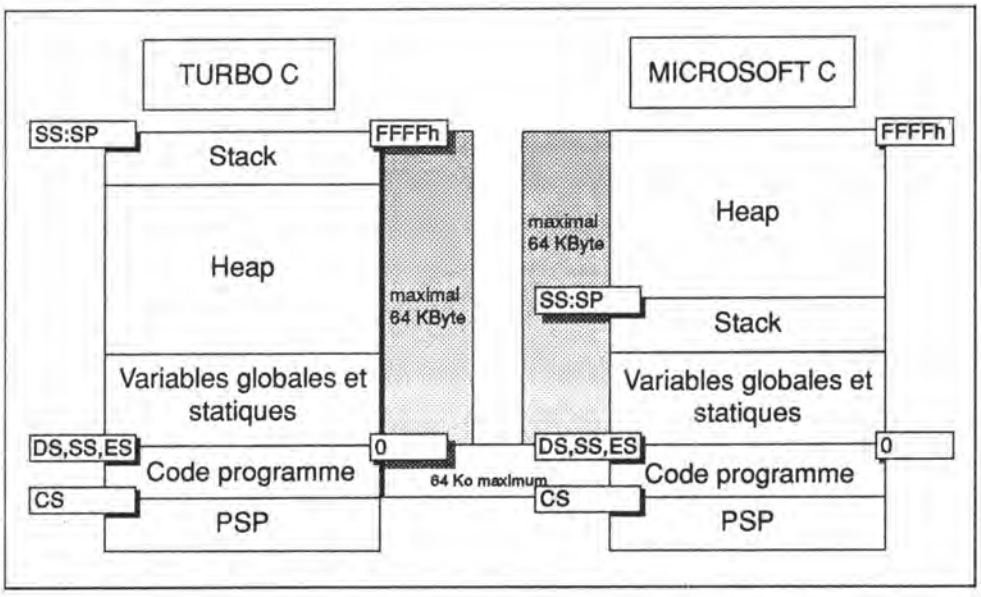
Elle prend note du nombre d'activations du programme résident tenu à jour dans la variable globale ATImes. Chaque fois que le programme résident est déclenché, cette variable est incrémentée par la procédure Tsr. Le buffer du clavier est vidé pour retirer la touche d'activation. Puis l'écran du programme interrompu est sauvegardé et l'utilisateur est invité à taper une touche quelconque.

Aussitôt l'écran du programme interrompu est restauré, la procédure résidente est achevée et le programme interrompu reprend son exécution.

Implémentation en C

Par nature un programme résident doit occuper le moins de mémoire possible. C'est pourquoi l'interface en assembleur suppose qu'il a été développé dans le modèle de mémoire SMALL. En modèle SMALL le compilateur de Microsoft et Turbo C disposent le code et les données dans deux segments distincts qui ne peuvent pas dépasser 64 Ko. Dans le segment de données se trouvent non seulement les données globales et statiques mais aussi la pile et le tas.

Comme le montre la figure suivante, les deux compilateurs adoptent des organisations différentes. Dans Turbo C, la pile est disposée derrière le tas et s'étend de la fin du segment de données jusqu'à l'extrémité du tas. Dans MSC la pile se trouve entre les données globales et le tas. Sa taille est donc prédéterminée dans le cas de MSC alors que dans le cas de Turbo C elle varie et dépend de l'espace pris par le tas.



Structure d'un programme en modèle de mémoire SMALL dans Turbo C et dans MSC

Cette structure n'aurait aucune influence sur l'interface en assembleur si à l'installation du programme nous étions prêts à laisser en mémoire non seulement le code mais aussi les 64 Ko du segment de données. Mais il s'agirait là d'une attitude de gaspillage et comme nous l'avons dit les programmes résidents doivent être économes en mémoire. L'interface en assembleur ne laisse donc en mémoire que la partie du segment des données qui est vraiment nécessaire.

La taille de cet espace dépend de la taille des données (ou plus précisément des objets) qui ont été alloués sur le tas par les fonctions CALLOC et MALLOC. Cette taille doit faire l'objet d'une estimation de votre part et être communiquée comme troisième et dernier paramètre à TsrInit. Par rapport à la fin du tas occupé jusqu'à présent, on maintient libre le nombre d'octets indiqué qui est ancré en mémoire centrale avec le programme résident.

Ce procédé vous permet de travailler normalement avec les fonctions de manipulation du tas à l'intérieur du programme résident. Mais à vrai dire la permission n'est valable sans restriction que pour le compilateur Turbo C. Comme MSC pour allouer de la mémoire utilise un algorithme qui suppose que la mémoire disponible s'étend jusqu'à la fin du segment de données, il vaut mieux renoncer à ce type d'allocation à l'intérieur du programme résident. Il est préférable d'allouer les buffers et variables nécessaires avant d'appeler TsrInit ou de définir les objets comme variables globales.

Le programme d'exemple TSRC.C suit ce principe en allouant les deux buffers nécessaires à l'intérieur de la fonction MAIN et en déposant leurs adresses dans des variables globales.

Si on utilise Turbo C il faut noter une autre particularité. Comme la pile descend de la fin du segment de données de 64 Ko vers le tas, elle se trouve, une fois le segment de données partiellement libéré, en dehors du programme dans une zone de mémoire que DOS peut attribuer à d'autres programmes. Pour que ceci ne pose pas problème, l'interface en assembleur déplace la pile juste à la suite du tas avec une taille fixe de 512 octets donnée par la constante TC_STACK du module TSRCA.ASM.

Dans la plupart des applications cette taille est suffisante mais si vous implantez des gros objets (par exemple des tableaux) comme variables locales, ou si vous les communiquez comme arguments à des fonctions, vous risquez d'avoir des problèmes. Dans ce cas n'hésitez pas à augmenter la taille en modifiant en conséquence la constante TC_STACK.

Le traitement différencié de la pile explique pourquoi il faut indiquer à TsrInit si le programme a été développé avec le compilateur de Microsoft ou avec Turbo C. En pratique il n'y a pas lieu de s'en soucier car à l'intérieur du programme en C ficure une constante appropriée traitée par le préprocesseur.

Le même type de directive conditionnelle se trouve dans la fonction GetHeapEnd qui est invoquée dans le module en assembleur à partir de TsrInit. Cette fonction fournit un pointeur FAR référençant l'extrémité du tas occupé. Avec les compilateurs de Borland la même information peut être obtenue par la fonction de bibliothèque SBRK, également disponible jusqu'à la version MSC 6.0 et QuickC 2.5. Lorsque cette fonction n'est plus supportée le tas doit être parcouru par _heapwalk pour détecter le dernier bloc occupé.

Dans la version en C du programme, il faut noter la manière dont les fonctions sont appelées dans l'exemplaire installé du programme résident. Comme nous l'avons déjà évoqué dans l'étude du module en assembleur, il faut d'abord invoquer TsrSetPtr qui recueille l'adresse de la routine à appeler et la transmet en vue d'un appel ultérieur de TsrCall.

Dans la version en C TsrSetPtr renvoie directement un pointeur sur TsrCall. On peut ainsi appeler la fonction souhaitée en exploitant directement le résultat de TsrSetPtr mais il est nécessaire de procéder à un transtypage pour que le compilateur y retrouve son latin. Examinons tout cela en détail.

Au début du module en C on définit deux types de pointeurs procéduraux appelés OAFP et SHKFP. OAFP pointe sur toute fonction qui n'attend aucun argument et ne renvoie aucun résultat. SHKFP a été conçu spécialement pour les besoins de la routine en assembleur TsrSetHotKey :

```
typedef void (*OAFP) (void);
typedef void (*SHKFP) (WORD Keymask, BYTE ScCode);
```

Pour appeler TsrSetHotKey par TsrSetPtr, il faut écrire :

```
(*(SHKFP) TsrSetPtr(TsrSetHottKey))(Keymask, ScCode);
```

On communique à TsrSetPtr l'adresse de la fonction TsrSetHotKey sous forme de pointeur FAR. En fait un pointeur NEAR suffirait puisque le segment sera plus tard celui de l'exemplaire déjà installé du programme. mais les fonctions ainsi appelées doivent être de type FAR sinn elles ne pourraient pas être appelées à partir d'un autre segment de code. Pour éviter l'avertissement émise par le compilateur lorsqu'on transforme un pointeur FAR en NEAR, TsrSetPtr prend l'adresse FAR de la fonction à appeler mais n'en utilise que l'offset.

TsrSetPtr retourne l'adresse de TsrCall comme un pointeur NEAR. Par transtypage on obtient un pointeur de type SHKFP. Ce n'est qu'à ce prix que les paramètres peuvent être indiqués pour que les compilateurs les dépose sur la pile sans maugréer. TsrCall sera appelé par référencement du pointeur ainsi transtypé.

Lorsqu'il intervient, TsrCall trouve sur la pile les arguments qui sont en faite destinée à la fonction à appeler? Ces arguments doivent être recopiés sur la pile de l'exemplaire installé du programme résident à cause du changement de pile qui va avoir lieu. La plupart des compilateurs C génèrent un code qui suppose que le registre DS pointe sur le même segment que SS. Si le changement de pile n'était pas effectué, cette hypothèse ne serait plus replie, et la fonction à appeler ne pourrait pas fonctionner normalement.

Pour appeler des fonctions à partir de l'exemplaire déjà installé du programem résident il faut donc suivre ces deux étapes :

① Définir un type de pointeur procédural sur une fonction qui attend les mêmes arguments que la vôtre

② Transmettre l'adresse de cette fonction à TsrSetPtr, transtyper le résultat de la fonction avec le pointeur défini précédemment et appeler la fonction avec les arguments souhaités

Listing : TSRC.C

```
/*****************************************************************
*                          T S R C                              *
*---------------------------------------------------------------*
*  Fonction       : Programme écrit en C transformable en programme *
*                   résident grâce à un module en assembleur    *
*---------------------------------------------------------------*
*  Auteur         : MICHAEL TISCHER                            *
*  Développé le   : 15.08.1988                                 *
*  Dernière MAJ   : 19.03.1992                                 *
*---------------------------------------------------------------*
*  Modèle mémoire : SMALL                                       *
*****************************************************************/

/*--- Fichiers d'en-tête ------------------------------------------*/

#include <stdlib.h>
#include <stdio.h>
#include <dos.h>
#include <conio.h>
#include <string.h>
#include <bios.h>

/*--- Typedef -----------------------------------------------------*/

typedef unsigned char BYTE;              /* Bricolage d'un type BYTE */
typedef unsigned int  WORD;
typedef BYTE          BOOL;              /* Comme BOOLEAN en Pascal */
typedef union vel far *VP;               /* VP : ptr FAR sur mémoire écran */

typedef void (*SAFP)(void);    /* Pointeur sur fonction sans argument */
typedef void (*SHKFP)( WORD KeyMask, BYTE ScCode );   /* TsrSetHotkey */

/*--- Macros ------------------------------------------------------*/

#ifndef MK_FP                            /* MK_FP pas encore défini  ? */
 #define MK_FP(s, o) ((void far *) (((unsigned long) (s)<<16)|(o)))
#endif
#define VOFS(x,y) ( 80 * ( y ) + ( x ) )
#define VPOS(x,y) (VP) ( vptr + VOFS( x, y ) )

/*--- Structures et unions ----------------------------------------*/

struct velb
       {                         /* Décrit une position d'écran en 2 octets */
         BYTE caractere,                                  /* Code ASCII */
              attribut;                          /* et attribut associé */
       };
```

```
struct velw
       {                              /* Décrit une position d'écran en 1 mot */
         WORD contenu;                /* Mémorise le code ASCII et l'attribut */
       };

union vel
      {                                      /* Décrit un position d'écran */
        struct velb h;
        struct velw x;
      };

/*-- Intègre les fonctions du module en assembleur ---------------------*/

extern void TsrInit( BOOL tc, void (*fct)(void), unsigned heap );
extern BOOL TsrIsInst( BYTE I2F_fctnr );
extern void TsrUnInst( void );
extern SAFP TsrSetPtr( void far *fct );
extern BOOL TsrCanUnInst( void );
extern void TsrCall( void );
extern void far TsrSetHotkey( WORD keymask, BYTE sccode );

/*-- Constantes et macros ----------------------------------------------*/

#ifdef __TURBOC__                       /* Compilation par TURBO-C ? */
  #include <alloc.h>
  #define TC TRUE                                          /* Oui */
  #define KeyAvail() ( bioskey(1) != 0 )
  #define GetKey() bioskey(0)
#else                                   /* Non on travaille avec Microsoft C */
  #include <malloc.h>
  #define TC FALSE
  #define KeyAvail() ( _bios_keybrd( _KEYBRD_READY ) != 0 )
  #define GetKey() _bios_keybrd( _KEYBRD_READ )
#endif

/*-- Scan codes de différentes touches ---------------------------------*/

#define SC_ESC              0x01
#define SC_1                0x02
#define SC_2                0x03
#define SC_3                0x04
#define SC_4                0x05
#define SC_5                0x06
#define SC_6                0x07
#define SC_7                0x08
#define SC_8                0x09
#define SC_9                0x0A
#define SC_0                0x0B
#define SC_SCHARFES_S       0x0C
#define SC_APOSTROPH        0x0D
#define SC_BACKSPACE        0x0E
#define SC_TAB              0x0F
#define SC_Q                0x10
#define SC_W                0x11
#define SC_E                0x12
#define SC_R                0x13
#define SC_T                0x14
#define SC_Z                0x15
#define SC_U                0x16
#define SC_I                0x17
#define SC_O                0x18
#define SC_P                0x19
#define SC_UE               0x1A
#define SC_PLUS             0x1B
#define SC_RETURN           0x1C
#define SC_CONTROL          0x1D
#define SC_A                0x1E
#define SC_S                0x1F
#define SC_D                0x20
#define SC_F                0x21
#define SC_G                0x22
#define SC_H                0x23
#define SC_J                0x24
#define SC_K                0x25
#define SC_L                0x26
#define SC_OE               0x27
#define SC_AE               0x28
#define SC_PLUSGRAND        0x29
#define SC_SHIFT_GAUCHE     0x2A
#define SC_FIS              0x2B
#define SC_Y                0x2C
#define SC_X                0x2D
#define SC_C                0x2E
#define SC_V                0x2F
#define SC_B                0x30
#define SC_N                0x31
#define SC_M                0x32
#define SC_VIRGULE          0x33
#define SC_POINT            0x34
#define SC_TIRET            0x35
#define SC_SHIFT_DROIT      0x36
#define SC_PRINT_SCREEN     0x37
#define SC_ALT              0x38
#define SC_SPACE            0x39
#define SC_CAPS             0x3A
#define SC_F1               0x3B
#define SC_F2               0x3C
#define SC_F3               0x3D
#define SC_F4               0x3E
#define SC_F5               0x3F
#define SC_F6               0x40
#define SC_F7               0x41
#define SC_F8               0x42
#define SC_F9               0x43
#define SC_F10              0x44
#define SC_NUM_LOCK         0x45
#define SC_SCROLL_LOCK      0x46
#define SC_CURSOR_HOME      0x47
#define SC_CURSOR_UP        0x48
#define SC_CURSOR_PG_UP     0x49
#define SC_NUM_MOINS        0x4A
#define SC_CURSOR_LEFT      0x4B
#define SC_NUM_5            0x4C
#define SC_CURSOR_RIGHT     0x4D
#define SC_NUM_PLUS         0x4E
#define SC_CURSOR_END       0x4F
#define SC_CURSOR_DOWN      0x50
#define SC_CURSOR_PG_DOWN   0x51
#define SC_INSERT           0x52
#define SC_DELETE           0x53
#define SC_SYS_REQUEST      0x54
#define SC_F11              0x57
#define SC_F12              0x58
#define SC_NOKEY            0x80        /* Pas de touche additionnelle */

/*-- touches de commande pour création masque de touche d'activation -*/

#define RSHIFT      1           /* Touche Majuscule Droite enfoncée */
#define LSHIFT      2           /* Touche Majuscule gauche enfoncée */
#define CTRL        4                 /* Touche CTRL enfoncée */
#define ALT         8                 /* Touche ALT enfoncée */
#define SYSREQ      1024        /* Touche SYS-REQ (sur clavier AT ) */
#define BREAK       4096              /* Touche Break enfoncée */
#define NUM         8192            /* Touche Num-Lock enfoncée */
#define CAPS        16384           /* Touche Caps-Lock enfoncée */
#define INSERT      32768             /* Touche INSERT enfoncée */

#define I2F_CODE    0xC4                /* Fonction numéro INT 2F */
#define I2F_FKT_0   0xAA          /* Code pour INT 2F, fonction 0 */
#define I2F_FKT_1   0xBB           /* Code für INT 2F, fonction 1 */

#define COULN       0x07                      /* Couleur normale */
#define INV         0x70                      /* Couleur inverse */
#define COULNC      0x0f                /* Couleur normale claire */
#define INVC        0xf0                /* Couleur inverse claire */

#define TAS_LIBRE   1024                    /* Laisse 1 Ko sur le tas */

#define TRUE  ( 0 == 0 )      /* Constantes pour travailler avec BOOL */
#define FALSE ( 0 == 1 )

/*-- Variables globales ------------------------------------------------*/

VP          vptr;             /* Ptr sur 1er caractère en mémoire écran */
unsigned    atimes = 0;  /* Nombre d'activations du programme résident */
union vel   *scrbuf;                  /* Pointeur sur buffer d'écran */
char        *lignevide;                 /* Pointeur sur ligne vide */

/*************************************************************************
*  Fonction         : D I S P _ I N I T                                  *
**-----------------------------------------------------------------------**
*          Donne l'adresse de base de la mémoire d'écran.               *
*  Entrées          : néant                                             *
*  Valeur retournée : néant                                             *
*************************************************************************/

void disp_init(void)
{
  union REGS regs;          /* Registres pour gérer les interruptions */

  regs.h.ah = 15;                  /* fonction "Déterminer mode vidéo" */
  int86(0x10, &regs, &regs);          /* interruption vidéo du BIOS */

/*-- Calcule adresse de base de mém. écran en fonction du mode vidéo -*/

  vptr = (VP) MK_FP((regs.h.al == 7) ? 0xb000 : 0xb800, 0);
}

/*************************************************************************
*  Fonction         : D I S P _ P R I N T                                *
**-----------------------------------------------------------------------**
*     Affiche une chaîne de caractères sur l'écran.                     *
*  Entrées: - COLONNE = Colonne d'affichage.                            *
*           - LIGNE   = Ligne d'affichage.                              *
*           - COULEUR = Attribut des caractères.                        *
*           - STRING  = Pointeur sur chaîne                             *
```

```
!*  Valeur retournée : néant                                            *
!*********************************************************************/
!
!void disp_print(BYTE colonne, BYTE ligne, BYTE couleur, char * string)
!{
!  register VP lptr;              /* Ptr pour accéder à mémoire écran */
!
!  lptr = VPOS(colonne, ligne);                  /* ptr mémoire écran */
!  for( ; *string ; ++lptr)                      /* Parcourt la chaîne */
!  {
!    lptr->h.caractere = *(string++);    /* caractère en mémoire écran */
!    lptr->h.attribut = couleur;          /* ainsi que son attribut */
!  }
!}
!
!/*********************************************************************
!*  Fonction        : S A V E _ S C R E N                              *
!**-----------------------------------------------------------------**
!*       Sauvegarde le contenu de l'écran dans un buffer             *
!*  Entrées: - SPTR = Pointeur sur le buffer dans lequel             *
!*                    l'écran va être sauvegardé                     *
!*  Valeur retournée : néant                                         *
!*  Info             : On suppose que le buffer est assez grand      *
!*                     pour mémoriser le contenu de l'écran.         *
!*********************************************************************/
!
!void save_screen( union vel * sptr )
!{
!  register VP lptr;        /* Pointeur courant sur la mémoire d'écran */
!  unsigned    i;                                        /* Compteur */
!  lptr = VPOS(0, 0);                          /* Positionne le pointeur */
!
!  for(i=0; i<2000; i++)      /* Parcourt les 2000 coordonnées d'écran */
!    (sptr++)->x.contenu = (lptr++)->x.contenu;  /* Mém. car. et attr. */
!}
!
!/*********************************************************************
!*  Fonction        : R E S T O R E _ S C R E E N                      *
!**-----------------------------------------------------------------**
!*     Copie le contenu d'un buffer dans la mémoire d'écran          *
!*  Entrée : - SPTR = Pointeur sur le buffer où se trouve le contenu de *
!*                    l'écran.                                       *
!*  Valeur retournée : néant                                         *
!*********************************************************************/
!
!void restore_screen( union vel * sptr )
!{
!  register VP lptr;     /* Pointeur pour accéder à la mémoire d'écran */
!  unsigned i;                                          /* Compteur */
!  lptr = VPOS(0, 0);                          /* Positionne le pointeur */
!
!  for(i=0; i<2000; i++)   /* Parcourt les 2000 coordonnées d'un écran */
!    (lptr++)->x.contenu = (sptr++)->x.contenu;      /* Lit car.+attr. */
!}
!
!/*********************************************************************
!*  Fonction        : E N D F C T                                      *
!**-----------------------------------------------------------------**
!*     Appelée lors de la désinstallation du programme résident      *
!*  Entrée           : néant                                         *
!*  Valeur retournée : néant                                         *
!*  Info             : Cette procédure doit être FAR pour pouvoir être *
!*                     invoquée à partir de l'exemplaire déjà installé *
!*                     du programme résident.                        *
!*********************************************************************/
!
!void far endfct( void )
!{            /*-- Libère les buffers alloués ----------------------*/
!
! free( lignevide );
! free( (void *) scrbuf );
!
! printf("Le programme résident a été activé  %u fois.\n", stimes);
!}
!
!/*********************************************************************
!*  Fonction        : T S R                                            *
!**-----------------------------------------------------------------**
!*     Appelée par le module en assembleur au moment où la touche    *
!*               d'activation est actionnée.                         *
!*  Entrées          : néant                                         *
!*  Valeur retournée : néant                                         *
!*********************************************************************/
!
!void tsr( void )
!{
!  BYTE  i;                                              /* Compteur */
!
!  ++stimes;                        /* Incrémente le nombre d'activations */
!
!  while( KeyAvail() )                        /* Vide le buffer du clavier */
!    GetKey();
!
!  disp_init();                /* Cherche l'adresse de la mémoire d'écran */
!  save_screen( scrbuf );      /* Mémorise le contenu courant de l'écran */
!  for(i=0; i<25; i++)                /* Parcourt les 25 lignes de l'écran */
!    disp_print(0, i, INV, lignevide);           /* Efface chaque ligne */
!  disp_print(22, 11, INV, "TSRC  -  (c) 1988, 92 by MICHAEL TISCHER");
!  disp_print(28, 13, INV, "Appuyez sur une touche SVP ...");
!  GetKey();                                  /* Attend une frappe de touche */
!
!  restore_screen( scrbuf );                    /* Restaure l'ancien écran */
!}
!
!/*********************************************************************
!*  GetHeapEnd: détermine la fin actuelle du tas en fonction          *
!*              du compilateur                                        *
!*  Entrée : néant                                                    *
!*  Sortie : Ptr sur le premier octet situé après la partie occupée   *
!*           du tas                                                   *
!*********************************************************************/
!
!void far *GetHeapEnd( void )
!{
!#ifdef __TURBOC__                                          /* TurboC ? */
!  return (void far *) sbrk(0);
!#else                                                      /* Non  MSC */
!  struct _heapinfo hi;       /* Stuct. avec informations sur le tas */
!  unsigned         heapstatus;                /* Etat de _heapwalk() */
!  void far         *dernier; /* Pointeur sur le dernier bloc occupé */
!
!  hi._pentry = NULL;                          /* Commence au début du tas */
!
!    /*-- Parcourt le tas jusqu'au dernier bloc ----------------------*/
!
!  while( (heapstatus = _heapwalk( &hi )) != _HEAPEND )
!    if( hi._useflag == _USEDENTRY )                  /* Bloc occupé ? */
!      dernier = (void far *) ((BYTE far *) hi._pentry + hi._size + 1);
!
!  return dernier;
!#endif
!}
!
!/*********************************************************************
!*  ParamGetHotKey: Recherche dans les param. de la ligne de commande *
!*   la définition de la touche d'activation (/T) et l'exploite       *
!*  Entrées : ARGC,   = Paramètres de la ligne de commande, comme pour *
!*                      main()                                       *
!*            ARGV                                                   *
!*            KEYMASK = Pointeur sur la variable destinée à mémoriser *
!*                      le masque de la touche                       *
!*            SCCODE  = Pointeur sur la variable qui doit mémoriser  *
!*                      le scan code                                 *
!*  Sortie : TRUE, si la touche d'activation est identifiée, sinon FALSE *
!*  Info   : - Les paramètres qui ne sont pas introduits par /T      *
!*             ne sont pas exploités : ils sont laissés à la disposition *
!*             d'autres fonctions                                    *
!*           - Si aucun paramètre /T n'est détecté, les variables    *
!*             contiennent respectivement les valeurs 0 et SC_NOKEY. *
!*********************************************************************/
!
!BOOL ParamGetHotKey(int argc, char *argv[], WORD *KeyMask, BYTE *ScCode)
!{
!  struct TComman
!         {
!           char  Nom[7];
!           WORD  Valeur;
!         };
!
!  static struct TComman ToucheC[9] =
!               {
!                 { "LSHIFT", LSHIFT },
!                 { "RSHIFT" ,RSHIFT },
!                 { "CTRL"   ,CTRL   },
!                 { "ALT"    ,ALT    },
!                 { "SYSREQ" ,SYSREQ },
!                 { "BREAK"  ,BREAK  },
!                 { "NUM"    ,NUM    },
!                 { "CAPS"   ,CAPS   },
!                 { "INSERT" ,INSERT }
!               };
!
!  int i , j,                                          /* Compteurs de boucle */
!      code;                                 /* Pour convertir le scan code */
!      char arg[80];                          /* Pour mémoriser un argument */
!
!  *KeyMask = 0;
!  *ScCode = SC_NOKEY;
!
!  for( i = 1; i < argc; ++i )          /* Parcourt la ligne de commande */
!  {
!    strcpy( arg, argv[i] );                              /* Lit un argument */
!    strupr( arg );
!    if( arg[0] == '/'  &&  arg[1] == 'T' )
!    {                                                /* Est-ce un argument /T ? */
!      code = atoi( &arg[2] );          /* Transforme le code en binaire */
```

```
    if( code )                                  /* Conversion réussie ? */
    {                                                          /* Oui */
      if( code < 128 )                             /* Code valide ? */
        *ScCode = code;                            /* Oui, le mémorise */
      else
        return FALSE;                              /* Code non valable */
    }
    else                          /* Pas de nombre : touche de commande */
    {
      for( j = 0; j < 9; ++ j )   /* Parcourt le tableau des touches */
        if(!strcmp( ToucheC[j].Nom, &arg[2] ))          /* comparaison */
          break;                          /* Egalité, on sort de la boucle */

      if( j < 9 )                          /* Nom de la touche trouvé ? */
        *KeyMask = *KeyMask | ToucheC[j].Valeur;  /* Oui, indicateur */
      else
        return FALSE;          /* Non, ni nombre ni touche de commande */
    }
  }
}
return TRUE;                          /* Si la fonction arrive jusqu'ici, */
}                                    /* c'est que les paramètres sont bons */

/*********************************************************************
**                        PROGRAMME PRINCIPAL                       **
*********************************************************************/

void main( int argc, char *argv[] )
{
 WORD  KeyMask;                /* masque bin. pour touches de commande */
 BYTE  ScCode;     /* Mémorise le scan code de la touche d'activation */

 printf("TSRC  -  (c) 1988, 92 by MICHAEL TISCHER\n");
 if(!ParamGetHotKey( argc, argv, &KeyMask, &ScCode ))
 {          /* Erreur dans les paramèteres de la ligne de commande */
   printf( "Paramètre erroné dans la ligne de commande \n" );
   exit(1);
 }

 /*-- Les paramètres de la ligne de commande sont en ordre----------*/

 if( !TsrIsInst( I2F_CODE ))              /* Programme déjà installé ? */
 {                                                          /* Non */
   atimes = 0;                 /* Le programme n'a pas encore été activé */
   printf( "Le programme résident a été installé.\n" );
   if( KeyMask == 0  &&  ScCode == SC_NOKEY )  /* pas de paramètres ? */
   {                                    /* Non, valeur implicite ALT-H */
     TsrSetHotkey( ALT, SC_H );
     printf( "Activation: <ALT> + H\n" );
   }
   else                                /* touche d'activation utilisateur */
     TsrSetHotkey( KeyMask, ScCode );

   /*-- Alloue un buffer pour la gestion de l'écran ----------------*/

   scrbuf = (union vel *) malloc(80 * 25 * sizeof(union vel));
   lignevide = (char *) malloc( 80 + 1 );       /* Réservation mémoire */
   *(lignevide + 80 ) = '\0';                  /* Clôture le buffer par NUL */
   memset(lignevide, ' ', 80);                  /* et le remplit d'espaces */

   TsrInit( TC, tsr, TAS_LIBRE );                /* Installe le programme */
 }
 else                                          /* Programme déjà installé */
 {                                                             /* OUI*/
   if( KeyMask == 0  &&  ScCode == SC_NOKEY )  /* Pas de paramètres ? */
   {                                        /* Non tente une désinstallation */
     if( TsrCanUnInst() )
     {
       (*(SAFP) TsrSetPtr(endfct))();
       TsrUnInst();
       printf( "Le programme a pu être désinstallé .\n" );
     }
     else
       printf( "Le programme ne peut pas être désinstallé.\n" );
   }
   else                              /* Fixe la nouvelle touche d'activation */
   {
     printf( "Nouvelle touche d'activation installée \n" );
     (*(SHKFP) TsrSetPtr(TsrSetHotkey))( KeyMask, ScCode );
   }
 }
}
```

Listing : TSRCA.ASM

```
;*****************************************************************;
;*                            T S R C A                          *;
;*---------------------------------------------------------------*;
;*   Fonction       : Interface en assembleur permettant de rendre *;
;*                    résident un programme écrit en langage C     *;
;*---------------------------------------------------------------*;
;*   Auteur         : MICHAEL TISCHER                              *;
;*   Développé le   : 10.08.1988                                   *;
;*   Dernière MAJ   : 18.04.1992                                   *;
;*---------------------------------------------------------------*;
;*   Assemblage     : MASM -mx TSRCA;     ou r TASM /mx TSRCA;     *;
;*                    ... puis lier au programme en C             *;
;*****************************************************************;

IGROUP group _text                  ;Regroupe les segments de programme
DGROUP group _bss, _data            ;Regroupe les segments de données
        assume CS:IGROUP, DS:DGROUP, ES:DGROUP, SS:DGROUP

_BSS    segment word public 'BSS'       ;Segment des variables statiques
_BSS    ends                                        ;non initialisées

_DATA   segment word public 'DATA'      ;Segment des variables globales
                                          ;et statiques non initialisées

extrn   __psp : word                    ;Segment du PSP du programme en C
_DATA   ends

;-- Constantes ---------------------------------------------------

TC_STACK    equ 512                                 ;Pile pour Turbo C
ARG_WORDS   equ 32                      ;Pile de mots copiés dans TsrCall

I2F_FCT_0   equ 0AAh                    ;Code pour INT 2F, Fonction 0
I2F_FCT_1   equ 0BBh                    ;Code pour INT 2F, Fonction 1
TIME_OUT    equ 9                         ;Time-Out en nombre de tops

;-- Programme ----------------------------------------------------

_TEXT   segment byte public 'CODE'                ;Segment de programme

;-- Références aux fonctions externes en C-----------------------

extrn   _GetHeapEnd:near              ;Renvoie l'adresse de fin du tas

;-- Déclarations publiques de fonctions internes -----------------

public  _TsrInit              ;Permet l'appel à partir du programme en C
public  _TsrIsInst
public  _TsrUnInst
public  _TsrCanUnInst
public  _TsrCall
public  _TsrSetHotkey
public  _TsrSetPtr

;-- Variables pour les gestionnaires d'interruption--------------
;-- (accessibles uniquement par le segment de code )-------------

call_ptr    equ this dword
call_ofs    dw 0                                  ;Offset pour TSRCall
call_seg    dw 0                                ;Segment non initialisé

ret_ax      dw 0                    ;Sauvegarde du résultat de la fonction
ret_dx      dw 0                                        ;pour TsrCall

;-- Variables nécessaires à l'activation du programme en C -------

c_ss        dw 0                                  ;Segment de pile en C
c_sp        dw 0                                 ;Pointeur de pile en C
c_ds        dw 0                               ;Segment de données en C
c_es        dw 0                                   ;Segment extra en C

c_dta_ofs   dw 0                      ;Adresse de la DTA du programme en C
c_dta_seg   dw 0

c_psp       dw 0                          ;Segment du PSP du programme en C
break_adr   dw 0                                ;Adresse de Break du tas
fct_adr     dw 0                      ;Adresse de la fonction résidente en C

;-- Variables pour tester la touche d'activation -----------------
```

```
key_mask   dw 3            ;Masque de touche pour indicateur clavier BIOS
                           ;Par défaut : Shift gauche + Shift droite
sc_code    db 128          ;Scan code de la touche d'activation
                           ;Par défaut pas de touche
f2F_code   db 0            ;Numéro de fonction pour INT 2F

;-- Variables pour l'activation du programme résident ----------------

tsrnow     db 0                    ;En attente d'activation ?
tsractif   db 0                    ;Programme résident déjà actif ?
in_bios    db 0            ;Indique une activité du disque dans le BIOS

daptr      equ this dword  ;Pointeur sur l'indicateur Indos de DOS
daptr_ofs  dw 0                                        ;Offset
daptr_seg  dw 0                                        ;Segment

;-- Les variables suivantes mémorisent les adresse des anciens    - ---
;-- gestionnaires d'interruption qui sont remplacés par les nouveaux ---

int8_ptr   equ this dword      ;Ancien vecteur de l'interruption 8h
int8_ofs   dw 0                  ;Offset de l'ancien gestionnaire
int8_seg   dw 0                 ;Segment de l'ancien gestionnaire

int9_ptr   equ this dword      ;Ancien vecteur de l'interruption 9h
int9_ofs   dw 0                  ;Offset de l'ancien gestionnaire
int9_seg   dw 0                 ;Segment de l'ancien gestionnaire

int13_ptr  equ this dword      ;Ancien vecteur de l'interruption 13h
int13_ofs  dw 0                  ;Offset de l'ancien gestionnaire
int13_seg  dw 0                 ;Segment de l'ancien gestionnaire

int28_ptr  equ this dword      ;Ancien vecteur de l'interruption 28h
int28_ofs  dw 0                  ;Offset de l'ancien gestionnaire
int28_seg  dw 0                 ;Segment de l'ancien gestionnaire

int2F_ptr  equ this dword      ;Ancien vecteur de l'interruption 2Fh
int2F_ofs  dw 0                  ;Offset de l'ancien gestionnaire
int2F_seg  dw 0                 ;Segment de l'ancien gestionnaire

;-- Variables pour garder des informations du programme interrompu -----

iu_dta_ofs dw 0                 ;Adresse DTA du prog interrompu .
iu_dta_seg dw 0

iu_psp     dw 0                 ;Segment du PSP du prog interrompu

iuprg_ss   dw 0                     ;SS et SP du prog interrompu
iuprg_sp   dw 0

;-----------------------------------------------------------------------
;-- TSRINIT: Termine le programme en C et active
;: les nouveaux gestionnaires d'interruption
;-- Appel depuis C: void TsrInit(  bool TC,
;--                          void (fct *)(void),
;--                          unsigned heap_byte );

_TsrInit  proc   near

sframe0   struc                 ;Structure pour accéder à la pile
bp0       dw ?                              ;Mémorise BP
ret_adr0  dw ?                          ;Adresse de retour
tc0       dw ?          ;Compilateur (1 = TURBO-C, 0 = MSC )
fctptr0   dw ?         ;pointeur sur fonction résidente en C
heap0     dw ?                      ;Octets sur le tas
sframe0   ends                     ;Fin de la structure

frame     equ [ bp - bp0 ]

          push bp                       ;Sauvegarde BP sur la pile
          mov  bp,sp                          ;Transfère SP en BP

          ;-- Sauvegarde les registres de segment de C ----------------

          mov  c_ss,ss              ;Mémorise les registres dans les
          mov  c_sp,sp                ;Variables prévues à cet effet
          mov  c_es,es
          mov  c_ds,ds

          ;-- mémorise les paramètres transmis   ----------------------

          mov  ax,frame.fctptr0   ;Lit le ptr sur fonction résidente C
          mov  fct_adr,ax                             ;et le mémorise

          ;-- Adresse de la DTA du programme en C ----------------------

          mov  ah,2fh      ;Nº de la fonction: Lire l'adresse de la DTA
          int  21h                  ;Déclenche l'interruption de DOS
          mov  c_dta_ofs,bx           ;Mémorise l'adresse à l'endroit
          mov  c_dta_seg,es                              ; prévu

          ;-- Lit l'adresse de l'indicateur INDOS----------------------

          mov  ah,34h          ;Nº fonc. : Lire adresse indicateur INDOS
          int  21h                      ;Déclenche l'interruption de DOS
          mov  daptr_ofs,bx       ;Mémorise l'adresse à l'endroit prévu
          mov  daptr_seg,es

          ;-- Lit les adresses des interruptions à détourner ----------

          mov  ax,3508h             ;Lit le vecteur de l'interruption 8h
          int  21h                        ;Appelle l'interruption de DOS
          mov  int8_ofs,bx          ;Mémorise l'adresse du gestionnaire
          mov  int8_seg,es        ;dans les variables prévues à cet effet

          mov  ax,3509h             ;Lit le vecteur de l'interruption 9h
          int  21h                        ;Appelle l'interruption de DOS
          mov  int9_ofs,bx          ;Mémorise l'adresse du gestionnaire
          mov  int9_seg,es        ;dans les variables prévues à cet effet

          mov  ax,3513h            ;Lit le vecteur de l'interruption 13h
          int  21h                        ;Appelle l'interruption de DOS
          mov  int13_ofs,bx         ;Mémorise l'adresse du gestionnaire
          mov  int13_seg,es       ;dans les variables prévues à cet effet

          mov  ax,3528h            ;Lit le vecteur de l'interruption 28h
          int  21h                        ;Appelle l'interruption de DOS
          mov  int28_ofs,bx         ;Mémorise l'adresse du gestionnaire
          mov  int28_seg,es       ;dans les variables prévues à cet effet

          mov  ax,352Fh            ;Lit le vecteur de l'interruption 2Fh
          int  21h                        ;Appelle l'interruption de DOS
          mov  int2F_ofs,bx         ;Mémorise l'adresse du gestionnaire
          mov  int2F_seg,es       ;dans les variables prévues à cet effet

          ;-- Installe les nouveaux gestionnaires d'interruption-------

          push ds                          ;Mémorise le segment de données
          mov  ax,cs                        ;Transfère CS en AX puis en DS
          mov  ds,ax

          mov  ax,2508h           ;Nº fonction : Fixer l'interruption 8h
          mov  dx,offset int08          ;DS:DX = adresse du gestionnaire
          int  21h                        ;Appelle l'interruption de DOS

          mov  ax,2509h           ;Nº fonction : Fixer interruption 9h
          mov  dx,offset int09          ;DS:DX = adresse du gestionnaire
          int  21h                        ;Appelle l'interruption de DOS

          mov  ax,2513h          ;Nº fonction : Fixer l'interruption 13h
          mov  dx,offset int13          ;DS:DX = adresse du gestionnaire
          int  21h                        ;Appelle l'interruption de DOS

          mov  ax,2528h          ;Nº fonction : Fixer l'interruption 28h
          mov  dx,offset int28          ;DS:DX =adresse du gestionnaire
          int  21h                        ;Appelle l'interruption de DOS

          mov  ax,252Fh          ;Nº fonction : Fixer l'interruption 2Fh
          mov  dx,offset int2F          ;DS:DX=adresse du gestionnaire
          int  21h                        ;Appelle l'interruption de DOS

          pop  ds                                 ;reprend DS sur la pile

          ;-- Calcule le nombre de paragraphes qui doivent rester -----
          ;-- en mémoire ----------------------------------------------

          call _GetHeapEnd              ;fonction C dans le module résident
          add  ax,frame.heap0      ;Ajoute la mémoire du tas nécessaire

          ;-- Dans le cas de TURBO-C la pile se trouve derrière le tas-
          ;-- commence à la fin du segment il faut donc la rajouter ---
          ;-- au tas --------------------------------------------------

          cmp  byte ptr frame.tc0,0              ;Utilise-t-on TURBO-C ?
          je   msc2                                          ;NON, MSC

          add  ax,TC_STACK-1       ;Calcule nouveau ptr de pile pour TC
          mov  c_sp,ax                                  ;et le mémorise
          inc  ax                                  ;fixe adresse de break

          ;-- Calcule le nombre de pargraphes qui doivent rester ------
          ;-- en mémoire ----------------------------------------------

msc2:     mov  dx,ax          ;Cherche l'adresse de Break et la met en DX
          add  dx,15            ;Evite la perte due à la division entière
          mov  cl,4                   ;Décale de 4 positions vers la droite
          shr  dx,cl                                  ; = division par 16
          mov  ax,ds                             ;Transfère DS en AX
          mov  bx,__psp                        ;Lit le segment du PSP
          mov  c_psp,bx                                ; et le mémorise
          sub  ax,bx                          ;Retranche DS de PSP
          add  dx,ax          ; et additionne le rés au nombre de parag.
          mov  ax,3100h                              ;Nº fonction : TSR
          int  21h                        ;Déclenche l'interruption de DOS
                                                    ; termine le programme

_TsrInit  endp
```

```
;
;       assume CS:IGROUP, DS:nothing, ES:nothing, SS:nothing
;
;-------------------------------------------------------------------
;-- TSRSETHOTKEY: Fixe la touche d'activation du programme
;-- Apepl depuis C: void TsrSetHotKey( unsigned KeyMask,
;--                                    byte     ScanCode );
;-- Info          : cette procédure est FAR pour qu'elle puisse être
;--       apelée dans un programme résident déjà installé
;--
;
_TsrSetHotkey  proc far
;
sframe1   struc                         ;Structure pour accéder à la pile
bp1       dw ?                                  ;Mémorise BP
ret_adr1  dd ?                                  ;Adresse de retour
keymask1  dw ?                          ;Masque pour la touche d'activation
sc_code1  dw ?                          ;Scan code de la touche d'activation
sframe1   ends                                  ;Fin de la structure
;
frame     equ [ bp - bp1 ]
;
          push bp                               ;Sauvegarde BP sur la pile
          mov  bp,sp                                  ;Transfère SP en BP
;
          ;-- Mémorise les paramètres transmis  ----------------------
;
          mov  ax,frame.keymask1   ;Lit/mém. masque touche d'activation
          mov  key_mask,ax
          mov  al,byte ptr frame.sc_code1       ;Lit scan code de touche
          mov  sc_code,al                             ;et le mémorise
;
          pop  bp                               ;Récupère BP sur la pile
          ret                                                 ;Retour
_TsrSetHotkey  endp
;
;-------------------------------------------------------------------
;-- TSRISINST: Teste si le programme est déjà installé ---------------
;-- Appel depuis C : bool TsrIsInst( byte i2f_fctnr);
;-- Valeur de retour : TRUE, si le programme est déjà installé ,
;--                    sinon FALSE
;
_TsrIsInst proc   near
;
sframe2   struc                         ;Structure pour accéder à la pile
bp2       dw ?                                  ;Mémorise BP
ret_adr2  dw ?                                  ;Adresse de retour
i2F_code2 dw ?                          ;Numéro de fonction pour INT 2F
sframe2   ends                                  ;Fin de la structure
;
frame     equ [ bp - bp2 ]
;
          push bp                               ;Sauvegarde BP sur la pile
          mov  bp,sp                                  ;Transfère SP en BP
;
          mov  ah,byte ptr frame.i2F_code2  ;N° de fonction pour INT 2F
          mov  i2F_code,ah                            ; à sauvegarder
          mov  al,I2F_FCT_0                                 ;N° d'option
          mov  bx,ax                            ;Prend note des deux numéros
          int  2Fh
          xchg bh,bl                                    ;Les intervertit
          cmp  ax,bx                          ;et compare à val de retour
          mov  ax,0                     ;A priori pas encore d'installation
          jne  isi_end                        ;non égal --> pas d'install.
;
          ;-- Segment de l'exemplaire déjà installé ----------------
;
          mov  ah,i2F_code                        ;Non, segment par INT 2Fh
          mov  al,I2F_FCT_1                           ;Charge l'option 01h
          int  2Fh
          mov  call_seg,ax                            ;Mémorise le résultat
          mov  ax,-1                                    ;Installation oui
;
isi_end:  pop  bp                               ;Récupère BP sur la pile
          ret                                                 ;retour
;
_TsrIsInst endp                                         ;Fin de la procédure
;
;-------------------------------------------------------------------
;-- TSRCANUNIST : Teste si l'exemplaire installé du programme résident--
;--               peut être désinstallé
;-- Appel depuis C Bool TsrCanUnInst( void );
;-- Sortie      : TRUE, si désinstallation possible, sinon FALSE
;-- Info        : Le programme ne peut être désinstallé que si
;--               aucun de ses vecteurs d'interruption
;--               n'a été détourné entretemps par un autre programme
;
tsrlist   db  08h,09h,13h,28h,2Fh,00h   ;Liste interruptions détournées
                                                ;00h marqueur de fin de liste
_TsrCanUninst proc  near
;
          push di                                               ;Sauve DI
          mov  dx,call_seg  ;Charge le segment de d'exemplaire installé
          mov  di,offset tsrlist-1                          ;DI sur liste
;
tcu_1:    inc  di                           ;DI sur numéro d'interr. suivant
          mov  al,cs:[di]                             ;Numéro suivant en AL
          or   al,al                                    ;Fin de la liste ?
          je   tcu_ok                                   ;Oui, ts vecteurs ok
;
          mov  ah,35h                       ;N° de la fonction "Get Interrupt"
          int  21h                            ;Appelle l'interruption de DOS
          mov  cx,es                                    ;Transfère ES en CX
          cmp  dx,cx                                        ;Même segment ?
          je   tcu_1                        ;Oui, pas de désinstallation possible
;
          xor  ax,ax                        ;Non pas de désinstallation possible
          pop  di
          ret
;
tcu_ok:   mov  ax,-1
;
          pop  di                                           ;restaure DI
          ret
;
_TsrCanUninst endp
;
;-------------------------------------------------------------------
;-- TSRUNINST: Désinstalle le programme résident et libère la mémoire --
;--            allouée
;-- Appel depuis C: void TsrUnInst( void );
;-- Info          : Cette routine ne doit être appelée que si
;--                 TSRCANUNINST() a renvoyé la valeur TRUE
;--
;_TsrUninst    proc    near
;
          push ds
          mov  es,call_seg      ;Segment du programme résident installé
;
          ;-- Rétablit les anciens gestionnaires d'interruption -------
;
          cli                                       ;Inhibe toute interruption
          mov  ax,2508h         ;N° fonction : Fixer gestionnaire INT 8
          mov  ds,es:int8_seg                         ;Rétablit le segment
          mov  dx,es:int8_ofs   ; et l'offset de l'ancien gestionnaire
          int  21h
;
          mov  ax,2509h         ;N° fonction : Fixer gestionnaire INT 9
          mov  ds,es:int9_seg                         ;Restaure le segment
          mov  dx,es:int9_ofs   ; et l'offset de l'ancien gestionnaire
          int  21h
;
          mov  ax,2513h        ;N° fonction : Fixer gestionnaire INT 13
          mov  ds,es:int13_seg                        ;Restaure le segment
          mov  dx,es:int13_ofs  ; et l'offset de l'ancien gestionnaire
          int  21h
;
          mov  ax,2528h        ;N° fonction : Fixer gestionnaire INT 28
          mov  ds,es:int28_seg                        ;Restaure le segment
          mov  dx,es:int28_ofs  ; et l'offset de l'ancien gestionnaire
          int  21h
;
          mov  ax,252Fh        ;N° fonction : Fixer gestionnaire INT 2F
          mov  ds,es:int2F_seg                        ;Restaure le segment
          mov  dx,es:int2F_ofs  ; et l'offset de l'ancien gestionnaire
          int  21h
          sti                           ;Autorise à nouveau les interruptions
;
          ;-- Libère la mémoire  -----------------------------------
;
          mov  es,es:c_psp          ;CX=segment PSP du programme résident
          mov  cx,es
          mov  es,es:[ 02ch ]                         ;PSP=seg de l'environment
          mov  ah,49h           ;N° de la fonction: Libérer de la mémoire
          int  21h                              ;Appelle l'interruption de DOS
;
          mov  es,cx                                    ;Reprend ES dans CX
          mov  ah,49h           ;N° de la fonction: Libérer de la mémoire
          int  21h                              ;Appelle l'interruption de DOS
;
          pop  ds                               ;récupère DS et BP sur la pile
          ret                                         ;retourne à l'appelant
;
_TsrUninst endp                                         ;Fin de procédure
;
;-------------------------------------------------------------------
;-- TSRSETPTR: Mémorise l'adresse de la routine qui devra être      --
;--            déclenchée lors d'un appel ultérieur à TSRCALL et
;--            renvoie un pointeur sur TSRCALL
;-- Appel depuis C: void (*)(void) TsrSetPtr( void far * Fct );
;
_TsrSetPtr  proc    near
;
sframe3   struc                         ;Structure pour accéder à la pile
bp3       dw ?                                  ;Mémorise BP
ret_adr3  dw ?                                  ;Adresse de retour
```

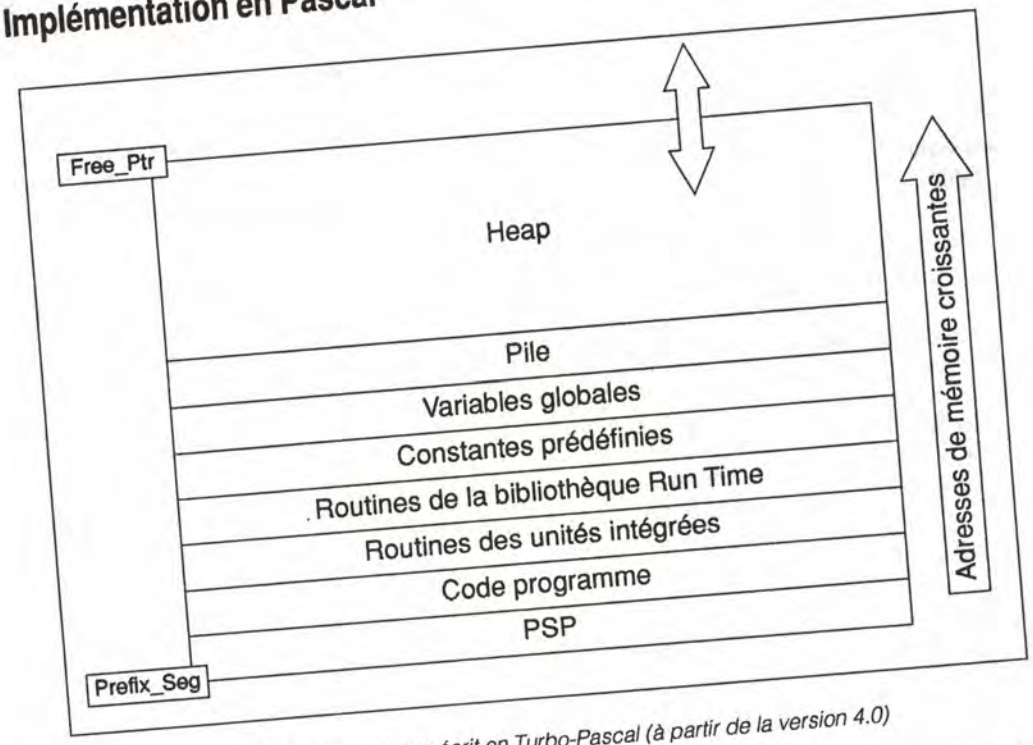
```
fctptr3     dd ?                        ;Pointeur sur la routine à appeler
sframe3     ends                                ;Fin de la structure

frame       equ [ bp - bp3 ]

            push bp                             ;Sauvegarde BP sur la pile
            mov  bp,sp                              ;Transfère SP en BP

            mov  ax,word ptr frame.fctptr3      ;Charge l'offset en AX
            mov  call_ofs,ax                        ; puis le stocke

            mov  ax,offset _TsrCall ;Renvoie un pointeur Near sur TSRCALL
            pop  bp                             ;Récupère BP sur la pile

            ret                                 ;Retourne à l'appelant

_TsrSetPtr endp                                 ;Fin de la procédure

;-----------------------------------------------------------------------
;-- TSRCALL: Appelle une  routine de l'exemplaire du programme résident-
;--          préalablement installé
;-- Appel depuis C: void TsrCall( void )
;-- Attention   Les arguments de la pile de l'appelant sont copiés
;:       sur la pile du programme à appeler et la pile
;:       est changée de façon que l'on ait toujours DS=SS

_TsrCall    proc near

            push di                             ;Prend note de DS, SI et DI
            push si
            push ds

            mov  ah,2fh     ;N° de la fonction: "Lire l'adresse de la DTA"
            int  21h                        ;Appelle l'interruption de DOS
            mov  u_dta_ofs,bx           ;Sauvegarde l'adresse de la DTA du
            mov  u_dta_seg,es                       ;programme interrompu

            mov  es,call_seg                    ;Charge en ES le segment du
                                                ;programme résident installé"
            mov  ah,50h         ;N° de la fonction: Fixer l'adresse du PSP
            mov  bx,es:c_psp                    ;Lit le segment du PSP
            int  21h                        ;Appelle l'interruption de DOS

            mov  ah,1ah     ;N° de la fonction: Fixer l'adresse de la DTA
            mov  dx,es:c_dta_ofs            ;Offset de la nouvelle DTA
            mov  ds,es:c_dta_seg            ;Segment de la nouvelle DTA
            int  21h                        ;Appelle l'interruption de DOS


            ;-- Copie les arguments sur la pile du programme ------------
            ;-- résident installé ---------------------------------------

            push ss                         ;DS:SI pointent sur les arguments
            pop  ds                                 ; de la pile courante
            mov  si,sp
            add  si,8                       ;Par dessus RET et PUSH DS,DI,SI

            mov  cx,ARG_WORDS*2
            mov  di,es:c_sp                 ;ES:DI=pile du programme installé
            sub  di,cx
            mov  es,es:c_ss
            rep  movsb                              ;Copie les arguments

            ;-- Fixe registres de segment du programme résident installé-

            mov  es,call_seg        ;Segment du programme résident installé
                                                    ;en ES

            cli                                 ;Inhibe les interruptions
            mov  uprg_ss,ss                 ;Mémorise le segment et le
            mov  uprg_sp,sp                 ; pointeur de pile courants

            mov  ss,es:c_ss                 ;Active la pile de l'exemplaire
            mov  sp,es:c_sp                 ;installé du programme résident
            sub  sp,ARG_WORDS*2             ;Passe par dessus les arguments
            sti                         ;Autorise à nouveau les interruptions

            mov  ds,es:c_ds     ;Fixe registre de segment pour programme C
            mov  es,es:c_es                         ;

            call [call_ptr]             ;Mémorise le résultat de la fonction
            mov  ret_ax,ax                  ;transmise par TSRSETPTR
            mov  ret_dx,dx

            cli                             ;Interdit toute interruption
            mov  ss,uprg_ss                 ;Repasse sur la pile initiale
            mov  sp,uprg_sp                         ;
            sti                             ;Rétablit les interruptions

            ;-- Effectue changement de contexte vers programme actuel ---

            mov  ah,1ah     ;N° de la fonction: Fixer l'adresse de la DTA
            mov  dx,u_dta_ofs   ;Charge l'offset et le segment de la DTA
            mov  ds,u_dta_seg                       ; du programme interrompu
            int  21h                        ;Déclenche l'interruption de DOS

            mov  es,call_seg                    ;Reprend ES et DS sur la pile
            pop  ds                                 ;

            mov  ah,50h         ;N° de la fonction: Fixer l'adresse du PSP
            mov  bx,cs                                  ;transfère CS en BX
            sub  bx,10h                         ;Calcule le segment du PSP
            int  21h                        ;Déclenche l'interruption de DOS

            mov  ax,ret_ax              ;Ramène le résultat de la fonction
            mov  dx,ret_dx

            pop  si                             ;Récupère les registres
            pop  di
            ret                                 ;Retourne à l'appelant

_TsrCall    endp                                ;Fin de la procédure

;-----------------------------------------------------------------------
;-- DOSACTIF: Détermine par l'indicateur INDOS si DOS peut être
;--           interrompu .
;-- Entrée:  néant
;-- Sortie:  Indicateur de zéro=1: DOS peut être interrompu

dosactif    proc near

            push ds                         ;Stocke BX et DS sur la pile
            push bx
            lds  bx,daptr           ;DS:BX pointent sur l'indicateur INDOS
            cmp  byte ptr [bx],0                ;Fonction de DOS active ?
            pop  bx                         ;Récupère BX et DS sur la pile
            pop  ds

            ret                                 ;Retourne à l'appelant

dosactif    endp

;-----------------------------------------------------------------------
;--          Voici les nouveaux gestionnaires d'interruption          ----
;-----------------------------------------------------------------------

;-- Nouveau gestionnaire de l'interruption 8h (Timer)------------------

int08       proc far

            cmp  tsrnow,0           ;Faut-il activer le programme résident
            je   i8_end                 ;Non, passe à l'ancien gestionnaire

            dec  tsrnow         ;Oui, décrémente l'indicateur d'activation

            ;-- TSR doit être activé mais est-ce possible ? -------------

            cmp  in_bios, 0         ;Interruption disque du BIOS active ?
            jne  i8_end                 ;OUI --> pas d'activation possible

            call dosactif                   ;DOS peut-il être interrompu ?
            je   i8_tsr                 ;Oui, appelle le programme résident

i8_end:     jmp  [int8_ptr]             ;retourne à l'ancien gestionnaire

            ;-- Active le programme résident ----------------------------

i8_tsr:     mov  tsrnow,0           ;Le TSR n'attend plus son activation
            mov  tsractif,1   ;le programme résident est maintenant actif
            pushf               ;Simule l'appel de l'ancien gestionnaire
            call [int8_ptr]                         ; par INT 8h
            call start_tsr                  ;Lance le programme résident
            iret                        ;retourne au programme interrompu

int08       endp

;-- Nouveau gestionnaire de l'interruption 09h (clavier)---------------

int09       proc far

            push ax
            in   al,60h                             ;Lit le port du clavier

            cmp  tsractif,0    ;Le programme résident est-il déjà actif ?
            jne  i9_end   ;OUI: appelle l'ancien gestionnaire puis retour

            cmp  tsrnow,0  ;Le programme est-il en attente d'activation ?
            jne  i9_end   ;OUI: appelle l'ancien gestionnaire puis retour

            ;-- Teste la touche d'activation  --------------------------

            cmp  sc_code,128                        ;Y a-t-il un scan code ?
            je   i9_ks          ;Non, ne teste que les touches de commande

            cmp  al,128                     ;Oui est-ce un code release ?
            jae i9_end                              ;Oui, pas d'activation
```

```
;
        cmp  sc_code,al          ;Code make à comparer avec le modèle
        jne  i9_end                 ;Pas d'activation si pas le même
;
i9_ks:  ;-- Teste l'état des touches de commande ------------------
;
        push ds
        mov  ax,040h                ;DS sur le segment des variables
        mov  ds,ax                                      ; du BIOS
        mov  ax,word ptr ds:[17h]  ;lit indic. d'état clavier en BIOS
        and  ax,key_mask              ;bits de la touche d'activation
        cmp  ax,key_mask             ;bits de la touche d'activation ?
        pop  ds
        jne  i9_end    ;Touche d'activation détectée ? NON --> retour
;
        cmp  in_bios, 0       ;Interruption disque du BIOS en cours ?
        jne  i9_e1                  ;OUI --> pas d'activation possible
;
        call dosactif             ;Est-il possible d'interrompre DOS ?
        je   i9_tsr                  ;Oui, lance le programme résident
;
i9_e1:  mov  tsrnow,TIME_OUT              ;TSR en attente d'activation
;
i9_end: pop  ax                                        ;Reprend AX
        jmp  [int9_ptr]         ;Se branche sur l'ancien gestionnaire
;
i9_tsr: mov  tsractif,1       ;Le TSR va être actif (dans un moment)
        mov  tsrnow,0                 ;Pas de délai de lancement
        pushf
        call [int9_ptr]              ;Appelle l'ancien gestionnaire
        pop  ax                                    ;Récupère AX
        call start_tsr               ;Lance le programme résident
        iret                        ;retourne au programme interrompu
;
int09   endp
;
;-- Nouveau gestionnaire de l'interruption 13 h (disque(tte)) ---------
;
int13   proc far
;
        inc  in_bios          ;Incrémente l'indicateur disque du BIOS
        pushf                ;Simule l'appel de l'ancien gestionnaire
        call [int13_ptr]                              ; par INT 13h
        dec  in_bios          ;Restaure l'indicateur disque du BIOS-
;
        sti                          ;Autorise à nouveau les interruptions
        ret  2                         ;Retourne à l'appelant, mais enlève
                ;en même temps le registre des indicateurs de la pile
int13   endp
;
;-- Nouveau gestionnaire de l'interruption 28h (DOS idle) -------------
;
int28   proc far
;
        cmp  tsrnow,0   ;Programme résident en attente d'activation ?
        je   i28_end                          ;Non, retourne à l'appelant
;
        cmp  in_bios, 0   ;Oui, mais a-t-on une interruption disque ?
        je   i28_tsr                          ;Oui, donc pas d'activation
;
i28_end: jmp [int28_ptr]                 ;retourne à l'ancien gestionnaire
;
        ;-- Lance le programme résident ----------------------------
;
i28_tsr: mov  tsrnow,0                    ;Le TSR n'est plus en attente
        mov  tsractif,1             ;Le TSR va être actif dans un moment
        pushf                      ;Simule l'appel de l'ancien gestionnaire
        call [int28_ptr]                    ;d'interruption par INT 28h
        call start_tsr                      ;Lance le programme résident
        iret                                    ;Retourne à l'appelant
;
int28   endp
;
;-- Nouveau gestionnaire de l'interruption 2Fh (multiplexeur) ---------
;
int2f   proc far
;
        cmp  ah,i2f_code  ;Appel de ce présent programme résident ? ?
        jne  i2f_end              ;Non, retourne à l'ancien gestionnaire
;
;
        cmp  al,i2f_fct_0           ;Oui, est-ce la sous-fonction 00h?
        je   i2f_0                                  ;Oui, passe à l'exécution
;
        cmp  al,i2f_fct_1     ;peut-être est-ce la sous-fonction 01h?
        je   i2f_1                                  ;Oui, passe à l'exécution
;
        iret                                            ;Non, ignore l'appel
;
i2f_end: ;-- Le TSR n'est pas concerné, fait suivre l'appel ----------
;
        jmp  [int2f_ptr]                        ;vers l'ancien gestionnaire
;
i2f_0:  ;-- Sous-fonction 00: test d'installation ------------------
;
        xchg ah,al      ;Echange numéros de fonction et sous-fonction.
        iret                                      ;Retourne à l'appelant
;
i2f_1:  ;-- Sous-fonction 01: retourne le segment ------------------
;
        mov  ax,cs                                        ;Segment en AX
        iret                                    ;Retourne à l'appelant
;
int2f   endp
;
;-- START_TSR: Active le programme résident ---------------------------
;
start_tsr  proc near
;
        ;-- Changement de contexte vers le programme en C------------
;
        cli                                        ;Inhibe les interruptions
        mov  uprg_ss,ss           ;Prend note du segment et du pointeur
        mov  uprg_sp,sp                                    ; de pile
;
        mov  ss,c_ss                    ;Active la pile du programme en C
        mov  sp,c_sp
        sti                                      ;Rétablit les interruptions
;
        push ax                  ;Sauvegarde les registres du processeur
        push bx                                      ;sur la pile de C
        push cx
        push dx
        push bp
        push si
        push di
        push ds
        push es
;
    ;-- Sauvegarde 64 mots de la pile de DOS -------------------------
;
        mov  cx,64                                      ;Compteur de boucle
        mov  ds,uprg_ss    ;DS:SI pointe sur la fin de la pile de DOS
        mov  si,uprg_sp
;
tsrs1:  push word ptr [si]        ;Transfère un mot de la pile de DOS
        inc  si                  ;sur la pile de C et fait pointer SI
        inc  si                                      ;sur le mot suivant
        loop tsrs1                             ;Traite tous les 64 mots
;
        mov  ah,51h          ;N° de la fonction: Lire l'adresse du PSP
        int  21h                              ;Appelle l'interruption de DOS
        mov  u_psp,bx                        ;Mémorise le segment du PSP
;
        mov  ah,2fh             ;N° de la fonction: Lire l'adresse DTA
        int  21h                        ;Appelle l'interruption de DOS
        mov  u_dta_ofs,bx                    ;Sauve l'adresse de la DTA
        mov  u_dta_seg,es                     ; du programme interrompu
;
        mov  ah,50h      ;N° de la fonction: "Fixer l'adresse du PSP"
        mov  bx,c_psp                  ;Sauvegard. PSP des Prog Turbo C
        int  21h                         ;Appelle l'interruption de DOS
;
        mov  ah,1ah           ;N° de la fonction: Fixer l'adresse DTA
        mov  dx,c_dta_ofs                                ;Lit l'offet
        mov  ds,c_dta_seg             ;et le segment de la nouvelle DTA
        int  21h                        ;Appelle l'interruption de DOS
;
        mov  ds,c_ds                     ;fixe les registres de segment
        mov  es,c_es                                ; pour le programme C
;
        call [fct_adr]     ;Appelle fonction de lancement programme C
;
        ;-- Changement de contexte vers le programme interrompu -----
;
        mov  ah,1ah  ;N° de la fonction: "Fixer l'adresse de la DTA "
        mov  dx,u_dta_ofs    ;Charge l'offset et le segment de la DTA
        mov  ds,u_dta_seg                        ;du programme interrompu
        int  21h                          ;Appelle l'interruption de DOS
;
        mov  ah,50h       ;N° de la fonction: "Fixe l'adresse du PSP"
        mov  bx,u_psp                    ;Seg du PSP du prog interr. ,
        int  21h                         ;Appelle l'interruption de DOS
;
        ;-- restaure la pile de DOS--------------------------------
;
        mov  cx,64                                          ;Compteur
        mov  ds,uprg_ss          ;DS:SI=adresse de fin de pile de DOS
        mov  si,uprg_sp                                          ;
        add  si,128                        ;SI au début de la pile de DOS
tsrs2:  dec  si                                  ;SI sur mot précédent
        dec  si
        pop  word ptr [si]             ;mot de pile C vers pile de DOS
        loop tsrs2                                  ;Extraire 64 mots
;
        pop  es                           ;Reprend les registres sauvés
```

```
                                        ;sur la pile C
        pop  ds
        pop  di
        pop  si
        pop  bp
        pop  dx
        pop  cx
        pop  bx
        pop  ax

        cli                             ;Inhibe les interruptions
        mov  ss,uprg_ss                 ;Restaure me pointeur et le segment
        mov  sp,uprg_sp                 ;de pile du programme interrompu
```

```
        mov  tsractif,0         ;Le programme résident n'est plus actif
        sti                     ;Autorise à nouveau les interruptions

        ret                             ;Retourne à l'appelant

start_tsr  endp
;-------------------------------------------------------------------------
                                        ;Fin du segment de code
_text   ends                            ;Fin du programme
        end
```

Implémentation en Pascal



Structure en mémoire d'un programme écrit en Turbo-Pascal (à partir de la version 4.0)

Contrairement aux compilateurs C du monde des PC, Turbo Pascal ne supporte qu'un seul modèle de mémoire auquel se trouve confronté l'implémentation de programmes résidents.

La figure précédente montre que le PSP est suivi du code et des routines issues des différentes unités et de la bibliothèque Run Time. Puis viennent les constantes prédéfinies, les données globales et le segment de pile. La taille de ces éléments est fixée au moment de la compilation du programme et ne se modifie plus par la suite quand le programme est chargé en mémoire centrale. Mais la taille du tas qui est contigu au segment de pile n'est pas prédéterminée. Lorsque des objets nouveaux sont créés ou

alloués par NEW, le tas croît en direction de l'extrémité de la mémoire, tandis que les désallocations par RELEASE le ramènent du côté de la fin du segment de pile.

Par rapport aux compilateurs C, Turbo Pascal offre l'avantage de permettre la fixation de la taille maximale du tas ainsi que celle de la pile par une directive de compilation au sein du code source. Il s'agit plus précisément de la directive $M qui s'utilise de la façon suivante :

```
{$M Taille de la pile, taille minimale du tas, taille maximum du tas }
```

Toutes les indications doivent être exprimées en octets. En écrivant :

```
{$M 2048, 0, 5000}
```

on installe au moment de la compilation une pile de 2 Ko et un tas s'étendant au maximum sur 5000 octets. Si ce type de directive n'est pas mis en place, le tas n'est pas limité et peut aller jusqu'à l'extrémité de la mémoire existante. En conséquence, la totalité de la mémoire devrait être réservée au programme résident ce qui évidemment. Aucun espace ne serait plus disponible pour les autres programmes. Avec la directive $M on peut limiter la taille du programme en mémoire et calculer en conséquence le nombre de paragraphes qui doivent rester résidents.

Là aussi Turbo Pascal présente un avantage car le nombre de paragraphes à réserver peut être déterminé dans le programme en Pascal ce qui évite les fastidieuses manipulations en assembleur. Le programme en C ne peut accéder à certaines données indispensables pour ce calcul (adresse du PSP, du segment de données et taille du tas) : il doit donc s'appuyer sur l'assembleur. Mais Turbo pascal définit ces informations dans des variables ordinaires sous forme de pointeurs de programmes. Ce sont surtout les adresses de début du PSP et de fin du tas qui nous intéressent pour délimiter la partie résidente du programme.

Le schéma précédent montre que l'adresse du PSP est donnée par PrefixSeg tandis que l'extrémité du tas peut être obtenue jusqu'à la version 6.0 par la variable (pointeur) FreePtr. Il est vrai que cette variable ne référence pas directement la fin du tas mais la partie segment du poinnteur est égale à l'adresse de la fin du tas moins $1000. A partir de la version 6.0 la gestion du tas a été quelque peu modifiée. On dispose alors du pointeur HeapEnd qui indique bien la fin du tas.

La procédure ResPara du programme résident exploite ces informations pour calculer le nombre de paragraphes à laisser en mémoire. Une directive permet de mettre en service soit HeapPtr soit HeapEnd en fonction de la version de Turbo Pascal.

La fonction résidente écrite en Pascal dont l'adresse est communiquée comme premier paramètre à TsrInit doit être une procédure qui se trouve à l'intérieur du programme principal et non dans une unité. Elle ne doit pas être transformée en procédure FAR par la directive de compilation $F+ car l'interface en assembleur suppose qu'il s'agit d'une procédure NEAR. Il faut donc prendre connaissance de l'adresse de cette

procédure par la fonction OFS avant de la transmettre de la même façon à TsrInit, sinon Turbo Pascal empile non seulement l'offset mais aussi le segment.

Par contre les procédures et fonctions à appeler de l'intérieur du programme résident doivent être de type FAR. Leur appel ne s'écrit pas d'une façon aussi confortable qu'en C car le transtypage des pointeurs de fonction est interdit en Turbo Pascal. TsrSetPtr ne renvoie pas de résultat et les appels de TsrSetPtr et de TsrCall ne peuvent pas être combinés.

Dans la version en Pascal il faut déclarer les pointeurs de code qui décrivent les procédures ou fonctions à appeler ainsi que leurs arguments. Comme le montre l'extrait suivant de TSRP.PAS, ces pointeurs s'appellent SAProcT et SHKProcT. SAProcT représente un pointeur qui référence une procédure sans argument, tandis que SHKProcT est créé pour les besoins de la procédure TsrSetHotKey :

```
type SAProcT  = procedure;
     SHKProcT = procedure( Keymask : word; ScCode  : byte );
     PPtrT  = record
         case integer of
             1 : ( SAProc  : SAProcT  );
             2 : ( SHKProc : SHKProcT );
         end;
const Call : PPtrT = ( SAProc : TsrCall
```

Les types sont rassemblés dans un enregistrement variable dont les deux composantes s'appellent OAProc et SHKProc. Pour invoquer les fonctions liés à ces types, on définit une variable globale appelée CAll dont la composante SAProc est initialisée par un pointeur sur TsrCall.

La procédure ou fonction souhaitée peut être appelée par cette variable pour autant que son offset ait été transmis auparavant à TsrSetPtr. Les deux lignes suivantes montrent comment TsrSetPtr fixe d'abord TsrHotKey comme routine à appeler puis TsrCall est appelé avec les arguments destinés à TsrSetHotkey.

```
TsrSetPtr(ofs(TsrSetHotKey));
Call.SHKProc( Keymask, ScCode );
```

Ceci est rendu possible parce que le compilateur en voyant la composante SHKProc de Call suppose qu'une telle procédure est vraiment appelée. Mais en réalité c'est TsrCall qui est exécuté, avec plus de facilités d'ailleurs pour gérer la pile que son pendant en C.

Car les programmes en Turbo pascal ne supposent pas a priori que le segment de données est égal au segment de pile. Ainsi à l'appel de la procédure ou de la fonction dans l'exemplaire déjà installé du programme a pile de l'exemplaire en cours d'exécution peut rester active.

Listing : TSRP.PAS

```
{*****************************************************************}
{*                            T S R P                            *}
{*---------------------------------------------------------------*}
{*   Fonction       : Crée un programme résident à l'aide d'un   *}
{*                    module en assembleur                       *}
{*---------------------------------------------------------------*}
{*   Auteur         : MICHAEL TISCHER                            *}
{*   Développé le   : 18.08.1988                                 *}
{*   Dernière MAJ   : 18.03.1992                                 *}
{*****************************************************************}

program TSRP;

uses DOS, CRT;                          { Intègre les unités DOS et CRT }

{$M 2048, 0, 5120}          { 2Ko pour la pile et 5Ko maxi pour le tas }
{$L tsrpa}                          { Intègre le module en assembleur }

{-- Déclaration des fonctions externes du module en assembleur --------}

procedure TsrInit( PrzPtr   : word; { Offset de la procédure résidente }
                   ResPara  : word  { Nombre de paragraphes résidents }
                 ) ; external ;          { Chaine d'identification }
function  TsrIsInst( I2F_fctnr : byte ) : boolean ; external ;
procedure TsrUnInst; external;      { Désinstalle le programme résident }
procedure TsrSetPtr( Offset : word ); external;
function  TsrCanUnInst : boolean; external;

{$F+}                       { Ci-dessous les procédures et fonctions FAR }

procedure TsrCall ; external;
procedure TsrSetHotKey( KeyMask  : word;        { shortcut (cf CONST) }
                        ScCode   : byte                 { Scan code }
                      ) ; external;

{$F-}

{-- Constantes --------------------------------------------------------}

{-- Scan codes de différentes touches ---------------------------------}

const SC_ESC            = $01;      SC_Y            = $2C;
      SC_1              = $02;      SC_X            = $2D;
      SC_2              = $03;      SC_C            = $2E;
      SC_3              = $04;      SC_V            = $2F;
      SC_4              = $05;      SC_B            = $30;
      SC_5              = $06;      SC_N            = $31;
      SC_6              = $07;      SC_M            = $32;
      SC_7              = $08;      SC_VIRGULE      = $33;
      SC_8              = $09;      SC_PPOINT       = $34;
      SC_9              = $0A;      SC_TIRET        = $35;
      SC_0              = $0B;      SC_SHIFT_GAUCHE = $36;
      SC_SCHARFES_S     = $0C;      SC_PRINT_SCREEN = $37;
      SC_APOSTROPH      = $0D;      SC_ALT          = $38;
      SC_BACKSPACE      = $0E;      SC_SPACE        = $39;
      SC_TAB            = $0F;      SC_CAPS         = $3A;
      SC_Q              = $10;      SC_F1           = $3B;
      SC_W              = $11;      SC_F2           = $3C;
      SC_E              = $12;      SC_F3           = $3D;
      SC_R              = $13;      SC_F4           = $3E;
      SC_T              = $14;      SC_F5           = $3F;
      SC_Z              = $15;      SC_F6           = $40;
      SC_U              = $16;      SC_F7           = $41;
      SC_I              = $17;      SC_F8           = $42;
      SC_O              = $18;      SC_F9           = $43;
      SC_P              = $19;      SC_F10          = $44;
      SC_UE             = $1A;      SC_NUM_LOCK     = $45;
      SC_PLUS           = $1B;      SC_SCROLL_LOCK  = $46;
      SC_RETURN         = $1C;      SC_CURSOR_HOME  = $47;
      SC_CONTROL        = $1D;      SC_CURSOR_UP    = $48;
      SC_A              = $1E;      SC_CURSOR_PG_UP = $49;
      SC_S              = $1F;      SC_NUM_MOINS    = $4A;
      SC_D              = $20;      SC_CURSOR_LEFT  = $4B;
      SC_F              = $21;      SC_NUM_5        = $4C;
      SC_G              = $22;      SC_CURSOR_RIGHT = $4D;
      SC_H              = $23;      SC_NUM_PLUS     = $4E;
      SC_J              = $24;      SC_CURSOR_END   = $4F;
      SC_K              = $25;      SC_CURSOR_DOWN  = $50;
      SC_L              = $26;      SC_CURSOR_PG_DOWN = $51;
      SC_OE             = $27;      SC_INSERT       = $52;
      SC_AE             = $28;      SC_DELETE       = $53;
      SC_PLUSGRAND      = $29;      SC_SYS_REQUEST  = $54;
      SC_SHIFT_DROIT    = $2A;      SC_F11          = $57;
      SC_FIS            = $2B;      SC_F12          = $58;
      SC_NOKEY          = $80;          { Pas de touche additionnelle }

      {-- Masques binaires pour les touches de commande ---------------}

      RSHIFT =    1;
      LSHIFT =    2;                              { Majuscule gauche }
      CTRL   =    4;                                 { Touche CTRL }
      ALT    =    8;                                  { Touche ALT }
      SYSREQ =  1024;             { Touche SYS-REQ (Clavier AT uniqt ) }
      BREAK  =  4096;                               { Touche BREAK }
      NUM    =  8192;                                 { Touche NUM }
      CAPS   = 16384;                                { Touche CAPS }
      INSERT = 32768;                              { Touche INSERT }

      I2F_CODE  = $C4;              { Numéro de la fonction INT 2F }
      I2F_FKT_0 = $AA;              { Code pour INT 2F, fonction 0 }
      I2F_FKT_1 = $BB;              { Code pour INT 2F, fonction 1 }

{-- Déclarations de types ---------------------------------------------}

type VBuf   = array[1..25, 1..80] of word;          { Décrit un écran }
     VPtr   = ^VBuf;                        { Pointe sur un buffer d'écran }

(*
   Déclarations des types de fonction et de procédures grâce auxquels
   il sera possible d'appeler des fonctions et des procédures de
   l'exemplaire déjà installé du programme résident
*)

     SAProcT = procedure;                           { Procédure sans arguments }
     SHKProcT = procedure( KeyMask : word;                { TsrSetHotkey }
                           ScCode  : byte );
     PPtrT = record      { Union pour compiler les ptrs de procédure }
               case integer of
                 1 : ( SAProc  : SAProcT );
                 2 : ( SHKProc : SHKProcT );
             end;

const Call : PPtrT = ( SAProc : TsrCall );

{-- Variables globales ------------------------------------------------}

var MBuf   : VBuf absolute $8000:0000;    { Mémoire d'écran monochrome }
    CBuf   : Vbuf absolute $B800:0000;        { Mémoire d'écran couleur }
    VioPtr : VPtr;                        { Pointeur sur la mémoire d'écran }
    ATimes : integer;                { Nombre d'activations du prog résident }

{*****************************************************************}
{* DispInit: Crée un pointeur sur la mémoire d'écran             *}
{* Entrée  : néant                                               *}
{* Sortie  : néant                                               *}
{*****************************************************************}

procedure DispInit;

var Regs: Registers;                        { Stocke le contenu des registres }

begin
  Regs.ah := $0f;           { Fonction N° 15 = déterminer le mode vidéo }
  Intr($10, Regs);             { Déclenche l'interruption vidéo du BIOS }
  if Regs.al=7 then                                { Carte écran monochrome ? }
    VioPtr := @MBuf         { Oui, fixe ptr sur mémoire écran monochrome }
  else                                  { Il s'agit d'une carte EGA, VGA ou CGA }
    VioPtr := @CBuf;                    { Fixe ptr sur mémoire écran couleur }
end;

{*****************************************************************}
{* SaveScreen: Sauvegarde le contenu de l'écran dans un buffer   *}
{* Entrée : SPTR : Pointeur sur le buffer qui va recevoir les données *}
{*                 sauvegardées                                  *}
{* Sortie : néant                                                *}
{*****************************************************************}

procedure SaveScreen( SPtr : VPtr );

var ligne,                                                  { Ligne courante }
    colonne : byte;                                     { Colonne courante }

begin
  for ligne:=1 to 25 do             { Parcourt les 25 lignes de l'écran }
    for colonne:=1 to 80 do                 { ainsi que les 80 colonnes }
      SPtr^[ligne, colonne] := VioPtr^[ligne, colonne];
                                  { Transfère un caractère et son attribut }
end;

{*****************************************************************}
{* RestoreScreen: Copie le contenu d'un buffer dans la mémoire d'écran*}
{* Entrée : BPTR : Pointeur sur le buffer dont le contenu doit   *}
{*                 être copié dans la mémoire d'écran            *}
{* Sortie : néant                                                *}
```

```
{*****************************************************************}

procedure RestoreScreen( BPtr : VPtr );

var ligne,                                          { Ligne courante }
    colonne : byte;                                 { Colonne courante }

begin
  for ligne:=1 to 25 do              { Parcourt les 25 lignes de l'écran }
    for colonne:=1 to 80 do                  { ainsi que les 80 colonnes }
      VioPtr^[ligne, colonne] := BPtr^[ligne, colonne];
                                  { Transfère un carcatère et son attribut }
end;

{*****************************************************************}
{* ResPara: Calcule le nombre de paragraphes à allouer au programme   *}
{*          résident                                                  *}
{* Entrées : néant                                                   *}
{* Sortie : Nombre de paragraphes à réserver                         *}
{*****************************************************************}

function ResPara : word;

begin

{$ifdef VER50}                                          { Pour TP 5.0 }
    ResPara := Seg(FreePtr^)+$1000-PrefixSeg;  { Nombre de paragraphes }
{$endif}

{$ifdef VER55}                                          { pour TP 5.5 }
    ResPara := Seg(FreePtr^)+$1000-PrefixSeg;  { Nombre de paragraphes }
{$endif}

{$ifdef VER60}                                          { pour TP 6.0 }
    ResPara := Seg(HeapEnd^)-PrefixSeg;
{$endif}

end;

{*****************************************************************}
{* ParamGetHotKey: Recherche dans la ligne de commande le paramètre /T*}
{*                 puis l'exploite                                   *}
{* Entrées : KEYMASK = Variable pour mémoriser le masque de la touche *}
{*           SCCODE  = Variable pour mémoriser le scan code          *}
{* Sortie  : TRUE, si la touche d'activation est identifiée, sinon   *}
{*           FALSE                                                   *}
{* Info    : - Les paramètres qui ne sont pas introduits par /T ne   *}
{*             sont pas traités pour être laissés à la disposition   *}
{*             d'autres fonctions                                    *}
{*           - Si aucun paramètre /T n'est détecté, les variables    *}
{*             contiennent respectivement les valeurs 0 et SC_NOKEY. *}
{*****************************************************************}

function ParamGetHotKey( var KeyMask : word;
             var ScCode  : byte ) : boolean;

type TComman = record
                 Nom : string[6];
                 Valeur : word;
               end;

const ToucheC : array[ 1..9 ] of TComman =
        ( ( Nom: 'LSHIFT'; Valeur : LSHIFT ),
          ( Nom: 'RSHIFT'; Valeur : RSHIFT ),
          ( Nom: 'CTRL';   Valeur : CTRL   ),
          ( Nom: 'ALT';    Valeur : ALT    ),
          ( Nom: 'SYSREQ'; Valeur : SYSREQ ),
          ( Nom: 'BREAK';  Valeur : BREAK  ),
          ( Nom: 'NUM';    Valeur : NUM    ),
          ( Nom: 'CAPS';   Valeur : CAPS   ),
          ( Nom: 'INSERT'; Valeur : INSERT )
        );

var i, j,                                       { Compteurs de boucle }
    code,                                  { Pour convertir le scan code }
    dummy : integer;                       { Mémorise l'erreur pour VAL }
    arg   : string;                                { Stocke un argument }

begin
  KeyMask := 0;
  ScCode  := SC_NOKEY;

  for i := 1 to ParamCount do              { Parcourt la ligne de commande }
    begin
      arg := ParamStr(i);                                { Lit un argument }
      for j := 1 to length(arg) do          { Le convertit en majuscules }
        arg[j] := upcase(arg[j]);
      if( arg[1] = '/' ) and ( arg[2] = 'T' ) then
        begin                                          { C'est un argument /T }
          delete( arg, 1, 2 );                  { Retire le /T de la chaîne }
          val( arg, code, dummy );          { Convertit le reste en binaire }
          if( dummy = 0 ) then                      { Conversion réussie ? }
            begin                                                 { Oui }
              if( code > 0 ) and ( code < 128 ) then  { Code valable ? }
                ScCode := Code                                { Oui, le mémorise }
              else
                begin                                     { Code non valable }
                  ParamGetHotKey := false;
                  exit;      { Termine la fonction avec la valeur FALSE }
                end;
            end
          else                          { Pas de nombre = touche de commande }
            begin
              j := 1; { recherche dans tableau des touches de commande }
              while( j < 10 ) and ( arg <> ToucheC[j].Nom ) do
                j := j + 1;
              if( j < 10 ) then                              { Nom trouvé ? }
                KeyMask := KeyMask or ToucheC[j].Valeur
                                              {Oui, incorpore l'indicateur }
              else
                begin          { Non, ni nombre ni touche de commande }
                  ParamGetHotKey := false;
                  exit;      { Termine la fonction avec la valeur FALSE }
                end;
            end;
        end;
    end;
  ParamGetHotKey := true;          { Si la fonction parvient jusqu'ici , }
end;                                        { c'est que les paramètres sont o.k. }

{*****************************************************************}
{* EndPrc: Appelée par le module assembleur lors de la désinstallation*}
{*         du programme résident                                     *}
{* Entrée : néant                                                    *}
{* Sortie : néant                                                    *}
{* Info : Cette procédure doit se trouver dans le programme principal *}
{*        et doit être transformée par une directive $F+             *}
{*        en procédure FAR                                           *}
{*****************************************************************}

{$F+}

procedure EndPrc;

begin
  TextBackground( Black );                                { Fond sombre }
  TextColor( LightGray );                          { Caractères clairs }
  writeln('Le programme résident a été activé ', ATimes, ' fois .');
end;

{$F-}

{*****************************************************************}
{* Tsr: Procédure appelée par le module en assembleur lorsqu'on      *}
{*      actionne la touche d'activation                              *}
{* Entrée : néant                                                    *}
{* Sortie : néant                                                    *}
{* Info : Cette procédure doit se trouver dans le programme principal *}
{*        et ne doit pas être transformée par une directive $F+      *}
{*        en procédure FAR                                           *}
{*****************************************************************}

{$F-}                                          { pas de procédure FAR }

procedure Tsr;

var BufPtr : VPtr;          { Mémorise un pointeur sur le buffer alloué }
    Colonne,                                  { Colonne d'écran courante }
    Ligne  : byte;                                { Ligne courante }
    Touche : char;

begin
  while KeyPressed do                          { Vide le buffer du clavier }
    Touche := ReadKey;
  Inc( ATimes );                                { Incrémente le compteur d'appels }
  DispInit;                                { Lit l'adresse de la mémoire d'écran }
  GetMem(BufPtr, SizeOf(VBuf) );                              { Alloue un buffer }
  SaveScreen( BufPtr );                  { Sauvegarde le contenu de l'écran }
  Ligne := WhereY;                                          { Ligne actuelle }
  Colonne := WhereX;                                     { Colonne actuelle }
  TextBackground( LightGray );                                 { Fond clair }
  TextColor( Black );                                   { Caractères foncés }
  ClrScr;                                              { Efface tout l'écran }
  GotoXY(22, 12);
  write('TSRP  -  (c) 1988, 92 by MICHAEL TISCHER');
  GotoXY(30, 14);
  write('Appuyez sur une touche SVP ...');
  Touche := ReadKey;                              { Attend une frappe de touche }
  RestoreScreen( BufPtr );                          { Restaure l'ancien écran }
  FreeMem( BufPtr, SizeOf(VBuf) );                    { Libère le buffer alloué }
  GotoXY( Colonne, Ligne );     { Remet le curseur en position initiale }
end;

{*****************************************************************}
{**                        PROGRAMME PRINCIPAL                      *}
{*****************************************************************}
```

```
var KeyMask : word;
    ScCode : byte;

begin
  writeln('TSRP  -  (c) 1988, 92 by MICHAEL TISCHER');
  if not ParamGetHotKey( KeyMask, ScCode ) then
    begin          ( Erreur dans les paramètres de la ligne de commande )
      writeln( 'Paramètre erroné dans la ligne de commande !');
      exit;
    end;

      {-- Les paramètres de la ligne de commande sont corrects --------}

  if( TsrIsInst( I2F_CODE ) = FALSE ) then  (Programme déjà installé ?)
                                                          ( Non )
    begin
      ATimes := 0;              ( Le programme n'a pas encore été activé )
      writeln( 'Le programme résident a été installé' );
      if( KeyMask = 0 ) and ( ScCode = SC_NOKEY ) then   (Pas de para ?)
        begin                            ( Non valeur implicite = ALT H )
          TsrSetHotkey( ALT, SC_H );
          writeln( 'Activation: <ALT> + H' );
        end
      else                    ( Installe touche d'activation utilisateur )
        TsrSetHotkey( KeyMask, ScCode );
      TsrInit( Ofs(Tsr), ResPara );              ( Installe le programme )
    end                                          ( Programm déjà installé )
  else
    begin
      if( KeyMask = 0 ) and ( ScCode = SC_NOKEY ) then   (Pas de para.?)
                                                ( Non tente une désinstallation )
        begin
          if TsrCanUnInst then
            begin
              TsrSetPtr(ofs(EndPrc));     ( Appelle la procédure EndPrc )
              Call.SAProc;              ( dans l'exemplaire déjà installé )
              TsrUnInst;
              writeln( 'Le programme a pu être désinstallé.');
            end
          else
            writeln( 'Le programme n''a pas pu être désinstallé .');
        end
      else                              ( Fixe une nouvelle touche d'activation )
        begin
          writeln( 'Nouvelle touche d''activation installée ' );
          TsrSetPtr(ofs(TsrSetHotKey));
          Call.SHKProc( KeyMask, ScCode );
        end
    end;
end.
```

Listing : TSRPA.ASM

```
;*****************************************************************;
;*                            T S R P A                          *;
;*---------------------------------------------------------------*;
;*   Fonction : Interface en assembleur permettant de rendre     *;
;*              résident un programme écrit en Turbo Pascal      *;
;*---------------------------------------------------------------*;
;*   Auteur      : MICHAEL TISCHER                               *;
;*   Développé le : 12.08.1988                                   *;
;*   Dernière MAJ : 18.03.1992                                   *;
;*---------------------------------------------------------------*;
;*   Info        : Ce module doit être intégré dans un programme *;
;*                 et non dans une unité                         *;
;*---------------------------------------------------------------*;
;*   Assemblage  : MASM TSRPA;    ou    TASM TSRPA               *;
;*                 puis intégrer dans un programme en TP         *;
;*****************************************************************;

DATA    segment word public                     ;Segment de données TP
DATA    ends                                    ;Fin du segment de données

;-- Constantes ---------------------------------------------------

I2F_FCT_0  equ 0AAh                     ;Code pour INT 2F, Fonction 0
I2F_FCT_1  equ 0BBh                     ;Code pour INT 2F, Fonction 1
TIME_OUT   equ 9            ;Time-Out en nombre de tops pour l'activation

;-- Programme ----------------------------------------------------

CODE        segment byte public                     ;Segment de code TP

            assume cs:CODE, ds:DATA

;-- Déclarations publiques de fonctions internes-----------------

public    tsrinit     ;autorise les appels à partir du programme en TP
public    tsrisinst
public    tsruninst
public    tsrcanuninst
public    tsrsetptr
public    tsrcall
public    tsrsethotkey

;-- Variables pour les gestionnaires d'interruption--------------
;-- (accessibles uniquement par le segment de code )-------------

call_ptr   equ this dword
call_ofs   dw 0                                 ;Offset pour TSRCall
call_seg   dw 0                                 ;Segment non initialisé

ds_save    dw 0                         ;Sauvegarde de DS pendant TSRCall
rptr_save  equ this dword           ;Pointeur FAR pour retour de TSR-CALL
rip_save   dw 0                          ;Sauvegarde de l'adresse de retour
                                                            ;pour TSR-Call
rcs_save   dw 0
ret_ax     dw 0                 ;Sauvegarde du résultat de la fonction pour
ret_dx     dw 0                                             ;TsrCall

;-- Variables nécessaires à l'activation du programme TP ---------

t_ss       dw 0                                     ;Segment de pile TP
t_sp       dw 0                                     ;Pointeur de pile TP
t_ds       dw 0                                     ;Segment de données TP
t_es       dw 0                                     ;Segment Extra TP

t_dta_ofs  dw 0                          ;Adresse de la DTA du programme TP
t_dta_seg  dw 0

t_psp      dw 0                                 ;Segment du PSP du programme TP
Prc_adr    dw 0                       ;Adresse de la procédure résidente en TP

;-- Variables pour tester la touche d'activation -----------------

key_mask   dw 3          ;Masque de touche pour indicateur clavier du BIOS
                                    ;Par défaut : Shift gauche + Shift droite
sc_code    db 128                        ;Scan code de la touche d'activation
                                                    ;Par défaut pas de touche
I2F_code   db 0                          ;Numéro de fonction pour INT 2F

;-- Variables pour l'activation du programme résident ------------

tsrnow     db 0                                 ;En attente d'activation ?
tsractif   db 0                             ;Programme résident déjà actif ?
in_bios    db 0                  ;Indique une activité du disque dans le BIOS

daptr      equ this dword          ;Pointeur sur l'indicateur Indos de DOS
daptr_ofs  dw 0                                                  ;Offset
daptr_seg  dw 0                                                  ;Segment

;-- Les variables suivantes mémorisent les adresse des anciens ---------
;-- gestionnaires d'interruption qui sont remplacés par les nouveaux ---

int8_ptr   equ this dword           ;Ancien vecteur de l'interruption 8h
int8_ofs   dw 0                          ;Offset de l'ancien gestionnaire
int8_seg   dw 0                         ;Segment de l'ancien gestionnaire

int9_ptr   equ this dword           ;Ancien vecteur de l'interruption 9h
int9_ofs   dw 0                          ;Offset de l'ancien gestionnaire
int9_seg   dw 0                         ;Segment de l'ancien gestionnaire

int13_ptr  equ this dword           ;Ancien vecteur de l'interruption 13h
int13_ofs  dw 0                          ;Offset de l'ancien gestionnaire
int13_seg  dw 0                         ;Segment de l'ancien gestionnaire

int28_ptr  equ this dword           ;Ancien vecteur de l'interruption 28h
int28_ofs  dw 0                          ;Offset de l'ancien gestionnaire
int28_seg  dw 0                         ;Segment de l'ancien gestionnaire

int2F_ptr  equ this dword           ;Ancien vecteur de l'interruption 2Fh
int2F_ofs  dw 0                          ;Offset de l'ancien gestionnaire
int2F_seg  dw 0                         ;Segment de l'ancien gestionnaire

;-- Variables pour garder des informations du programme interrompu -----

u_dta_ofs  dw 0                          ;Adresse DTA du prog interrompu.
u_dta_seg  dw 0
```

```
;
u_psp     dw 0                          ;Segment du PSP du prog interrompu
;
uprg_ss   dw 0                          ;SS et SP du prog interrompu
uprg_sp   dw 0
;
;;-------------------------------------------------------------------
;;-- TSRINIT: Termine le programme TP et active nouveaux gestionnaires
;;--          d'interruption
;;-- Appel depuis TP: procdure TsrInit( PrcPtr   : word;
;;--                                   ResPara  : word );
;
tsrinit   proc    near
;
sframe0   struc                         ;Structure pour accéder à la pile
bp0       dw ?                                  ;Mémorise BP
ret_adr0  dw ?                                  ;Adresse de retour
respara0  dw ?                          ;Nombre de paragraphes à réserver
PrcPtr0   dw ?                      ;Offset de la procédure résidente en TP
sframe0   ends                                  ;Fin de la structure
;
frame     equ [ bp - bp0 ]
;
          push bp                                       ;Empile BP
          mov  bp,sp                            ;Transfère SP en BP
          push es                             ;Sauvegarde ES sur la pile
;
          ;--Sauvegarde les registres de segment de TP ----------------
;
          mov  t_ss,ss                    ;Mémorise les segments dans les
          mov  t_sp,sp                     ;variables prévues à cet effet
          mov  t_es,es
          mov  t_ds,ds
;
          ;-- Lit le PSP du programme TP       ----------------------
;
          mov  bx,cs                                    ;Transfère CS en BX
          sub  bx,10h           ;10h paragraphes = retrancher 256 octets
          mov  t_psp,bx                                 ;Prend note du segment
;
          ;-- Mémorise les paramètres transmis    ---------------------
;
          mov  ax,frame.PrcPtr0         ;Lit ptr sur procédure résidente
          mov  Prc_adr,ax                                 ;et le mémorise
;
          ;-- Lit l'adresse DTA du programme TP  ----------------------
;
          mov  ah,2fh               ;N° de la fonction: Lit l'adresse DTA
          int  21h                        ;Appelle l'interruption de DOS
          mov  t_dta_ofs,bx               ;Mémorise l'adresse à l'endroit
          mov  t_dta_seg,es                                   ; prévu
;
          ;-- Lit l'adresse de l'indicateur INDOS----------------------
;
          mov  ah,34h       ;N° fonction: Lire adresse indicateur INDOS
          int  21h                        ;Appelle l'interruption de DOS
          mov  daptr_ofs,bx               ;Mémorise l'adresse à l'endroit
          mov  daptr_seg,es                                   ;prévu
;
          ;-- Cherche les adresses des gestionnaires d'interruption à -
          ;-- détourner ----------------------------------------------
;
          mov  ax,3508h             ;Lit le vecteur de l'interruption 8h
          int  21h                        ;Appelle l'interruption de DOS
          mov  int8_ofs,bx          ;Mémorise l'adresse du gestionnaire
          mov  int8_seg,es        ;dans les variables prévues à cet effet

          mov  ax,3509h             ;Lit le vecteur de l'interruption 9h
          int  21h                        ;Appelle l'interruption de DOS
          mov  int9_ofs,bx          ;Mémorise l'adresse du gestionnaire
          mov  int9_seg,es        ;dans les variables prévues à cet effet

          mov  ax,3513h            ;Lit le vecteur de l'interruption 13h
          int  21h                        ;Appelle l'interruption de DOS
          mov  int13_ofs,bx         ;Mémorise l'adresse du gestionnaire
          mov  int13_seg,es       ;dans les variables prévues à cet effet

          mov  ax,3528h            ;Lit le vecteur de l'interruption 28h
          int  21h                        ;Appelle l'interruption de DOS
          mov  int28_ofs,bx         ;Mémorise l'adresse du gestionnaire
          mov  int28_seg,es       ;dans les variables prévues à cet effet

          mov  ax,352Fh            ;Lit le vecteur de l'interruption 2Fh
          int  21h                        ;Appelle l'interruption de DOS
          mov  int2F_ofs,bx         ;Mémorise l'adresse du gestionnaire
          mov  int2F_seg,es       ;dans les variables prévues à cet effet

          ;-- Installe les nouveaux gestionnaires d'interruption-------

          push ds                         ;Mémorise le segment de données
          mov  ax,cs                       ;Transfère CS en AX puis en DS
          mov  ds,ax
;
          mov  ax,2508h     ;N° de la fonction: Fixer l'interruption 8h
          mov  dx,offset int08          ;DS:DX=adresse du gestionnaire
          int  21h                      ;Appelle l'interruption de DOS

          mov  ax,2509h     ;N° de la fonction: Fixer l'interruption 9h
          mov  dx,offset int09          ;DS:DX =adresse du gestionnaire
          int  21h                      ;Appelle l'interruption de DOS

          mov  ax,2513h    ;N° de la fonction: Fixer l'interruption 13h
          mov  dx,offset int13          ;DS:DX=adresse du gestionnaire
          int  21h                      ;Appelle l'interruption de DOS

          mov  ax,2528h    ;N° de la fonction: Fixer l'interruption 28h
          mov  dx,offset int28          ;DS:DX=adresse du gestionnaire
          int  21h                      ;Appelle l'interruption de DOS

          mov  ax,252Fh    ;N° de la fonction: Fixer l'interruption 2Fh
          mov  dx,offset int2F          ;DS:DX=adresse du gestionnaire
          int  21h                      ;Appelle l'interruption de DOS

          pop  ds                                 ;reprend DS sur la pile
;
          ;-- Laisse le programme résident ---------------------------
;
          mov  ax,3100h                                 ;N° fonction: TSR
          mov  dx,frame.respara0 ;Lit le nombre de paragraphes réservés
          int  21h          ;Appelle l'interruption de DOS et termine le
                                                              ;programme
;
tsrinit   endp
;
;;-------------------------------------------------------------------
;;-- TSRSETHOTKEY: Fixe la touche d'activation du programme
;;-- Appel depuis TP: procédure TsrSetHotKey( KeyMask  : word;
;;--                                          ScanCode : byte );
;;-- Info    : Cette procédure est FAR , pour qu'elle puisse aussi
;;--           être appelée dans un programme résident déjà installé.
;;--
;
tsrsethotkey  proc far
;
sframe1   struc                     ;Structure pour accéder à la pile
bp1       dw ?                                  ;Mémorise BP
ret_adr1  dd ?                                  ;Adresse de retour
sc_code1  dw ?                  ;Scan-Code de la touche d'activation
keymask1  dw ?                    ;Masque de la touche d'activation
sframe1   ends                                  ;Fin de la structure
;
frame     equ [ bp - bp1 ]
;
          push bp                                       ;Mémorise BP
          mov  bp,sp                            ;Transfère SP en BP
;
          ;-- Mémorise les paramètres transmis ----------------------
;
          mov  ax,frame.keymask1    ;masque pour la touche d'activation
          mov  key_mask,ax
          mov  al,byte ptr frame.sc_code1       ;Scan code de la touche
          mov  sc_code,al                               ;et le mémorise
;
          pop  bp                               ;Récupère BP sur la pile
          ret  4                      ; et retire les arguments de la pile
;
tsrsethotkey  endp
;
;;-------------------------------------------------------------------
;;-- TSRISINST: teste si le programme est déjà installé          ----
;;-- Appel depuis TP: function TsrIsInst( 12F_fctnr : byte ) : boolean;
;;-- Valeur de retour : TRUE, si programme déjà installé sinon
;;--                    FALSE
;
tsrisinst proc    near
;
sframe2   struc                     ;Structure pour accéder à la pile
bp2       dw ?                                  ;Mémorise BP
ret_adr2  dw ?                                  ;Adresse de retour
I2F_code2 dw ?                  ;Numéro de fonction pour INT 2F
sframe2   ends                                  ;Fin de la structure
;
frame     equ [ bp - bp2 ]
;
          push bp                                       ;Empile BP
          mov  bp,sp                            ;Transfère SP en BP
;
          mov  ah,byte ptr frame.I2F_code2  ;N° de fonction pour INT 2F
          mov  I2F_code,ah                          ; à sauvegarder
          mov  al,I2F_FCT_0                             ;N° d'option
          mov  bx,ax                        ;Prend note des deux numéros
          int  2Fh
          xchg bh,bl                                ;Les intervertit
          cmp  ax,bx                          ;et compare à val de retour
          mov  ax,0                     ;A priori pas encore d'installation
          jne  isi_end                        ;non égal --> pas d'install.
```

```
;
;       ;-- Segment de l'exemplaire déjà installé ------------------
;
        mov  ah,12f_code                ;Non, segment par INT 2Fh
        mov  al,12F_FCT_1                   ;Charge l'option 01h
        int  2Fh
        mov  call_seg,ax                ;Mémorise le résultat
        mov  ax,-1                          ;Installation oui
;
isi_end:  pop  bp                       ;Récupère BP sur la pile
        ret  2                          ; et retire les arguments
;
tsrisinst  endp                                 ;Fin de la procédure
;
;-----------------------------------------------------------------
;-- TSRCANUNIST: Teste si l'exemplaire installé du programme résident --
;--              peut être désinstallé
;-- Appel depuis TP : function TsrCanUnInst : boolean;
;-- Ausgabe         : TRUE, si désinstallation possible sinon  FALSE
;-- Info            : Le programme ne peut être désinstallé que si
;--                   aucun de ses vecteurs d'interruption
;--                 n'a été détourné entretemps par un autre programme -
;
;
tsrlist   db  08h,09h,13h,28h,2Fh,00h       ;Liste des INT détournés
                                          ;00h marque la fin de laliste
tsrcanuninst  proc  near
;
        mov  dx,call_seg  ;Charge le segment de d'exemplaire installé
        mov  di,offset tsrlist-1                    ;DI sur liste
;
tcu_1:  inc  di                         ;DI sur numéro d'interr. suivant
        mov  al,cs:[di]                     ;Numéro suivant en AL
        or   al,al                            ;Fin de la liste ?
        je   tcu_ok                         ;Oui, ts vecteurs ok
;
        mov  ah,35h                 ;N° de la fonction "Get Interrupt"
        int  21h                      ;Appelle l'interruption de DOS
        mov  cx,es                          ;Transfère ES en CX
        cmp  dx,cx                                ;Même segment ?
        je   tcu_1                ;Oui, pas de désinstallation possible
;
        xor  ax,ax                ;Non pas de désinstallation possible
        ret
;
tcu_ok:  mov  ax,-1
        ret
;
tsrcanuninst endp
;
;-----------------------------------------------------------------
;-- TSRUNINST: Désinstalle le programme résident et libère       -----
;--            la mémoire allouée
;-- Appel depuis TP : procedure TsrUnInst;
;-- Info            : Cette routine ne doit être appelée que si
;--                   TSRCANUNINST() a renvoyé le valeur TRUE
;--
;
tsruninst  proc  near
;
        push ds
        mov  es,call_seg          ;Charge le segment du TSR installé
;
        ;-- Rétablit les gestionnaires d'interruption ---------------
;
        cli                                 ;Inhibe toute interruption
        mov  ax,2508h             ;N° fonction: Fixer gestionnaire INT 8
        mov  ds,es:int8_seg                   ;Rétablit le segment
        mov  dx,es:int8_ofs     ; et l'offset de l'ancien gestionnaire
        int  21h
;
        mov  ax,2509h             ;N° fonction: Fixer gestionnaire INT 9
        mov  ds,es:int9_seg                   ;Restaure le segment
        mov  dx,es:int9_ofs     ; et l'offset de l'ancien gestionnaire
        int  21h
;
        mov  ax,2513h            ;N° fonction: Fixer gestionnaire INT 13
        mov  ds,es:int13_seg                  ;Restaure le segment
        mov  dx,es:int13_ofs    ; et l'offset de l'ancien gestionnaire
        int  21h
;
        mov  ax,2528h             ;N° fonction: Fixer gestionnaire INT 28
        mov  ds,es:int28_seg                  ;Restaure le segment
        mov  dx,es:int28_ofs    ; et l'offset de l'ancien gestionnaire
        int  21h
;
        mov  ax,252Fh             ;N° fonction: Fixer gestionnaire INT 2F
        mov  ds,es:int2F_seg                  ;Restaure le segment
        mov  dx,es:int2F_ofs    ; et l'offset de l'ancien gestionnaire
        int  21h
;
;
        ;-- Libère la mémoire ---------------------------------------
;
        sti                         ;Autorise à nouveau les interruptions
;
        mov  es,es:t_psp     ;CX=segment du PSP du programme résident
        mov  cx,es
        mov  es,es:[ 02ch ]                   ;PSP=seg de l'environment
        mov  ah,49h          ;N° de la fonction: Libérer de la mémoire
        int  21h                        ;Appelle l'interruption de DOS
;
        mov  es,cx                              ;Reprend ES dans CX
        mov  ah,49h          ;N° de la fonction: Libérer de la mémoire
        int  21h                        ;Appelle l'interruption de DOS
;
        pop  ds                         ;récupère DS et BP sur la pile
        ret                                     ;retourne à l'appelant
;
tsruninst  endp                                 ;Fin de la procédure
;
;-----------------------------------------------------------------
;-- TSRSETPTR: Mémorise l'adresse  de la routine qui devra être --------
;--            déclenchée lors d'un appel ultérieur à TSRCALL
;-- Appel depuis TP: procedure TsrSetPtr( offset : word );
;
tsrsetptr  proc   near
;
sframe3   struc                     ;Structure pour accéder à la pile
bp3       dw ?                                      ;Mémorise BP
ret_adr3  dw ?                                 ;Adresse de retour
offset3   dw ?                      ;Offset de la routine à appeler
sframe3   ends                                  ;Fin de la structure
;
frame     equ [ bp - bp3 ]
;
        push bp                                          ;Empiler BP
        mov  bp,sp                          ;Transfère SP en BP
;
        mov  ax,frame.offset3               ;Transfère l'offset en AX
        mov  call_ofs,ax                              ; et le mémorise
;
        pop  bp                             ;Récupère BP sur la pile
        ret  2                              ; et retire les arguments
;
tsrsetptr  endp                                 ;Fin de la procédure
;
;-----------------------------------------------------------------
;-- TSRCALL: Appelle une routine dans l'exemplaire du programme résident
;--          préalablement installé
;-- Appel depuis TP : procedure TsrCall;
;-- Attention : - C'est à dessein que dans cette routine la pile
;--               n'est pas modifiée: ainsi des paramètres pourront être
;--               transmis à la fonction ou procédure à appeler
;--             - Cette procédure doit être FAR car elle est appelée
;--               depuis TP par un pointeur de procédure
;
tsrcall   proc far
;
        ;-- Effectue le changement de contexte vers le programme TP--
        ;--  et appelle la procédure indiquée ----------------------
;
        pop  rip_save               ;Prend note de l'adresse de retour
        pop  rcs_save                                          ;et de
        mov  ds_save,ds                                        ;DS
;
        mov  ah,2fh           ;N° de la fonction: "Lire l'adresse DTA"
        int  21h                        ;Appelle l'interruption de DOS
        mov  u_dta_ofs,bx                ;Sauvergarde l'adresse DTA du
        mov  u_dta_seg,es                        ;programme interrompu
;
        mov  es,call_seg                    ;Charge en ES le segment du
                                          ;programme résident installé"
        mov  ah,50h          ;N° de la fonction: Fixer l'adresse du PSP
        mov  bx,es:t_psp                    ;Lit le segment du PSP
        int  21h                        ;Appelle l'interruption de DOS
;
        mov  ah,1ah           ;N° de la fonction: Fixer l'adresse DTA
        mov  dx,es:t_dta_ofs                ;Offset de la nouvelle DTA
        mov  ds,es:t_dta_seg                ;Segment de la nouvelle DTA
        int  21h                        ;Appelle l'interruption de DOS
;
        mov  ds,es:t_ds                 ;Fixe les registres de segment
        mov  es,es:t_es                         ; pour le programme TP
;
        call [call_ptr]                         ;Appelle la procédure TP
        mov  cs:ret_ax,ax             ;Note le résultat de la fonction
        mov  cs:ret_dx,dx
;
        ;--Changement de contexte: retour au programme TP -------------
;
        mov  ah,1ah                             ;N° fonction: Adresse DTA
        mov  dx,u_dta_ofs     ;Charge l'offset et le segment de la DTA
        mov  ds,u_dta_seg                       ; du programme interrompu
        int  21h                        ;Appelle l'interruption de DOS
;
        mov  es,call_seg                            ;Récupère ES et DS
        mov  ds,ds_save
```

```
        mov  ah,50h             ;N° fonction: Fixer adresse du PSP
        mov  bx,cs                      ;Transfère CS en BX
        sub  bx,10h                 ;Calcule le segment du PSP
        int  21h                ;Appelle l'interruption de DOS

        mov  ax,cs:ret_ax       ;Ramène le résultat de la fonction
        mov  dx,cs:ret_dx
        jmp  [rptr_save]                ;retourne à l'appelant

tsrcall endp                            ;Fin de la procédure

;---------------------------------------------------------------------
;-- DOSACTIF: Détermine grâce à l'indicateur "INDOS" si DOS peut être
;--       interrompu
;-- Entrée:  néant
;-- Sortie:
;:  Indicateur de zéro = 1 : DOS peut être interrompu

dosactif  proc near

        push ds                                 ;Empile DS et BX
        push bx
        lds  bx,daptr           ;DS:BX pointent sur l'indicateur INDOS
        cmp  byte ptr [bx],0            ;Fonction de DOS active ?
        pop  bx                 ;Récupère BX et DS sur la pile
        pop  ds

        ret                             ;Retourne à l'appelant

dosactif  endp

;---------------------------------------------------------------------
;--        Voici les nouveaux gestionnaires d'interruption          --
;---------------------------------------------------------------------

;-- Nouveau gestionnaire de l'interruption 8h (Timer)-----------------

int08   proc far

        cmp  tsrnow,0           ;Faut-il activer le programme résident
        je   i8_end               ;Non, passe à l'ancien gestionnaire

        dec  tsrnow         ;Oui, décrémente l'indicateur d'activation

        ;-- TSR doit être activé mais est-ce possible ? ------------

        cmp  in_bios, 0         ;Interruption disque du BIOS active ?
        jne  i8_end               ;OUI --> pas d'activation possible

        call dosactif                   ;DOS peut-il être interrompu ?
        je   i8_tsr             ;Oui, appelle le programme résident

i8_end: jmp  [int8_ptr]           ;retourne à l'ancien gestionnaire

        ;-- Active le programme résident ---------------------------

i8_tsr: mov  tsrnow,0           ;Le TSR n'attend plus son activation
        mov  tsractif,1   ;le programme résident est maintenant actif
        pushf               ;Simule l'appel de l'ancien gestionnaire
        call [int8_ptr]                 ; par INT 8h
        call start_tsr                  ;Lance le programme résident
        iret                    ;retourne au programme interrompu

int08   endp

;-- Nouveau gestionnaire de l'interruption 09h (clavier)--------------

int09   proc far

        push ax
        in   al,60h                     ;Lit le port du clavier

        cmp  tsractif,0    ;Le programme résident est-il déjà actif ?
        jne  i9_end   ;OUI: appelle l'ancien gestionnaire puis retour

        cmp  tsrnow,0  ;Le programme est-il en attente d'activation ?
        jne  i9_end   ;OUI: appelle l'ancien gestionnaire puis retour

        ;-- Teste la touche d'activation  --------------------------

        cmp  sc_code,128                ;Y a-t-il un scan code ?
        je   i9_ks          ;Non, ne teste que les touches de commande

        cmp  al,128                     ;Oui est-ce un code release ?
        jae i9_end                      ;Oui, pas d'activation

        cmp  sc_code,al         ;Code make à comparer avec le modèle
        jne  i9_end                 ;Pas d'activation si pas le même

i9_ks:  ;-- Teste l'état des touches de commande -------------------

        push ds

        mov  ax,040h                    ;DS sur le segment des variables
        mov  ds,ax                                      ; du BIOS
        mov  ax,word ptr ds:[17h]    ;Indicateur d'état clavier BIOS
        and  ax,key_mask                        ;touche d'activation
        cmp  ax,key_mask            ;bits de la touche d'activation ?
        pop  ds
        jne  i9_end    ;Touche d'activation détectée ? NON --> retour

        cmp  in_bios, 0       ;Interruption disque du BIOS en cours ?
        jne  i9_e1              ;OUI --> pas d'activation possible

        call dosactif           ;Est-il possible d'interrompre DOS ?
        je   i9_tsr               ;Oui, lance le programme résident

i9_e1:  mov  tsrnow,TIME_OUT            ;TSR en attente d'activation

i9_end: pop  ax                                         ;Reprend AX
        jmp  [int9_ptr]       ;Se branche sur l'ancien gestionnaire

i9_tsr: mov  tsractif,1       ;Le TSR va être actif (dans un moment)
        mov  tsrnow,0                   ;Pas de délai de lancement
        pushf
        call [int9_ptr]                 ;Appelle l'ancien gestionnaire
        pop  ax                                         ;Récupère AX
        call start_tsr                  ;Lance le programme résident
        iret                        ;retourne au programme interrompu

int09   endp

;-- Nouveau gestionnaire de l'interruption 13 h (disque(tte))) --------

int13   proc far

        inc  in_bios          ;Incrémente l'indicateur disque du BIOS
        pushf               ;Simule l'appel de l'ancien gestionnaire
        call [int13_ptr]                        ; par INT 13h
        dec  in_bios          ;Restaure l'indicateur disque du BIOS-

        sti                     ;Autorise à nouveau les interruptions
        ret  2                    ;Retourne à l'appelant, mais enlève
                ;en même temps le registre des indicateurs de la pile
int13   endp

;-- Nouveau gestionnaire de l'interruption 28h (DOS idle) -------------

int28   proc far

        cmp  tsrnow,0   ;Programme résident en attente d'activation ?
        je   i28_end                    ;Non, retourne à l'appelant

        cmp  in_bios, 0   ;Oui, mais a-t-on une interruption disque ?
        je   i28_tsr                    ;Oui, donc pas d'activation

i28_end:  jmp  [int28_ptr]          ;retourne à l'ancien gestionnaire

        ;-- Lance le programme résident ----------------------------

i28_tsr:  mov  tsrnow,0                 ;Le TSR n'est plus en attente
        mov  tsractif,1         ;Le TSR va être actif dans un moment
        pushf               ;Simule l'appel de l'ancien gestionnaire
        call [int28_ptr]                ;d'interruption par INT 28h
        call start_tsr                  ;Lance le programme résident
        iret                                    ;Retourne à l'appelant

int28   endp

;-- Nouveau gestionnaire de l'interruption 2Fh (multiplexeur) ---------

int2f   proc far

        cmp  ah,i2f_code  ;Appel de ce présent programme résident ? ?
        jne  i2f_end          ;Non, retourne à l'ancien gestionnaire

        cmp  al,I2F_FCT_0         ;Oui, est-ce la sous-fonction 00h?
        je   i2f_0                      ;Oui, passe à l'exécution

        cmp  al,I2F_FCT_1     ;peut-être est-ce la sous-fonction 01h?
        je   i2f_1                      ;Oui, passe à l'exécution

        iret                                    ;Nein, ignore l'appel

i2f_en':  ;-- Le TSR n'est pas concerné, fait suivre l'appel ---------

        jmp  [int2f_ptr]                ;vers l'ancien gestionnaire

i2f_0:  ;-- Sous-fonction 00: test d'installation ------------------

        xchg ah,al        ;Echange numéros fonction et sous-fonction.
        iret                            ;Retourne à l'appelant

i2f_1:  ;-- Sous-fonction 01: retourne le segment ------------------

        mov  ax,cs                              ;Segment en AX
```

```
        iret                                    ;Retourne à l'appelant
int2F   endp
;-- START_TSR: Active le programme résident ----------------------------
start_tsr proc near
        ;-- Changement de contexte vers le programme TP -------------
        cli                                     ;Inhibe les interruptions
        mov uprg_ss,ss          ;Prend note du segment et du pointeur
        mov uprg_sp,sp                                  : de pile
        mov ss,t_ss                     ;Active la pile du programme TP
        mov sp,t_sp
        sti                                 ;Rétablit les interruptions
        push ax                 ;Sauvegarde les registres du processeur
        push bx                                         ;sur la pile de TP
        push cx
        push dx
        push bp
        push si
        push di
        push ds
        push es
        ;-- Sauvegarde 64 mots de la pile de DOS --------------------
        mov cx,64                                   ;Compteur de voucle
        mov ds,uprg_ss      ;DS:SI pointe sur la fin de la pile de DOS
        mov si,uprg_sp
tsrs1:  push word ptr [si]          ;Transfère un mot de la pile de DOS
        inc si                      ;sur la pile de TP et fait pointer SI
        inc si                                          ;sur le mot suivant
        loop tsrs1                                  ;Traite tous les 64 mots
        mov ah,51h              ;N° de la fonction: Lire l'adresse du PSP
        int 21h                             ;Appelle l'interruption de DOS
        mov u_psp,bx                        ;Mémorise le segment du PSP
        mov ah,2fh              ;N° de la fonction: Lire l'adresse DTA
        int 21h                             ;Appelle l'interruption de DOS
        mov u_dta_ofs,bx                    ;Sauve l'adresse de la DTA
        mov u_dta_seg,es                     : du programme interrompu
        mov ah,50h      ;N° de la fonction: "Fixer l'adresse du PSP"
        mov bx,t_psp                                    ;Saugard. PSP
        int 21h                         ;Appelle l'interruption de DOS
        mov ah,1ah              ;N° de la fonction: Fixer l'adresse DTA
        mov dx,t_dta_ofs                                ;Lit l'offet
        mov ds,t_dta_seg            ;et le segment de la nouvelle DTA
        int 21h                             ;Appelle l'interruption de DOS
        mov ds,t_ds                         ;fixe les registres de segment
        mov es,t_es                                  ; pour le programme TP
        call [Prc_adr]                  ;Appelle la fonction de lancement
        ;-- Changement de contexte vers le programme interrompu -----
        mov ah,1ah                      ;N° fonction: Fixer adresse DTA
        mov dx,u_dta_ofs        ;Charge l'offset et le segment de la DTA
        mov ds,u_dta_seg                        ;du programme interrompu
        int 21h                         ;Appelle l'interruption de DOS
        mov ah,50h                      ;N° fonction: Fixe adresse PSP
        mov bx,u_psp                        ;Seg duPSP du prog interr.
        int 21h                         ;Appelle l'interruption de DOS
        ;-- restaure la pile de DOS---------------------------------
        mov cx,64                                           ;Compteur
        mov ds,uprg_ss                  ;DS:SI=adresse fin de pile DOS
        mov si,uprg_sp
        add si,128                      ;SI au début de la pile de DOS
tsrs2:  dec si                                      ;SI sur mot précédent
        dec si
        pop word ptr [si]       ;Transfère mot de pile TP -> pile DOS
        loop tsrs2                                      ;Traite 64 mots
        pop es                              ;Reprend les registres sauvés
        pop ds                                          ;sur la pile de TP
        pop di
        pop si
        pop bp
        pop dx
        pop cx
        pop bx
        pop ax
        cli                                         ;Inhibe les interruptions
        mov ss,uprg_ss              ;Restaure me pointeur et le segment
        mov sp,uprg_sp                  ;de pile du programme interrompu
        mov tsractif,0          ;Le programme résident n'est plus actif
        sti                         ;Autorise à nouveau les interruptions
        ret                                         ;Retourne à l'appelant
start_tsr endp
;-------------------------------------------------------------------------
CODE    ends                                        ;Fin du segment de code
        end                                             ;Fin du programme
```

32.2.4. Quelques conseils pour terminer

Ce chapitre a montré combien il était simple de développer des programmes en langage évolué, lorsque les routines de base étaient écrites en assembleur. Malgré tout la mise au point de programmes résidents reste un travail délicat, même s'il présente le charme des défis.

Il faut respecter certaines règles qui tiennent au caractère particulier des programmes résidents. Le mieux est de développer d'abord le programme comme un programme ordinaire à lancer à partir de la ligne de commande de DOS ou de l'environnement intégré du compilateur.

Pour effectuer le moins de transformations possibles par la suite, on peut déjà prévoir distinctement la routine d'initialisation et la routine proprement résidente qui sera déclenchée par la touche d'activation.

Contrairement à la version résidente, on appelle explicitement ces routines dans la procédure (ou fonction) principale du programme sans les faire dépendre de la touche d'activation.

On développe et on teste complètement le programme sous cette forme. Ce n'est que lorsqu'il sera sans erreur qu'on entreprendra la transformation en programme résident. En effet la recherche des erreurs d'un programme résident est très difficile voire impossible même avec un débogueur.

La transformation proprement dite est relativement simple, il suffit d'inclure la partie en assembleur et d'en appeler les fonctions. Le détail de l'opération est expliqué dans le cadre des programmes d'exemple qui présentent tous les appels de fonctions nécessaires.

33. Mode protégé, DOS-Extender, DPMI/VCPI

Le mode protégé du 80286 et de ses successeurs s'est inexorablement frayé son chemin dans le monde de DOS, un monde qui par principe ne pouvait lui être qu'hostile. DOS est et reste fondamentalement un système d'exploitation qui fonctionne en mode dit réel, qui n'a rien à voir avec le mode protégé. La même observation s'applique au BIOS du PC qui est en ROM et qui a été conçu pour être exécuté en mode réel: en cas de déclenchemnt du mode protégé le premier appel au BIOS entraîne inévitablement un plantage du système.

Mais les véritables capacités de processeurs comme le 80386 et l'i486 ne se révèlent qu'en mode protégé. Il n'est donc guère étonnant que de nombreux éditeurs de logiciels aient recherché un moyen de rendre opérationnel le mode protégé sous DOS. Il en a résulté des émulateurs EMS, des programmes de gestion de la mémoire, des DOS-Extenders et des gestionnaires d'exploitation multitâche qui à son insu placent DOS sous le contrôle d'un système en mode protégé.

Ce chapitre va décrire le fonctionnement de ces logiciels.

33.1. Le mode protégé

John Crawford, l'un des développeurs en chef de la société Intel, s'est souvent emporté contre l'industrie micro-informatique. C'est sous son autorité que les processeurs 80286, 80386 et i486 ont été pourvus de possibilités riches d'avenir. Mais l'utilisateur n'en a guère profité jusqu'ici faute de système d'exploitation capable de les exploiter. Or le mode protégé des processeurs a justement été inventé pour permettre le développement de systèmes d'exploitation multitâche.

C'est en 1982 déjà que le 80286 a été introduit avec le mode protégé. Mais le monde de DOS était et reste dans l'incapacité d'en tirer profit. DOS en effet est basé sur l'antique mode réel et le système ne serait plus DOS si on le réécrivait pour le mode protégé. Près de dix ans auront passé jusqu'à ce que le mode protégé soit réellement exploité avec l'introduction de la version 32-bits de Windows et l'apparition de la version 2.0 d'OS/2. Dans l'intervalle, il faudra se contenter de solutions intérimaires, telles qu'elles s'expriment par exemple à travers les modes standard et étendu de Windows 3.

Pour vous préparer à l'avenir et pour les besoins des chapitres suivants, le présent chapitre sera consacré au fonctionnement du mode protégé et de son environnement comparés au mode réel. Nous supposerons que vous êtes familier de ce dernier.

33.1.1. Caractéristiques des systèmes multitâche

Avant d'étudier le mode protégé, il est indispensable d'approfondir les conditions de son introduction. En d'autres termes, il s'agit d'examiner la structure et les besoins des systèmes d'exploitation multitâche en liaison avec les possibilités du processeur sous-jacent.

Lorsqu'on parle de système multitâche, la plupart des utilisateurs pensent à l'exécution simultanée de plusieurs programmes (appelés tâches). Mais l'exécution en parallèle ne concerne pas seulement des programmes indépendants mais également des modules à l'intérieur de chacun de ces programmes. Prenez l'exemple d'un traitement de texte qui procède à une impression en tâche de fond tout en autorisant la poursuite de la saisie. Nous avons là deux tâches au sein d'un même programme.

Cette exécution en parallèle suppose une coexistence harmonieuse des tâches en cours, chacune d'elles se gardant bien d'empiéter sur la mémoire gérée par l'autre car aucune n'est seul maître à bord de la mémoire centrale. Les tâches doivent être isolées les unes des autres, le système d'exploitation devant lui aussi être protégé contre l'incursion des programmes et de leurs différentes tâches. C'est pour cela qu'on qualifie le mode de "protégé".

Caractéristiques d'un processeur indispensables au support d'un système multitâche

① Protection réciproque des tâches et du système d'exploitation contre l'écriture dans des zones de mémoire étrangères

② Gestion de la commutation des tâches, notamment pour sauver et restaurer l'environnement d'une tâche

③ Définition d'un statut privilégié pour le système d'exploitation permettant l'exécution de certaines instructions machine réservées

④ Soutien d'une gestion de la mémoire virtuelle

Nous étudierons plus loin l'isolement des zones mémoire. Mais retenons dès à présent que cet isolement est la première condition requise pour qu'un système d'exploitation puisse être multitâche, même si le processeur n'a pas besoin de spécialisation sous ce rapport.

Multitâche

L'exécution simultanée de plusieurs programmes dans un système multitâche demeure une illusion si l'électronique sous-jacente ne repose pas sur plusieurs processeurs

travaillant en parallèle. Dans la micro-informatique traditionnelle, les PC ne disposent évidemment que d'un seul processeur, abstraction faite de quelques rares cas particuliers d'ordinateurs multiprocesseurs. Le mode multitâche revient donc à répartir le temps de traitement sur plusieurs tâches de façon qu'elles ne consomment qu'une fraction de seconde.

Par exemple avec trois tâches, la première prendra 1/3 de seconde, la deuxième 1/3 de seconde et la troisième le tiers restant. Une fois la boucle bouclée, le contrôle est rendu à la première tâche, et ainsi de suite.

C'est ce qu'on appelle le multitâche préemptif, fondé sur le découpage du temps en petites tranches ("time-slicing"). Le système intervient sans ménagement sur chacune des tâches en stoppant brutalement son déroulement pour donner la main à une autre. Pour la tâche interrompue, le processus doit être transparent car elle ne peut pas savoir d'avance à quel moment aura lieu l'interruption.

Le système d'exploitation est donc responsable pour que l'exécution se fasse sans bavure: après chaque interruption les choses doivent reprendre comme si rien ne s'était passé. L'environnement d'une tâche donnée doit donc être minutieusement sauvegardé au moment de l'interruption et être rétabli par la suite. Les données concernées sont les ressources système telles que la mémoire et les fichiers traités mais aussi les registres du processeur qui ne doivent pas se trouver altérés.

La sauvegarde et la restauration de l'environnement constituent une activité essentielle pour un système multitâche. Il n'est pas absolument nécessaire que le processeur soit accompagné de périphériques spéciaux mais lorsque c'est le cas les choses se trouvent accélérées par rapport à un traitement purement logiciel. Il ne faut pas oublier que l'illusion de simultanéité des exécutions exige des commutations de tâches nombreuses et rapprochées, ce qui exclut que le système passe trop de temps à assurer précisément la gestion de ces commutations.

Le soutien de ces commutations est la deuxième exigence requise pour faire fonctionner un système multitâche.

Traitement privilégié du système d'exploitation

La troisième exigence concerne la définition d'un statut privilégié pour le système d'exploitation qui doit pouvoir exécuter certaines instructions machines interdites aux tâches. C'est ainsi que les instructions qui servent à commuter les tâches ou à changer le mode de fonctionnement du processeur ne doivent être qu'à la seule disposition du système. Si une tâche prenait l'initiative de déclencher le mode réel alors que le processeur tourne en mode protégé, les conséquences seraient évidemment catastrophiques.

Mais tout comme l'accès à des zones interdites de la mémoire, le non-respect de ces dispositions ne doit pas entraîner le plantage du système tout entier, mais seulement l'appel d'une commande système spéciale. S'il n'est pas possible de corriger l'erreur à ce stade, le système doit mettre fin au programme litigieux et le retirer de la mémoire. Les autres programmes, qui n'y sont pour rien dans l'incident, ne doivent pas être pénalisés pour autant. Une gestion des erreurs programmable constituera ainsi la quatrième exigence d'un système multitâche.

Mémoire virtuelle

Pour terminer, le processeur devra apporter son aide à la gestion de la mémoire. Lorsqu'augmentent le nombre et la complexité des programmes exécutés simultanément, les besoins en mémoire s'accroissent et ne peuvent pas toujours être satisfaits par la mémoire physique présente. Le système d'exploitation doit donc être en mesure de mettre à la disposition des programmes plus de mémoire qu'il n'en existe réellement. En fait, l'intégralité de la mémoire n'est pas constamment utilisée, et il est possible de stocker sur disque les parties inactives jusqu'à ce qu'elles soient de nouveau réclamées par un programme. Pour déterminer les zones de mémoire qui ne sont pas indispensables et qui peuvent être temporairement transférées sur disque, ainsi que celles qui doivent être rapatriées de toute urgence, il est préférable de s'appuyer sur certaines caractéristiques du processeur, sans quoi on est obligé de passer par des artifices logiciels coûteux en temps d'exploitation.

Les sections suivantes vont montrer comment appliquer pratiquement ces exigences. Nous commencerons par une description du mode protégé du 80286. Puis nous étudierons le mode protégé des processeurs 80386 et i486, qui sont compatibles avec le 80286 mais présentent des améliorations très importantes. Vous verrez que les processeurs mentionnés sont très puissants en mode protégé, mais que la programmation de ce mode n'est pas simple et doit s'appuyer sur une logique interne très complexe.

En contrepartie de cette complexité, il faut payer un certain prix car les processeurs d'Intel fonctionnent plus lentement en mode protégé qu'en mode réel. Pourquoi ? C'est ce que nous verrons un peu plus loin.

33.1.2. Le mode protégé du 80286

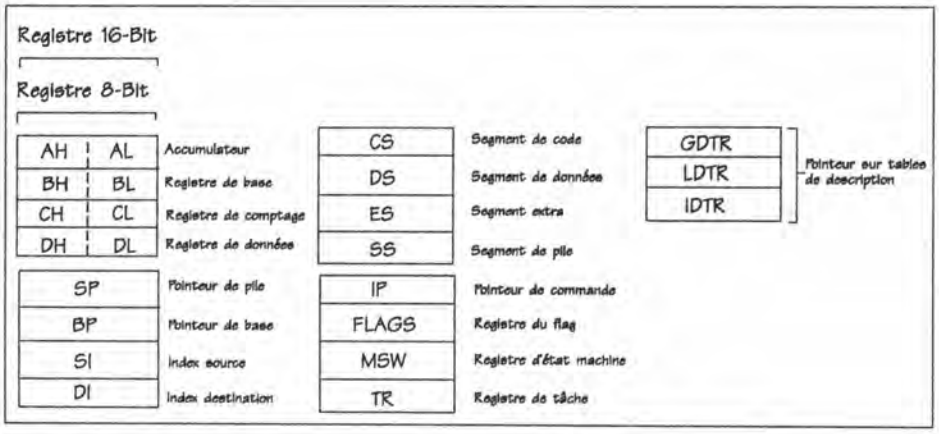
Les programmeurs familiarisés avec l'assembleur en mode réel sous DOS mais peu au courant du mode protégé se demandent souvent dans quelle partie de l'assembleur se révèle la différence entre les deux modes. Est-ce que par exemple en mode protégé le processeur ne connaîtrait plus des instructions aussi banales que JMP, PUSH ou MOV, est-ce que d'autres instructions viennent les remplacer, est-ce que les instructions habituelles fonctionnent différemment ? Les programmeurs expriment ainsi leur inquiétude de voir surgir devant eux un autre processeur à partir du moment où ils déclenchent

le mode protégé. Ne se trouveraient-ils pas entraînés dans un monde où les certitudes acquises se dérobent, où toutes choses ont un comportement nouveau ?

Mais évidemment il n'en est rien. Toutes les instructions du mode réel restent valables en mode protégé et seules quelques instructions en nombre limité viennent s'y rajouter. S'il est vrai que certaines instructions fonctionnent différemment, les nouveaux mécanismes sont la plupart du temps invisibles pour le programmeur qui travaille en assembleur. Car les modifications s'effectuent sur le plan interne. Il s'agit principalement de l'adressage de la mémoire, autrement dit du chargement des adresses de segment dans le registre de segment, de la formation des pointeurs FAR et de leur codage dans les instructions de branchement et d'appel aux sous-programmes. Dans ce domaine le mode protégé suit des voies toutes nouvelles. Mais nous allons commencer par jeter un coup d'oeil sur les registres du 80286, et plus spécialement sur ceux qui gèrent le mode protégé.

Les registres du 80286

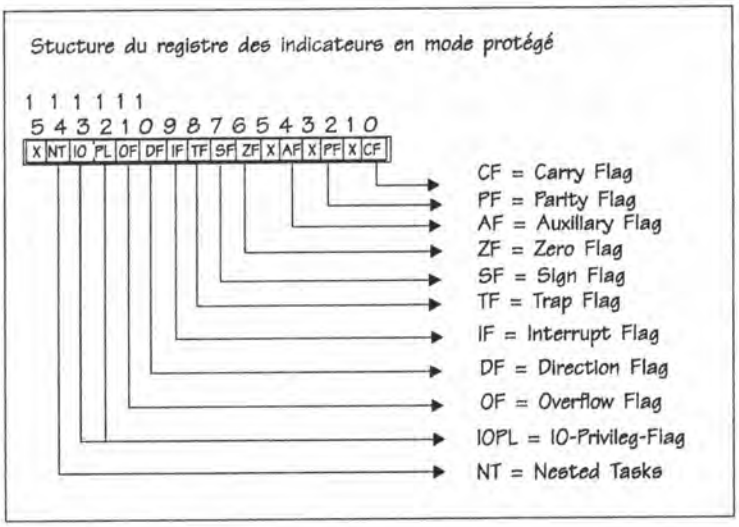
Par rapport aux registres du 8086, de nouveaux registres ont été ajoutés comme vous le voyez sur la figure suivante. Ces registres sont également disponibles en mode réel mais ils n'ont alors aucune signification, ce qui explique que vous ne vous en soyez jamais préoccupé. Certains d'entre eux, par exemple GDTR, LDTR et IDTR doivent être chargés avant tout déclenchement du mode protégé. Le système se plante sans délai s'ils ne sont pas correctement initialisés lors du fonctionnement en mode protégé. Le rôle précis de ces registres dans la programmation du mode protégé sera étudié dans les sections qui suivent.



Les registres du 80286 en mode protégé

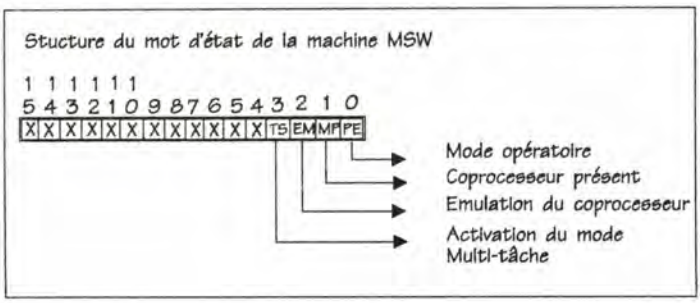
Les programmeurs qui pratiquent l'assembleur connaissent bien le registre des indicateurs qui joue un rôle clé pour beaucoup d'instructions (comparaisons, branchements).

Sa signification est la même en mode protégé qu'en mode réel, la position des différents bits étant conservée. Il s'y ajoute cependant deux bits supplémentaires appelés IOPL et NT. L'indicateur IOPL sera étudié dans la partie consacrée à l'accès au matériel, tandis que l'indicateur NT sera examiné au moment de l'étude de la commutation de tâche.



Déclenchement du mode protégé

Le registre appelé MSW (Machine Status Word = mot d'état machine) est aussi une sorte de registre d'indicateurs car les différents bits qui le constituent décrivent l'état ("statut") du processeur. Des seize bits présents le 80286 n'en exploite cependant que quatre, ceux dont le poids est le plus faible. Les bits 1, 2 et 3 assurent la liaison avec un coprocesseur arithmétique, ce qui ne nous intéresse pas dans le cadre de ce chapitre. Mais le bit 0 est autrement important pour nous: il constitue en effet la clé d'accès au mode protégé. Au moment de l'initialisation du processeur, il est mis automatiquement à zéro, ce qui signifie que le mode par défaut est le mode réel. Mais si un programme charge la valeur 1 dans ce bit, le processeur fonctionne aussitôt en mode protégé.



Hélas il n'est pas possible de ramener le 80286 en mode réel en remettant ce même bit à zéro. Cette particularité a causé bien du tracas aux développeurs et elle a alimenté d'innombrables rumeurs et anecdotes. En effet les programmes DOS comme les DOS-Extenders et les émulateurs EMS ont toujours besoin, à un moment ou à un autre, de revenir au mode réel pour que l'utilisateur puisse continuer son travail sous DOS.

Comme le 80286 n'a pas d'instruction pour remettre à zéro le bit PE du MSW, on en est réduit à des solutions du type radical, par exemple en déclenchant une réinitialisation complète (Reset) du processeur ce qui remet effectivement en route le mode réel. Normalement ce genre de mesure entraîne aussi la réinitialisation du BIOS en ROM ainsi que l'effacement intégral de la mémoire vive: il s'agit donc de contrecarrer ces débordements en mettant en service des astuces plus ou moins heureuses.

Mais pourquoi le 80286 ne permet-il pas de revenir au mode réel ? En fait, il existe à ce sujet de nombreuses spéculations. Selon certains, les responsables du développement du 80286 ont simplement oublié l'instruction. D'autres prétendent que les développeurs du 80286 ont été tellement enthousiasmés par le mode protégé qu'ils ne pensaient pas que quelqu'un éprouve un jour le besoin de repasser en mode réel. Selon une troisième rumeur, l'instruction de retour au mode réel existerait bel et bien mais elle ne fonctionnerait pas correctement, ce qui explique l'absence de documentation à son sujet.

Quoi qu'il en soit, un retour au mode réel est quand même possible mais au prix d'une complication extrême et d'une énorme perte de temps.

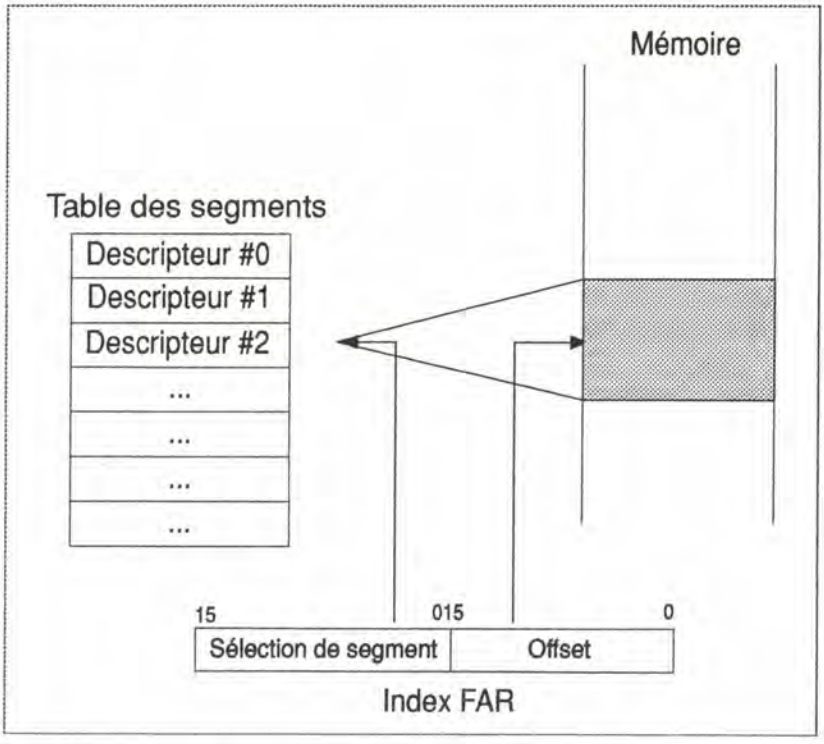
Le programmeur désireux de travailler en mode protégé dispose d'un autre registre spécialisé appelé TR pour Task Register. Comme son nom l'indique, il assure la gestion et la commutation des tâches. Nous l'étudierons dans la partie précisément consacrée à la commutation des tâches.

La gestion de la mémoire en mode protégé

En mode réel le programmeur dispose de l'intégralité de l'espace d'adressage de 1 Mo. Une fois qu'il a chargé dans l'un des quatre registres de segment une adresse de segment comprise entre 0000h et FFFFh, il peut librement fouiller dans la partie résidente de COMMAND.COM, effacer des variables appartenant au BIOS ou modifier des entrées dans la table des vecteurs d'interruption: le plantage est presque toujours garanti. En mode protégé il faut renoncer à ce genre de distraction car le simple chargement incontrôlé d'une adresse de segment dans l'un des registres de segment conduit déjà à une protestation du processeur. Il est vrai qu'il ne se mettra pas en grève mais il générera un appel dit d'exception dont nous parlerons plus loin. Le système d'exploitation reprendra le contrôle et il punira le programme qui s'est mal comporté en y mettant fin. C'est cela, le mode protégé !

Un tel comportement s'explique par le fait que les adresses de segment ont toujours 16 bits mais en mode protégé elles ne représentent plus des emplacements physiques. Les

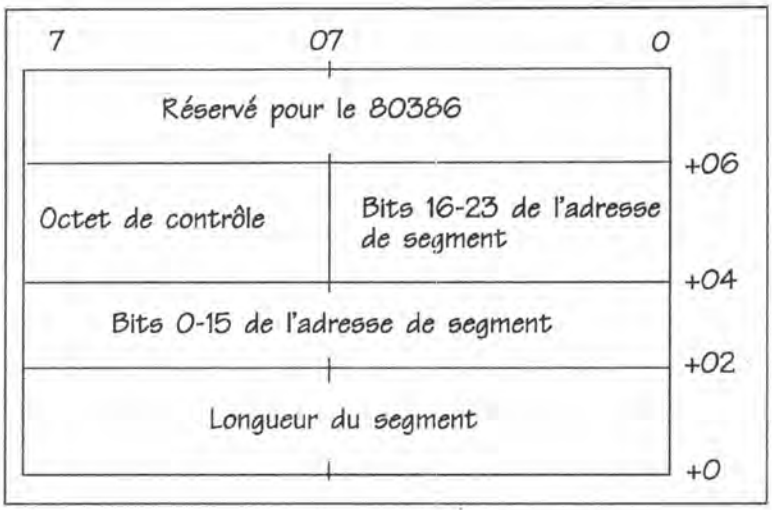
adresses sont données sous forme de sélecteurs qui sont des indices renvoyant à un tableau dans lequel se trouvent décrits les segments mémoire. Ce n'est qu'en consultant ce tableau des descripteurs de segments que le processeur prend connaissance de l'adresse de base des segments. Il lui ajoute l'offset donné sous forme de pointeur FAR pour obtenir l'adresse finale. Notez que l'adresse de base n'est pas multipliée par 10h comme c'est le cas dans le calcul en mode réel. En mode protégé, les segments mémoire peuvent donc débuter à n'importe quel emplacement et pas seulement aux emplacements multiples de 16.



Accès mémoire en mode protégé avec sélecteur de segment et offset

La table des descripteurs associe une adresse virtuelle (sélecteur de segment:offset) à une adresse linéaire (adresse de base issue du sélecteur + offset). Pour le 80286, l'adresse linéaire est en même temps l'adresse physique, c'est-à-dire en fin de compte l'adresse concrète finale. Avec les processeurs 80836 et 80486 le système est plus compliqué. L'adresse linéaire doit encore subir une transformation avant de livrer l'adresse physique. Mais nous verrons ce point plus tard.

La figure précédente ne mentionne pas la structure des descripteurs de segment à l'intérieur de la table des descripteurs. Chaque descripteur contient en effet d'autres informations que la simple adresse de départ. La figure suivante montre la structure complète d'un descripteur.



Structure d'un descripteur de segment

Vous voyez qu'un descripteur s'étend sur huit octets et est divisé en plusieurs champs. On y trouve d'abord la longueur du segment car contrairement à ce qui se passe dans le mode réel un segment n'occupe pas forcément 64 Ko. Sa longueur peut aller de 1 à 64 Ko.

Les trois octets suivants donnent l'adresse de début (de base) du segment dans la mémoire centrale. Alors qu'en mode réel les adresses sont codées sur 20 bits, on dispose maintenant de 24 bits. La mémoire physique adressable passe donc de1 Mo à 16 Mo.

Le fait que la situation d'un segment soit référencée dans le descripteur, et non par les pointeurs FAR qui résultent de l'accès à ce segment, présente de nombreux avantages pour gérer efficacement la mémoire. L'exécution simultanée de plusieurs programmes se caractérise en effet par l'allocation et la libération incessante de zones de mémoire. Au fur et à mesure du déroulement des programmes, la mémoire se trouve de plus en plus fragmentée. Pour prévenir cette fragmentation, il est nécessaire de procéder à des déplacements de segments. En mode réel, lorsqu'un segment est déplacé, toutes les références à ce segment situées dans les programmes en cours doivent être modifiées. En mode protégé, il suffit d'ajuster la table des descripteurs en y inscrivant la nouvelle adresse de base du segment déplacé. Toute référence au segment sera automatiquement à jour lorsqu'un programme fera appel à la table des decsripteurs.

A la suite de l'adresse de base du segment, le descripteur contient un octet appelé registre des indicateurs et qui représente divers indicateurs sur lesquels nous allons nous pencher dans un instant. Le dernier champ est un mot réservé à l'usage du 80386 et de ses successeurs et qui doit être à zéro pour un 80286. Nous étudierons ce point dans la partie consacrée à la programmation des processeurs 80386 et i486 en mode protégé.

Les différents types de segments

Le mode protégé connaît trois types de segments : les segments de données, les segments de code et les segments système. Les segments de données sont destinés à recevoir des données et à être relus, il n'est pas possible d'y exécuter des programmes. Inversement les segments de code sont réservés à des instructions exécutables. Le troisième type, les segments système, recouvre en fait plusieurs catégories de segments essentiellement destinés à gérer le mode protégé. Il en sera question un peu plus loin.

En plus du type de segement, le registre des indicateurs contient d'autres attributs dont la signification dépend en partie du type de segment. C'est ainsi que dans le cas d'un segment de code un bit spécial indique si le contenu du segment peut être lu. Dans le cas d'un segment de données, le même bit indique une protection contre l'écriture (statut Read Only).

Indépendamment du type de segment, le registre des indicateurs contient un bit de présence qui facilite la programmation de la gestion de mémoire virtuelle. Si le système d'exploitation expatrie un segment en le stockant sur disque, il doit obligatoirement mettre ce bit à 0 pour que le processeur soit prévenu. Ce dernier signalera une erreur et appellera une routine de rechargement de la mémoire centale avant de poursuivre l'exécution du programme.

Dans le même ordre d'idées, il existe un bit d'accès qui est mis à 1 lors de chaque accès à un segment. Lorsque le gestionnaire de mémoire virtuelle se voit contraint de retirer des segments de la mémoire pour y ménager un peu de place, il sera amené à sélectionner les segments qui n'ont pas été utilisés ces derniers temps: ce sont ceux dont le bit d'accès est à 0.

Niveaux de privilège

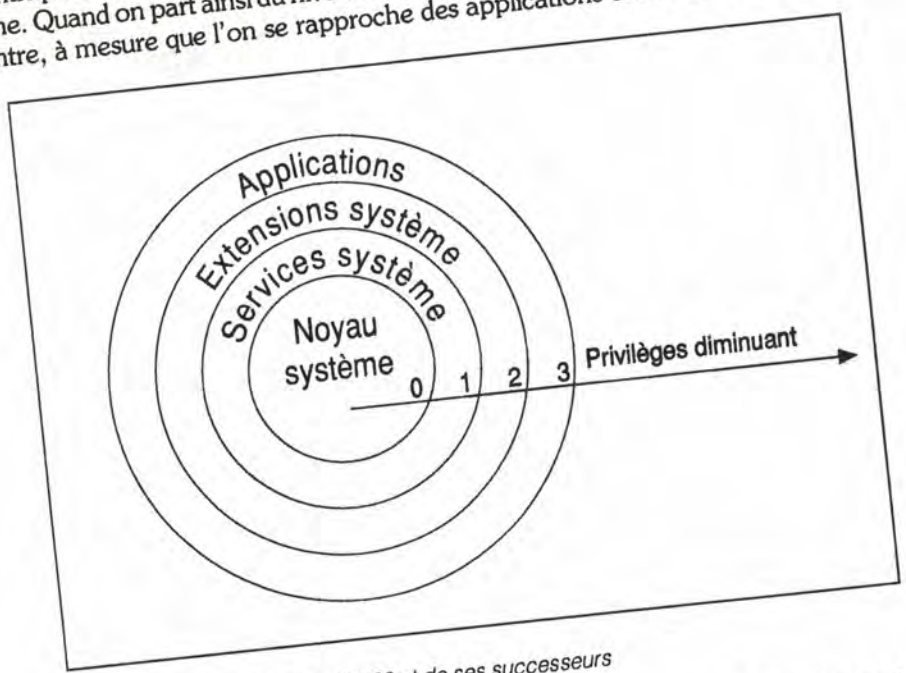
Pour isoler les applications du système d'exploitation, le processeur connaît 4 niveaux de privilège en mode protégé. Ces niveaux n'affectent pas seulement l'exécution de certaines instructions, ils servent aussi à réglementer l'accès aux segments mémoire. Ils se caractérisent par un numéro compris entre 0 et 3, le niveau 0 représentant le privilège maximal. S'il s'agissait simplement de faire la distinction entre applications et système, deux niveaux suffiraient. Mais le systèmle d'exploitation lui-même est composé de plusieurs modules qui n'ont pas la même importance (seul DOS ne connaît pas ces raffinements). Le privilège le plus élevé est attribué au noyau du système qui surveille la gestion de la mémoire et la commutation des tâches. Le niveau de ce module est donc 0.

Le niveau 1 concerne les différentes fonctions système appelées par les applications et les utilitaires d'exploitation. Les primitives de gestion des fichiers, les routines d'affichage et les drivers d'imprimantes se rattachent à ce même niveau de privilège.

Au niveau suivant, qui porte le numéro 2, on trouve les extensions du système d'exploitation qui reposent sur les fonctions de niveau 1. Dans le cas du système OS/2, il s'agit par exemple du serveur SQL ou du LAN Manager (gestionnaire de réseau local).

Au privilège le plus bas, c'est-à-dire de niveau 3, se situent les applications qui se déroulent sous le contrôle du système d'exploitation. C'est justement parce qu'ils sont moins privilégiés que le système qu'ils peuvent être contrôlés par ce dernier (et non vice-versa).

La hiérarchie des niveaux de privilège est généralement illustrée sous forme de cercles concentriques, comme le montre la figure. Au coeur de l'ensemble se tient le noyau du système. Quand on part ainsi du niveau zéro, les privilèges diminuent lorsqu'on s'éloigne du centre, à mesure que l'on se rapproche des applications et de l'utilisateur.

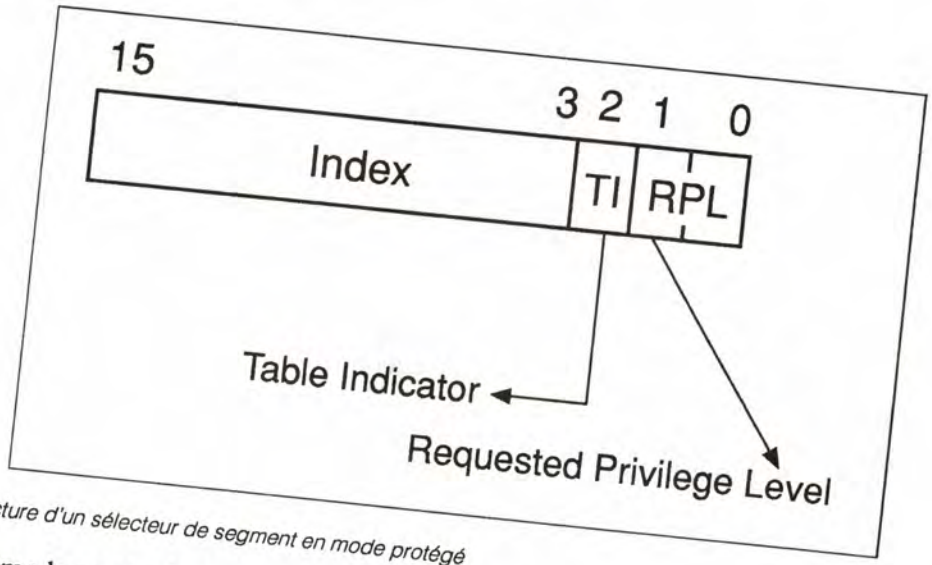


Les quatre niveaux de privilège du 80286 et de ses successeurs

Les niveaux de privilège jouent un rôle crucial lorsque des accès mémoire portent sur des segments étrangers ou lorsqu'une exécution donne lieu à un changement de segment.

Il faut alors éviter qu'une application vienne par exemple mettre son nez dans le noyau du système ou que son exécution soit transférée à un autre point d'entrée.

Le processeur effectue alors une comparaison des niveaux de privilège et signale une erreur lorsqu'une tâche essaye d'accéder à un segment plus prioritaire. Les bases de cette comparaison sont stockées dans les sélecteurs et les descripteurs de segments qui contiennent tous deux une information sur le niveau de privilège.

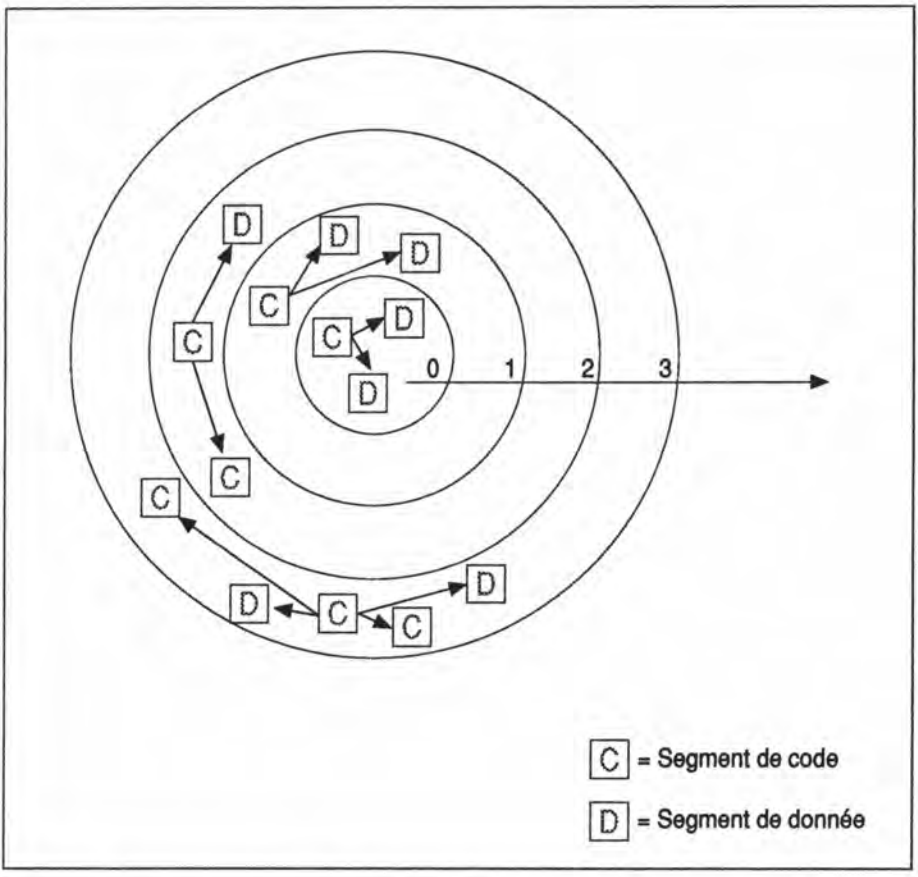


Structure d'un sélecteur de segment en mode protégé

Comme le montre la figure, les deux bits inférieurs de chaque sélecteur de segment contiennent le RPL = "Requested Privilege Level".

Par ailleurs le processeur dispose encore de deux autres sources d'information appelées CPL et DPL. La signification de ces indicateurs est donnée par la table suivante:

DPL (Descriptor Privilege Level)
Niveau de privilège de la tâche en cours d'exécution. Il est tiré du descripteur de segment de code courant. Le sélecteur de ce descripteur de segment est issu du registre CS.
RPL (Requested Privilege Level)
Niveau de privilège mémorisé à l'intérieur d'un sélecteur. Il correspond toujours au niveau de privilège du segment référencé par le descripteur.
On a donc : RPL= DPL du segment adressé



Accès autorisés aux données et au code

En principe on peut dire que l'accès aux données et aux instructions (branchements et appels de sous-programmes) est autorisé lorsque

```
CPL=DPL
```

autrement dit le segment de code doit avoir le même niveau de privilège que les instructions ou les données adressées. Si les niveaux diffèrent, le système applique diverses règles selon la nature des accès.

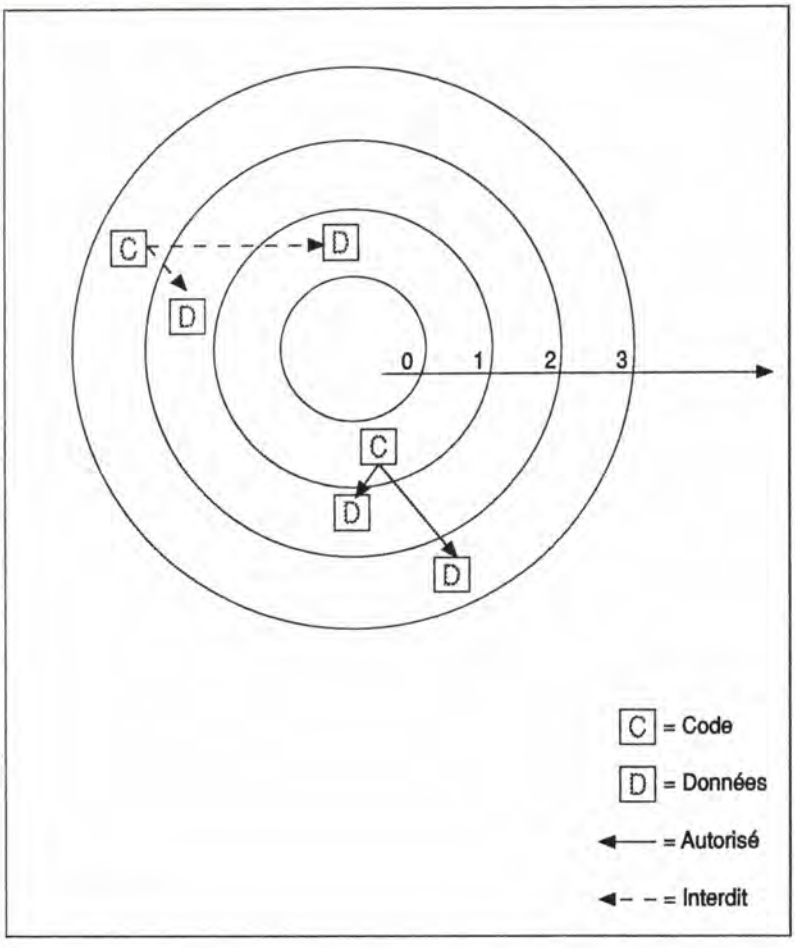
Accès aux données

Les accès aux données suivent la règle:

```
CPL <= DPL
```

Une tâche ne peut pas exploiter un segment de données qui possède un privilège supérieur au sien. Une application est donc dans l'impossibilité d'accéder aux segments de données du système d'exploitation, mais inversement le système peut par exemple déposer des informations dans un buffer géré par une application.

Cette règle entre en jeu à partir du moment où un sélecteur référençant un segment de données est chargé dans l'un des registres DS ou ES. Si elle est violée, le processeur arrête l'exécution de la tâche en cours et déclenche une exception en appelant une routine de gestion spécialisée dans le traitement de cette erreur.



Accès aux données interdits et autorisés en mode protégé

Accès au code et portes d'appel (call gates)

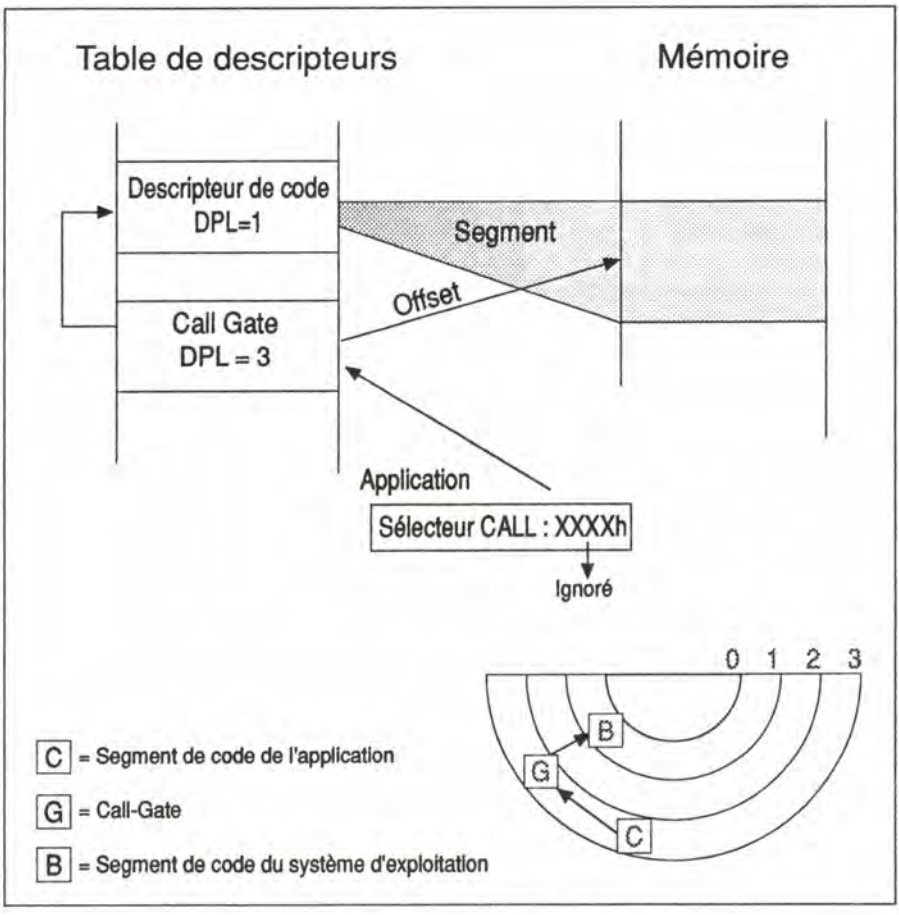
Le processeur est encore plus restrictif pour les accès au code, c'est-à-dire dans le traitement des instructions JMP et CALL. Les branchements et appels de sous-programmes ne sont permis que si le segment de destination se trouve au même niveau de privilège que le code appelant. Cette règle a évidemment pour but d'empêcher des exécutions incontrôlées.

Mais alors comment une application peut-elle faire appel aux fonctions du système d'exploitation, ces dernières étant par définition situées à un niveau de privilège supérieur ? La réponse est donnée par des instruments spécialement créés à cette intention, les call-gates ou portes d'appel. Ce sont des descripteurs de segment spéciaux (descripteurs système) qui occupent cependant huit octets comme les autres descripteurs et se rangent comme eux dans la table des descripteurs. Mais contrairement aux descripteurs de segments de données et de code, il ne définissent pas un segment mémoire mais un point d'entrée dans une routine dont le segment de code peut avoir un privilège de plus haut rang que l'appelant.

Ce segment est déterminé sous la forme d'un sélecteur tout à fait banal à l'intérieur du descripteur de la porte d'appel mais il doit évidemment s'agir d'un segment de code. Mais en plus de l'adresse du segment, le sélecteur indique aussi l'offset, c'est-à-dire le point d'entrée dans la routine. On évite ainsi qu'un branchement conduise à l'exécution de n'importe quelle partie du code segment. Sinon il serait trop facile de violer les principes des règles d'accès.

Comme la porte d'appel mémorise l'offset de la routine, l'offset indiqué dans l'instruction CALL proprement dite perd sa signification: il est alors simplement ignoré et le sélecteur seul dirige l'accès à la porte d'appel. Ceci n'empêche pas que la réglementation des privilèges reste valable.

Sous ce rapport la porte d'appel est traitée comme un segment de données. Elle ne peut être adressée que si son DPL a un privilège inférieur ou égal à celui de l'appelant (c'est-à-dire un numéro supérieur): CPL <=DPL. En conséquence, le système d'exploitation doit mettre ses portes d'appel au niveau 3 pour les ouvrir aux applications.



Déclenchement d'une routine système par l'intermédiaire d'une porte d'appel

Un système d'exploitation se voit donc contraint d'offrir des portes d'appel pour que des programmes externes puisse profiter de ses fonctions. Les programmes ne connaissent jamais l'adresse des routines appelées mais peuvent lire les sélecteurs de segment concernés.

Dans le même ordre d'idées, il existe des segments spéciaux appelés conforming segments (segments adaptables). Ce sont des segments de code ordinaires dont le descripteur se caractérise simplement par un bit spécial du registre des indicateurs. Les segments de code ainsi repérés adaptent automatiquement leur privilège à celui du programme appelant.

Lorsqu'une application appelle par exemple une fonction système mémorisée dans un segment adaptable, le code concerné prend le niveau de privilège de l'appelant, passant donc de 0 (niveau système) à 3 (niveau de l'aplication). Ce mécanisme revient à

considérer certaines fonctions système comme des prolongements purs et simples des applications, sans privilège particulier.

Tables de descripteurs locales et globales

Jusqu'ici nous avons évoqué "une" table des descripteurs dans laquelle le processeur mémorisait différents segments. En réalité le mode protégé permet de gérer jusqu'à 8193 tables de descripteurs : une table globale et 8192 tables locales.

A un moment donné, cependant, seules deux tables peuvent être actives : la table globale et une table locale. Chaque tâche possède sa propre table locale qui change donc à chaque commutation. Les deux tables sont désignées sous le nom de "Global Descriptor Table" et "Local Descriptor Table". Elles sont incarnées par les registres GDTR et LDTR (qui n'existent qu'à partir du 80286). Ces registres ne peuvent être chargés que par le noyau du système d'exploitation par l'intermédiaire des instructions LGDT et LLDT. Si une tâche dont le privilège n'est pas nul appelle ces instructions, le processeur l'interrompt en émettant une exception qui déclenche une routine d'erreur.

L'instruction GDTR a pour effet de charger l'adresse physique et la longueur des tables de descripteurs globales. L'indication de la longueur permet au processeur de connaître le nombre de descripteurs de la table (longueur / 8) et de surveiller son accès.

Pour savoir si un sélecteur se réfère à la table globale ou à la table locale, le processeur examine le bit TI. Si ce bit est nul, la table concernée est la table globale, sinon c'est la table locale.

Dans la plupart des cas, une application utilisera la table locale, car le système y dépose les descripteurs des segments de données et de code de chaque tâche. Mais cette table ne contient pas les portes d'appel qui servent à déclencher les fonctions système : leur nombre est en effet trop élevé, il serait trop dispendieux de mémoriser plusieurs centaines de ces portes dans chaque table locale.

C'est la table globale qui abrite les portes d'appel. Elle contient par ailleurs les segments de code et de données du système d'exploitation ainsi que divers autres descripteurs système. Parmi ces derniers se trouvent les descripteurs des segments mémoire qui contiennent les tables de descripteurs locales. Contrairement au registre GDTR qui héberge l'adresse physique des tables globales, le registre LDTR ne mémorise qu'un seul sélecteur qui doit référencer un descripteur LDT dans la table des descripteurs globale. Il est alors très simple d'installer une nouvelle table locale en cas de commutation de tâche, il suffit de mettre dans le registre LDTR le sélecteur de la nouvelle table locale.

Espace mémoire adressable

Pour calculer la taille de la mémoire virtuelle adressable, il faut multiplier le nombre maximal de descripteurs par la taille maximale des segments, c'est-à-dire 64 Ko. Quel est donc le nombre de descripteurs qui peuvent être mémorisés dans la table des descripteurs globale et dans les différentes tables des descripteurs locales ? Dans les deux cas la réponse est 8192, et ce pour deux raisons convergentes.

D'abord les tables locales et globales sont des segments qui ne peuvent dépasser 64 Ko dans le cadre d'un processeur 80286. Il se trouve que 64 Ko ne permettent pas de stocker plus de 8192 descripteurs car chacun d'eux occupe 8 octets.

Mais par ailleurs le numéro d'un descripteur à l'intérieur d'un sélecteur est codé sur 13 bits, ce qui impose également une limite de 8192 descripteurs (numérotés de 0 à 8191).

Supposons que la table globale contienne 8192 descripteurs de tables locales et que chacune de ces dernières contienne à son tour 8192 descripteurs référençant 64 Ko de code ou de données.

La mémoire virtuellement accessible est alors de:

```
8192 * 8192 * 64 Ko = 1 Gigaoctet
```

Il s'agit là d'un résultat qui exprime une limite théorique.

Comment accéder à un descripteur ?

Quelle que soit la complexité des liaisons entre sélecteurs et descripteurs, entre les tables locales et globales, en pratique le développeur d'applications n'en est guère affecté. Ce qui est important, c'est de ne pas utiliser de sélecteur fantaisiste dans les instructions de branchement et de ne pas charger n'importe quoi dans les registres de segment. mais après tout d'où proviennent les sélecteurs utilisés dans une application ?

En principe ils viennent tous directement ou indirectement du système d'exploitation, car une application ne peut pas accéder directement à la table des descripteurs globale. Il lui est interdit d'ajouter de nouveaux descripteurs à la table existante ou de modifier ceux qui s'y trouvent déjà. C'est là une tâche réservée au noyau du système qui possède le privilège 0 et qui est seul habilité à manipuler les tables.

Le système d'exploitation est également responsable de la création des tables locales pendant le processus de chargement d'une application. Il mémorise alors les descripteurs des segments de données et de code nécessaires à l'application. Les segments créés n'occupent pas systématiquement 64 Ko. la mémoire allouée est ajustée aux besoins exacts de chacun. Si par suite d'une erreur de programmation l'application essaye d'accéder à des données ou des instructions suitées au-delà des segments prévus, une

exception vient interrompre la tâche et une routine système appropriée se déclenche aussitôt.

Mais il ne suffit pas de créer les segments en y associant des descripteurs car les programmes se réfèrent eux-mêmes aux différents segments. Prenons par exemple la séquence suivante extraite d'un programme en assembleur:

```
MOV AX, seg donnnees
MOV DS,AX
```

Le même genre de problème se pose lorsque par exemple dans un programme écrit en C un pointeur FAR sur une fonction située dans un autre segment de code est transmis à une fonction. Le sélecteur du segment visé doit être disponible : par rapport à l'exemple précédent il s'agit simplement d'un segment de code au lieu d'un segment de données.

Un système d'exploitation multitâche va se comporter comme le bon vieux DOS : il va citer dans l'en-tête du fichier EXE (ou du fichier exécutable quel que soit son nom) les adresses des instructions ou des variables qui se réfèrent à des segments de code ou de données. La table correspondante permet au chargeur du système d'exploitation d'écrire directement les sélecteurs des segments adressés dans le code ou les variables concernées. Le principe est le même pour les appels des fonctions système. Dans ce cas, ce ne sont plus des sélecteurs de segments de données ou de code qui sont nécessaires mais des portes d'appel.

Par ailleurs de nombreuses fonctions système vous retourner des sélecteurs comme résultat de leur activité, surtout les fonctions de gestion de la mémoire. Dans ce cas les sélecteurs peuvent être traités comme des adresses de segment ordinaires faisant partie d'un pointeur FAR. Il est évidemment interdit de manipuler ces adresses comme on le ferait sous DOS, sinon on risque de se trouver devant un sélecteur qui référence un descripteur fantaisiste ou inexistant.

Le champ RPL joue un rôle important lorsque le système d'exploitation traite les sélecteurs d'une application. Nous avons vu précédemment qu'il contient toujours le niveau de privilège de la tâche qui le possède. Mais ce n'est là qu'une demi-vérité. Ce champ peut en fait recevoir n'importe quelle valeur mais le système d'exploitation, lorsqu'il attribue des sélecteurs, va y reporter le privilège de la tâche concernée. La raison de cette précaution est simple. Si plus tard un tel sélecteur est communiqué à une fonction système pour qu'elle accède par exemple à un buffer d'application, le niveau de privilège élevé de la fonction système ne doit pas se transmettre. Il est préférable que l'accès se fasse sur la base du niveau de privilège de l'appelant. Ainsi le détour par la fonction système n'ouvrira pas la voie à toutes les zones de données. Lorsque le champ RPL du sélecteur sera testé, il renverra au privilège de l'appelant et non à celui du système.

Registres fantômes

Si les différents segments mémoire ne sont plus référencés par leurs adresses physiques mais par des sélecteurs qui pointent sur des descripteurs de segment, il est clair que les registres de segment eux-mêmes ne doivent plus contenir d'adresses physiques. Grâce au sélecteur chargé dans un registre de segment le processeur doit être à même de trouver le descripteur concerné dans la table des descripteurs globale ou locale, et de déterminer la situation et la longueur du segment. Si ce mécanisme devait se répéter à chaque instruction qui implique un registre de segment, - ce qui est presque toujours le cas-, il en résulterait une perte de temps inacceptable.

C'est pourquoi au moment du chargement d'un registre de segment, l'adresse physique et la longueur du segment sont chargées dans un registre qualifié de fantôme. Tout registre de segment possède une partie fantôme. Le sélecteur de 16 bits est prolongé par une extension invisible de 48 bits. le même principe s'applique d'ailleurs aux registres GDTR, LDTR et IDTR, simplement la partie fantôme possède alors une autre signification.

C'est le chargement des registres fantômes et les contrôles de validité et de privilège qui l'accompagnent qui sont responsables du ralentissement du processeur en mode protégé. Ainsi l'instruction

```
MOV DS,AX
```

qui ne prend que 2 cycles en mode réel en nécessite 18 en mode protégé. Les programmes ne seraient pas ralentis si ce genre d'instruction restait rare, mais c'est exactement l'inverse qui est vrai. Sont notamment concernés les langages de haut niveau compilés dans les modèles mémoire qui génèrent des pointeurs FAR, par exemple le langage C à l'exception des modèles SMALL et TINY. La restriction à 64 Ko de la taille des segments empêche la plupart du temps une autre approche en mode réel.

Création des alias

Il n'est pas permis d'écrire des données dans un segment de code et le contenu d'un segment de données ne peut être lu par deux tâches dont les segments de code possèdent des privilèges différents. Ce sont là deux règles parmi de nombreuses chargées d'assurer le fonctionnement fiable d'un système multitâche. Mais il existe aussi des situations dans lesquelles ces protections doivent être levées.

Comment le système d'exploitation peut-il charger le code d'une application dans un segment de code puisque par définition il est interdit d'y écrire quoi que ce soit ? Comment deux tâches de privilège différent peuvent-elles accéder à un segment de données commun ?

Pour faire face à ce type de situation, il existe un mécanimse de création d'alias destiné à contourner en toute connaissance de cause les règles de protection du processeur. S'il est vrai que le processeur contrôle les accès à un segment mémoire par l'intermédiaire de son descripteur, il ne peut pas empêcher que l'on définisse plusieurs descripteurs différents pour un même segment et qu'on les mémorise dans la même table ou dans des tables différentes. Une zone de mémoire déjà identifiée comme segment de code par un certain descripteur peut également être décrite, en partie ou en totalité, comme segment de données. Le deuxième descripteur est qualifié d'alias car il rend possible l'adressage de la zone mémoire sous un autre "nom".

On parvient ainsi à résoudre non seulement le problème du chargement de code dans un segment de données mais aussi de la mémoire partagée. Il suffit de créer dans la table LDT de la tâche plusieurs descripteurs de segments de données différents qui reflètent à chaque fois le niveau de privilège de la tâche qui doit accéder au segment.

La création des alias est très utile pour assouplir les mécanismes de protection du processeur. Mais mal utilisée, elle ouvre évidemment la porte à tous les abus. C'est pourquoi le système d'exploitation n'acoorde des alias que dans des situations bien déterminées.

Accès au matériel

De même que l'accès des applications à la mémoire vive est contrôlée par la distribution de sélecteurs, de même il est possible d'accéder sous surveillance au matériel par les ports d'entrée-sortie. Les instructions associées IN et OUT, qui ont été complétées par INS et OUTS à partir du 80286, peuvent en effet recevoir des privilèges.

Ce sont deux bits à l'intérieur du registre des indicateurs qui gèrent ces privilèges. Ils forment l'indicateur appelé IOPL (I/O Privilege Level) qui indique le niveau de privilège que doit posséder une tâche pour exécuter les instructions d'entrée-sortie mentionnées. Si les deux bits sont par exemple à 1, seules les tâches de privilège 0 et 1 ont le droit d'exécuter ces instructions. Si jamais une tâche de niveau 2 ou 3 essaye de faire fonctionner une des instructions, le processeur génère une exception et active une routine d'erreur du système.

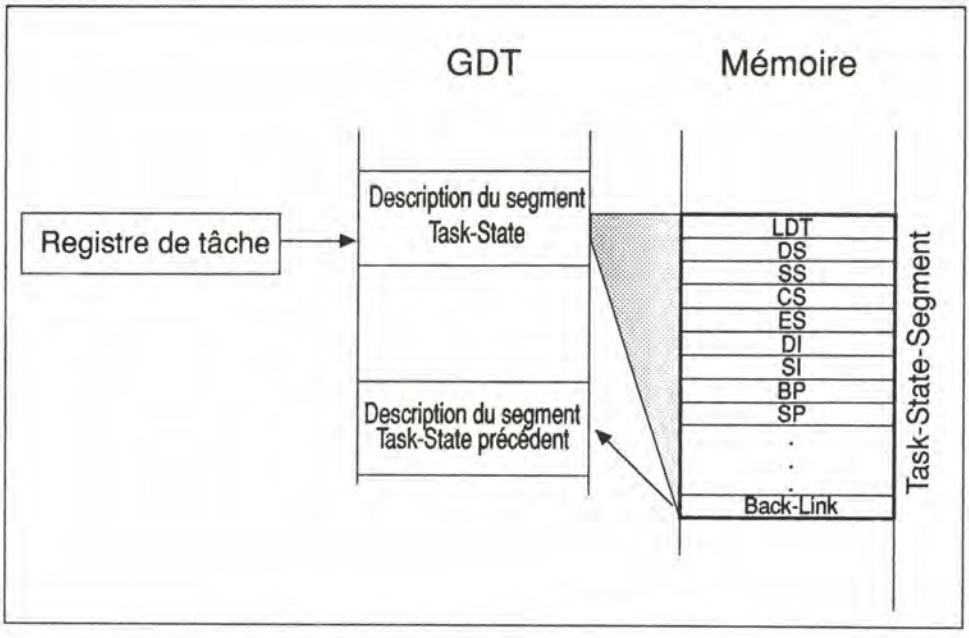
Ce mécanisme n'a de sens que si les applications ont un droit limité sur l'IOPL, sinon il leur suffirait de le modifier à sa guise pour s'accorder tous les privilèges. Il ne faut pas oublier qu'il est très facile de modifier un bit du registre des indicateurs: il suffit d'empiler le contenu souhaité puis d'exécuter une instruction POPF.

Mais les concepteurs du 80286 ont prévu le coup en modifiant quelque peu le fonctionnement de l'instruction POPF. Cette dernière peut toujours servir à fixer librement le contenu du registre des indicateurs quelque soit le privilège de l'appelant. Mais les bits de l'IOPL ne peuvent être modifiés que par une tâche de niveau 0, c'est-à-dire par le système d'exploitation. Ainsi les applications ne peuvent pas y toucher.

Commutation des tâches

Pour donner une impression d'exécution en parallèle, la commutation des tâches doit être extrêmement rapide. En mode protégé, le processeur se sert à cet effet du registre de tâche (TR, Task Register) et des segments d'état des tâches (TSS, task State Segments). Ce sont des zones mémoire de 44 octets qui au moment de l'interruption d'une tâche accueillent le contenu de tous les registres du processeur. Par ailleurs un pointeur référençant le TSS de la tâche interrompue est stocké pour que le processeur puisse la reprendre ultérieurement.

Comme tous les segments de mémoire, chaque TSS possède un descripteur qui en principe doit être mentionné dans la table des descripteurs globale. Même s'il s'agit en fait d'un descipteur système particulier, il indique la situation et la taille de son segment comme un descripteur commun. Ce descripteur est référencé par le registre TR qui doit toujours contenir le sélecteur du descripteur de la tâche courante. Au moment de la commutation, le contenu des registres du processeur peut ainsi être sauvegardé sans délai dans le segment d'état de la tâche.



Lien entre le registre de tâche TR et le segment d'état TSS

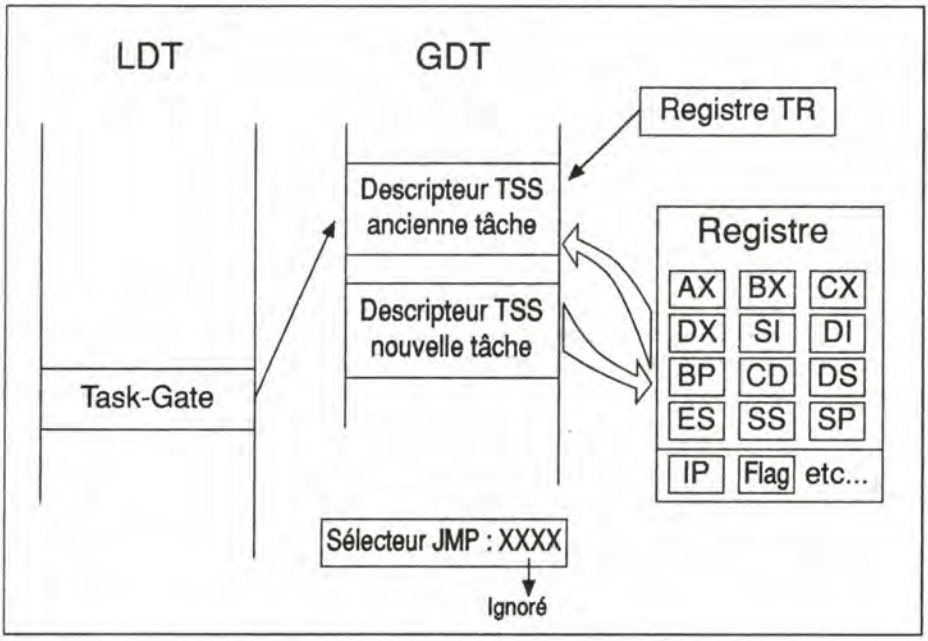
Une commutation de tâche peut s'opérer de plusieurs manières. En règle générale, elle est impliquée par un branchement JMP FAR ou un appel CALL FAR. Bien que ce type d'instruction nécessite à la fois la donnée d'un sélecteur et d'un offset, seul le sélecteur est pris en compte par la commutation, ce qui nous rappelle le comportement des portes d'appel. L'adresse d'offset est complètement ignorée.

Le sélecteur peut être issu d'un descripteur de TSS de la table GDT ou il peut s'agir du sélecteur d'une porte de tâche (Task gate). Tout comme une porte d'appel, une porte de tâche joue un rôle d'entremetteur et occupe une place de descripteur système dans une table locale ou la table globale. La seule information importante qui s'y trouve est le sélecteur du descripteur du TSS de la table globale qui identifie la nouvelle tâche. Par le détour de la porte de tâche, le processeur parvient finalement au descripteur du segment d'état de la tâche situé dans la table globale.

La commutation peut alors commencer mais il faut encore traiter le contenu du registre TR qui référence le descripteur du segment d'état de la tâche actuelle. Ce segment va mémoriser le contenu des registres du processeur. Puis le registre TR va être chargé avec le sélecteur du TSS de la nouvelle tâche. Le contenu présent du registre TR est encore conservé brièvement pour être reporté dans le champ Back-Link du nouveau segment d'état de tâche.

Les registres du processeur vont recevoir les valeurs référencées dans le nouveau segment d'état de tâche. Parmi les valeurs chargées figurent celles de CS et IP, de sorte que l'exécution continue à l'endroit même où la nouvelle tâche a été précédemment interrompue.

Pour créer et lancer une nouvelle tâche, le système d'exploitation créera d'abord un descripteur de TSS dans la table GDT puis le segment correspondant associé à la tâche. Comme la tâche en question n'a pas encore été active, les zones mémoire prévues pour les registres du processeur dans le nouveau TSS devront être initialisées manuellement. Ce sont surtout les registres de segment, le pointeur de pile et le registre IP qui sont importants car le couple CS :IP détermine l'adresse de départ de la nouvelle tâche. Grâce à un JMP FAR associé au sélecteur du TSS, l'exécution est transmise à la nouvelle tâche.



Commutation de tâche par un JMP FAR associé à un descripteur de porte de tâche

Interruptions et exceptions

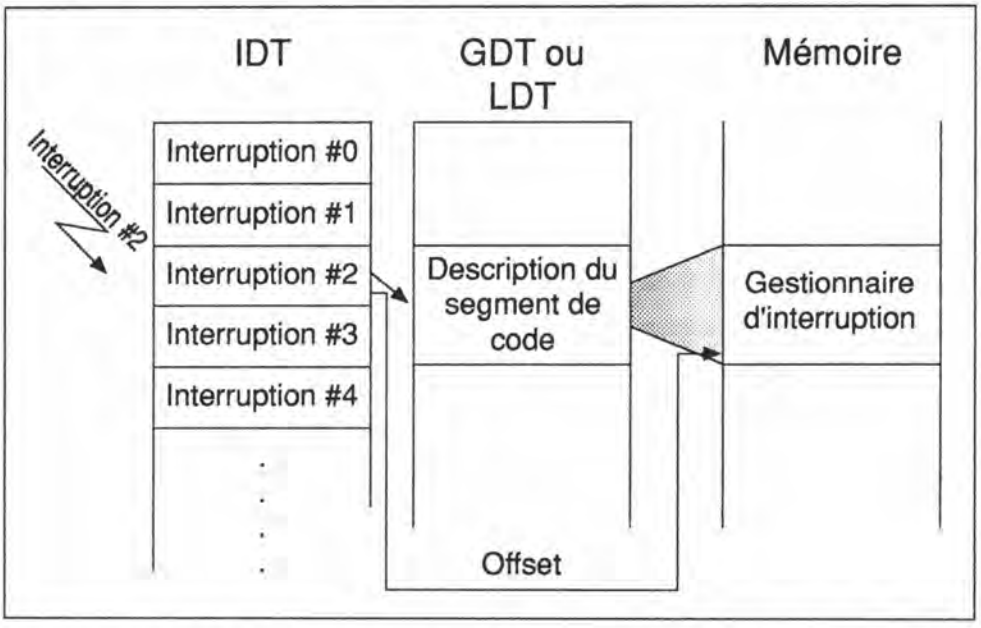
En mode protégé les interruptions sont traitées autrement qu'en mode réel. Tout comme avant, le processeur reconnaît 256 interruptions différentes et fait le lien entre une interruption et le gestionnaire associé mais ce lien ne s'effectue plus par l'intermédiaire d'une table de vecteurs d'interruption. C'est une table dite de descripteurs d'interruption (IDT, Interrupt Descriptor Table) qui reprend ce rôle. L'adresse où débute cette table dans la mémoire physique ainsi que sa longueur sont mémorisés dans le registre IDTR.

Seul le noyau du système qui tourne avec un niveau de privilège 0 a accès à ce registre en mode protégé. Contrairement à ce qui se passe en mode réel, une application n'est pas en mesure d'influencer l'exécution des interruptions. C'est là une condition d'importance pour assurer la stabilité de fonctionnement d'un système multitâche. Il faut se rappeler à ce sujet les nombreux problèmes suscités sous DOS par les programmes TSR qui gèrent librement la table des vecteurs d'interruption.

Si une interruption est déclenchée par un périphérique externe sur la ligne INTR ou par une instruction d'assembleur INT, le processeur prend le numéro de l'interruption comme indice d'accès à la table des descripteurs d'interruption et il y lit le descripteur associé. Il peut s'agir d'une porte d'interruption (interrupt gate) ou d'une porte de trappe (trap gate) qui ne se distinguent guère par leur structure. Dans l'une et l'autre on trouve essentiellement un sélecteur et un offset.

Le sélecteur sert de clé d'accès à un descripteur de segment de code situé dans la table globale ou une table locale. Le processeur prend ainsi connaissance du segment de code prévu par le gestionnaire d'interruption pour l'interruption déclenchée. L'offset sert à fixer le point d'entrée dans ce segment de code, c'est donc l'adresse de départ du gestionnaire d'interruption à l'intérieur du segment.

La différence entre porte d'interruption et porte de trappe tient simplement au traitement de l'indicateur d'interruption situé dans le registre des indicateurs. En mode réel, le principe général veut qu'une fois une interruption déclenchée, aucune autre ne peut plus être prise en compte. En effet le processeur met automatiquement l'indicateur d'interruption à 0 jusqu'à ce que l'interruption soit traitée. Le comportement du processeur est exactement le même en mode protégé lorsqu'un appel d'interruption passe par une porte d'interruption. Mais si l'appel passe par une porte de trappe, l'indicateur d'interruption du registre des indicateurs n'est pas mis à 0. En conséquence, de nouvelles interruptions peuvent survenir alors même que le gestionnaire n'a pas fini de traiter celle qui est en cours.



Déclenchement d'une interruption en mode protégé par l'intermédiaire de la table des descripteurs d'interruption

Les exceptions ne sont au fond que des interruptions directement provoquées par le processeur lorsqu'il détecte une anomalie d'exécution. Il peut s'agir par exemple d'une infraction à la réglementation des privilèges ou d'un accès au-delà des limites d'un segment.

Alors que le processeur ne prend en compte les interruptions envoyées par le contrôleur uniquement entre deux instructions, les exceptions sont déclenchées en plein milieu

d'une instruction. Une fois l'erreur traitée et si possible corrigée par le gestionnaire d'exception du système d'exploitation, l'instruction interrompue doit donc être réexécutée.

Les exceptions du 80286 occupent les numéros d'interruption de 0 à 16. Ils sont récapitulés par le tableau ci-dessous. Selon le type d'exception déclenché, le processeur dépose des informations sur la pile avant d'appeler le gestionnaire d'exception. Ces informations concernent l'origine de l'exception.

Numéro	Signification
0	Erreur de division
1	Pas à pas
2	NMI (erreur mémoire)
3	Point d'arrêt (pour DEBUG)
4	Déclenchée par la commande INTO
5	Déclenchée par la commande BOUND
6	Code opératoire inconnu
7	Coprocesseur non disponible
8	Double erreur
9	Dépassement du segment du coprocesseur
10	Segment d'état de tâche (TSS) non-valide
11	Segment indisponible (transféré)
12	Erreur de pile
13	Accès segment non autorisé
16	Erreur du coprocesseur

Exploitation des nouvelles instructions

Lorsqu'on développe des programmes pour le mode protégé du 80286 ou de l'un ses successeurs, on doit évidemment renoncer à la compatibilité avec le 8086. On peut donc sans regret se servir des instructions nouvellement introduites par le 80286. En voici la liste. Elles sont toutes exploitables en mode réel :

BOUND Teste si le contenu du registre se trouve dans un certain intervalle

ENTER Empile des paramètres de fonction et réserve de la place sur la pile pour les variables locales. Surtout utile pour interfacer l'assembleur avec des langages de haut niveau

INS Répétition de l'instruction IN

LEAVE Instruction symétrique de ENTER. Retire de la pile les paramètres transmis à une fonction et les variables locales

OUTS Répétition de l'instruction OUT

PUSHA Sauvegarde tous les registres sur la pile

POPA Reprend tous les registres sur la pile

Les instructions qui vont suivre sont indispensables lorsqu'on programme en mode protégé et qu'on est obligé de faire un peu de système, ce qui est pratiquement inévitable compte tenu du manque de support de DOS. Toutes ces instructions sont privilégiées et ne peuvent être exécutées qu'au niveau de privilège 0. Mais elles sont également accessibles en mode réel, ce dernier étant traité par le processeur par une tâche de niveau 0.

ARPL Contrôle et corrige le niveau de privilège de segment de code

LAR Charge le registre des indicateurs d'un descripteur

LGDT Charge dans le registre GDT l'adresse et la longueur de la table des descripteurs globale

LIDT Charge dans le registre IDT l'adresse et la longueur de la table des descripteurs d'interruption

LLDT Charge dans le registre LDT un sélecteur sur une table de description locale

LMSW Lit le mot d'état machine

LSL Charge dans un registre la longueur d'un segment

LTR Charge le registre de tâche TR

SGDT Mémorise le contenu d'un registre GDT

SIDT Mémorise le contenu d'un registre IDT

SLDT Mémorise le contenu d'un registre LDT

SMSW Mémorise le contenu du mot d'état machine

STR Mémorise le contenu du registre de tâche TR

VERR Vérifie si les accès en lecture sont autorisés pour un segment donné

VERW Vérifie si les accès en écriture sont autorisés pour un segment donné

33.1.3. Le mode protégé du 80386 et de l'i486

Le fonctionnement du 80386 et de l'i486 n'est pas fondamentalement différent de celui du 80286 tel qu'il a été décrit dans la section précédente. De nouvelles notions ont certes été introduites mais elles n'ont pas à être exploitées obligatoirement par un système multitâche. C'est ainsi que la gestion des pages de mémoire par blocs de 4 Ko est surtout utile lorsqu'on installe une gestion de mémoire virtuelle. De même il est possible maintenant d'inhiber séparément certains ports d'entrée-sortie. Ces extensions vont être décrites dans les pages qui suivent.

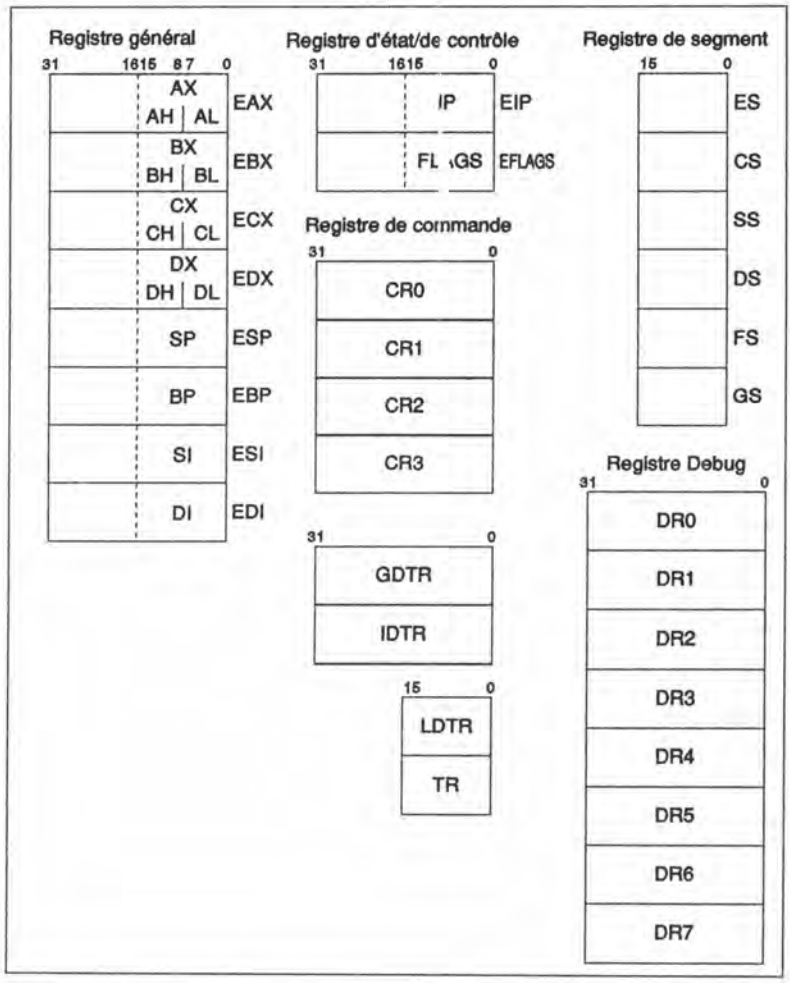
Les modifications les plus profondes affectent également le fonctionnement en mode réel du 80386 et de l'i486 : il s'agit avant tout de la présence de registres 32 bits et de l'adaptation du jeu des instructions à la nouvelle dimension de ces registres. La première section de ce chapitre sera donc consacrée aux registres du 80836 et de l'i486 et à leurs implications sur la programmation en mode protégé.

Les registres du 80386 et de l'i486

L'introduction du 80386 marque un changement de génération dans le monde de la famille 80xxx d'Intel: elle inaugure l'ère des processeurs de 32 bits. Alors que les troispremières générations de cette famille, représentées par le 80886, le 80186 et le 80286 se limitaient à traiter des nombres de 16 bits, le 80386 et ses successeurs ouvrent la porte de l'univers à 32 bits.

Le jeu des registres du processeur en tire les conséquences car la plupart de ces registres ont maintenant 32 bits. Les registres généraux AX, BX, CX et DX ont certes encore 16 bits, tout comme DI, SI et BP mais ils ne constituent plus que la partie inférieure de registres étendus de 32 bits. Ces registres étendus s'appellent EAX, EBX, ECX etc... la lettre "E" signifiant évidemment "extended". Les registres de 8 bits AH, AL, BH, BL etc... sont encore disponibles mais les 16 bits supérieurs des registres E ne sont plus divisibles en deux registres de 8 bits. Le nombre des registres de 8 bits présents ne s'est donc pas accru par rapport à la situation qui existait avec le 80286 et ses prédecesseurs.

Toutes les instructions telles que MOV, SHL, ADD etc... peuvent être appliquées non seulement comme avant aux registres de 8 et 16 bits mais aussi aux nouveaux registres de 32 bits. Le 80386 et ses successeurs accélèrent donc la vitesse de traitement des mots doubles DWORDS (type long en C et longint en Pascal).



Les registres du 80386 et de l'i486 (sans le registre de virgule flottante)

La figure précédente montre que par rapport au 80286 les différents registres n'ont pas seulement augmenté en taille mais aussi en nombre. Sont venus s'ajouter les deux registres FS et GS ainsi que deux autres registres de segment qui comme ES peuvent faciliter certains accès mémoire. DS reste le registre des données par défaut tandis que CS reste le registre par défaut où s'exécute le programme en cours.

Comme avec le 80286, les différents registres de segment reflètent en mode réel les adresses des segments référencés. En mode protégé, ils contiennent des sélecteurs pour l'accès aux descripteurs de segment issus de la tables des descripteurs globale ou locale. Rien de neuf par rapport au 80286, les registres en question conservant leur taille de 16 bits.

De même rien n'a changé pour les registres GDTR, LDTR, IDTR et TR qui ont une importance déterminante dans la gestion de la mémoire et des tâches. Ces registres ne sont toujours accessibles qu'au noyau du système d'exploitation au niveau de privilège 0. Mais ils reçoivent le renfort de quatre registres de commande CR0, CR1, CR2 et CR3 qui possèdent 32 bits et dont l'usage est également réservé au système d'exploitation. Nous reparlerons d'eux un peu plus tard lors de l'étude du mécanisme de pagination.

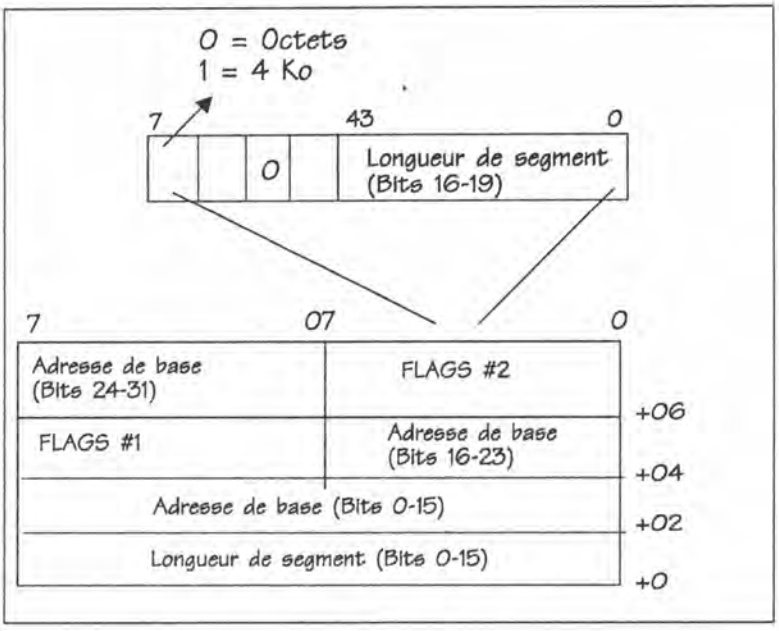
Comme dernière nouveauté le 80386 présente huit registres de débogage appéle DR0 à DR7. Grâce à eux il est possible de poser quatre points d'arrêt qui ne se déclencheront pas seulement sur une instruction donnée mais également en cas d'accès à une mémoire fixée à l'avance, à n'importe quel endroit d'un programme. Des horizons nouveaux s'ouvrent ainsi aux débogueurs logiciels avec des perfectionnements jusqu'ici réservés à de coûteuses extensions matérielles.

Des segments agrandis

Les registres généraux AX, BX, CX et DX ne sont pas les seuls à acquérir un format 32 bits, les registres d'index DI, SI, BP et même SP suivent le même chemin. Mais à quoi peuvent bien servir des registres d'index 32 bits si un segment ne peut dépasser 64 Ko et si une indication d'offset ne prend que 16 bits ?

A rien serait-on tenté de répondre mais les développeurs d'Intel ont certainement poursuivi un but. Il s'avère en effet que les adresses de segments ont été étendues à 32 bits, ce qui permet à un segment d'atteindre une taille de 4 Go (2^{32}). Il n'a pas été nécessaire de modifier la longueur du descripteur de segment par rapport au 80286 car les ingénieurs d'Intel y avaient déjà réservé un mot terminal à l'usage du 80386 et ses successeurs. Compte tenu de ce mot, la structure d'un descripteur est donc identique que l'on soit en présence d'un 80286 ou de l'un des processeurs ultérieurs.

Dans le cas du 80386 et de l'i486, le septième octet du descripteur prend en compte une extension du registre des indicateurs, les quatre bits inférieurs recevant les bits 16 à 19 de la longueur du segment. La longueur du segment est ainsi portée à 20 bits, ce qui ouvre un espace accessible de 1 Mo, et non de 4 Go comme mentionné précédemment.



Structure des descripteurs de segments avec les 80386 et i486

Dans les nouveaux descripteurs de segments, l'adresse de base du segment occupe désormais 32 bits. Alors que les 24 bits inférieurs sont à leur place habituelle, les bits 24 à 31 sont hébergés dans le dernier octet du descripteur.

L'adresse de base et la longueur d'un segment s'exprimant sur 32 bits, l'espace d'adressage linéaire atteint 4 Go lorsqu'on exploite des processeurs 80386 et i486, la mémoire virtuelle va même jusqu'à 64 To (téraoctets, soit 2^{46}). Ce sont là des capacités astronomiques qui nous paraissent aujourd'hui inépuisables. Mais les dix premières années de l'histoire de la micro-informatique nous ont montré que les besoins croissaient très rapidement, mais la réalisation d'une capacité de 4 Go est encore une gageure du point de vue de l'électronique.

Pour l'instant on trouve surtout sur le marché des mémoires de 512 Ko (4 Mbits). Huit de ces circuits sont rassemblés en un module SIMM (Single Inline Memory Module) qui possède une capacité de 4 Mo. Pour atteindre 4 Go, il nous faut un facteur multiplicatif de 2^{10} (1024), ce qui représente dix générations de composants si on considère que les capacités doublent à chaque génération. Jusqu'ici les générations se succédaient à environ deux ans d'intervalle, il nous faudra donc attendre encore 20 ans sans compter les difficultés croissantes de finition. Il n'y a donc pas de raison de prévoir des registres d'adresses de 64 bits pour le nouvel i586: 20 ans en informatique, c'est une petite éternité.

Les offsets prenant 24 bits, les pointeurs FAR occuperont non plus 4 mais 6 octets dans le mode protégé du 80386 et de l'i486: deux octets pour le sélecteur de segment et quatre pour l'offset.

Pagination et gestion de mémoire virtuelle

Le 80286 permet déjà de réaliser une gestion de mémoire virtuelle car chaque descripteur de segment contient un indicateur qui signale la présence du segment en mémoire. Si pendant l'exécution le processeur tombe sur un segment absent, il déclenche automatiquement une exception qui est interceptée par le système d'exploitation et est interprétée comme une demande de chargement du segment.

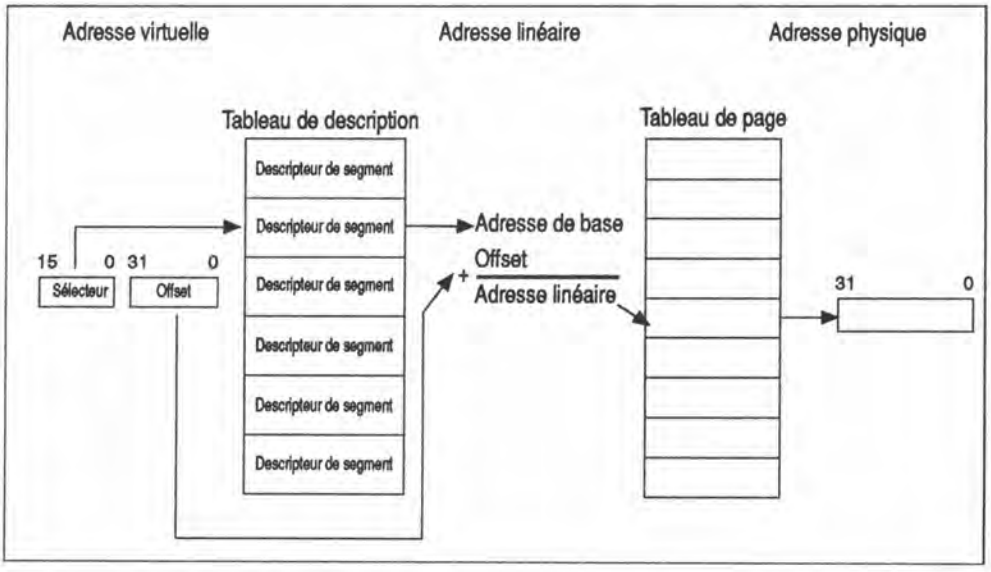
Cette méthode n'est cependant pas la meilleure pour mettre en place une gestion de mémoire virtuelle, et cela pour plusieurs raisons. D'abord les blocs de mémoire à importer et à exporter ont des tailles différentes, car il s'agit de segments complets qui chez un 80286 peuvent varier de 1 octet à 64 Ko. Il est possible que seule une partie du segment soit utile, et que dans l'instant qui suit elle doive être retirée de la mémoire en raison d'une commutation de tâche. C'est pourtant l'intégralité du segment qui est expatriée avant d'être rechargée. Par ailleurs la taille du fichier d'échange (swap file) croît sans cesse. Il arrive souvent qu'un segment B doive être réenregistré dans le fichier d'échange mais qu'il ne puisse pas y prendre la place du segment A qui vient d'être rappelé en mémoire.

Un problème de fragmentation similaire se pose en mémoire centrale car les différents segments rappelés et restockés n'ont pas la même taille. Le gestionnaire de mémoire virtuelle est sans cesse contraint de rapprocher les segments. Même si ce mécanisme est transparent et s'effectue sans que les différents programmes s'en rendent compte (le sélecteur de segment ne change pas, seuls les descripteurs de segments sont modifiés), il n'en reste pas moins vrai qu'il gaspille du temps d'exécution.

Tous ces problèmes trouvent une solution si on maintient constante la taille des blocs en circulation. Ce processus doit se trouver en retrait par rapport à la segmentation, c'est-à-dire à un niveau plus bas qui le rend transparent. C'est ce que font le 80386 et ses successeurs qui implémentent un mécanisme de pagination qui repose sur des blocs de 4 Ko. Tant que ce mécanisme est désactivé, les adresses linéaires qui résultent de l'exploitation des sélecteurs de segment et des offsets correspondent à des adresses physiques auxquelles accèdent effectivement le processeur, comme c'est le cas pour le 80286.

Lorsque la pagination est activée, les adresses linéaires ne sont plus identiques aux adresses physiques mais elles y sont associées par le moyen d'une table des pages. Si examine isolément les instructions machines qui accèdent à la mémoire, on s'aperçoit que ce processus ralentit l'exécution car la formation des adresses requiert un décodage supplémentaire. Mais si à un niveau plus élevé on prend en considération la performance

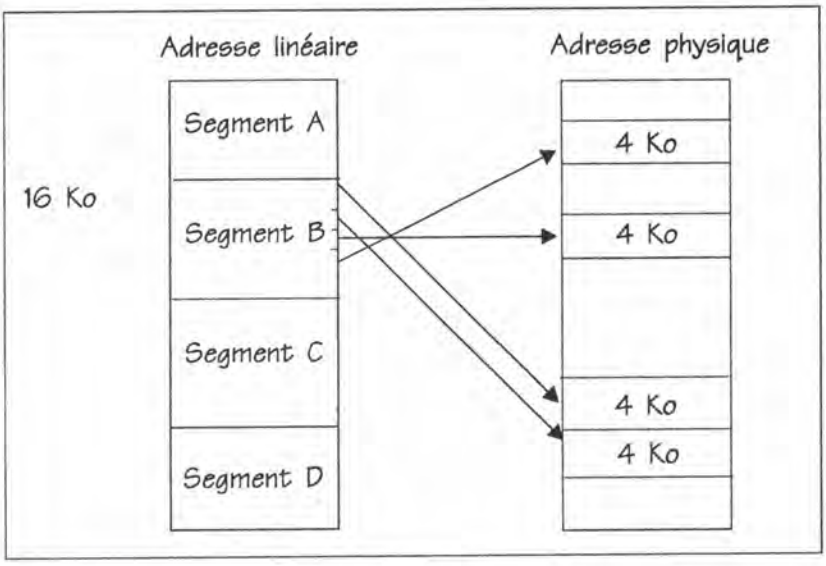
d'un système d'exploitation multitâche, ce mécanisme se révèle très efficace car il permet la réalisation d'une gestion de mémoire virtuelle nettement meilleure.



Formation d'une adresse virtuelle avec pagination active

La mise en service de blocs de mémoire de taille constante ne simplifie pas seulement la gestion du fichier d'échange. L'accroissement de ce fichier n'est plus incessant: lorsqu'une page de 4 Ko se libère, une autre peut prendre sa place puisqu'elle a la même taille. Par ailleurs seules des pages limitées à 4 Ko vont circuler, et non plus des segments entiers pouvant théoriquement aller jusqu'à 1 Go.

Autre avantage: le rapprochement des blocs en mémoire n'est plus nécessaire. En effet les blocs en question peuvent être placés à n'importe quelle adresse linéaire par l'intermédiaire de la table des pages. Les quatre blocs qui composent un segment de 16 Ko peuvent ainsi s'installer à des adresses physiques complètement différentes tout en formant un espace linéaire continu pouvant être exploité de la façon habituelle par un programme. Mais quels sont les aspects concrets de ce mécanisme ?



Un espace linéaire peut apparaître continu alors que les adresses physiques correspondantes sont disséminées en mémoire

La table des pages

C'est le bit PG du registre de contrôle 0 (CR0) qui déclenche la mise en service de la pagination. Normalement ce bit est à 0: les adresses linéaires sont alors reproduites par des adresses physiques et il n'y a pas de pagination. Si dans le cadre du mode protégé ce bit est mis à 1, les adresses linéaires sont converties par l'intermédiaire d'une table des pages.

La taille de chaque page est de 4 Ko, comme nous l'avons déjà vu, et chaque page commence à une adresse physique multiple de 4 Ko. L'espace linéaire de 4 Go (232) exploitable par les 80386 et i486 est réparti sur 220 pages de 212 octets (4 Ko) chacune. Le compte est bon car 220 * 212 = 232.

La division en pages de 4 Ko permet de reprendre directement dans l'adresse physique les 12 bits inférieurs d'une adresse linéaire. Ils représentent donc une sorte d'offset à l'intérieur de la page. Les 20 bits supérieurs de l'adresse linéaire indiquent le numéro de la page où se trouve la mémoire adressée. Ils sont considérés comme un indice dans la table des pages, d'où il est possible de tirer l'adresse de base physique de chaque page.

Chaque élément ou entrée de la table des pages possède 32 bits. Pour l'adresse de base, 20 bits suffisent puisqu'elle doit représenter un emplacement physique multiple de 4 Ko. Les 12 bits inférieurs sont donc nécessairement nuls. On les utilise en fait pour

héberger différents indicateurs qui servent à signaler au gestionnaire de mémoire virtuelle si une page donnée se trouve en mémoire ou doit être lue dans le fichier d'échange.

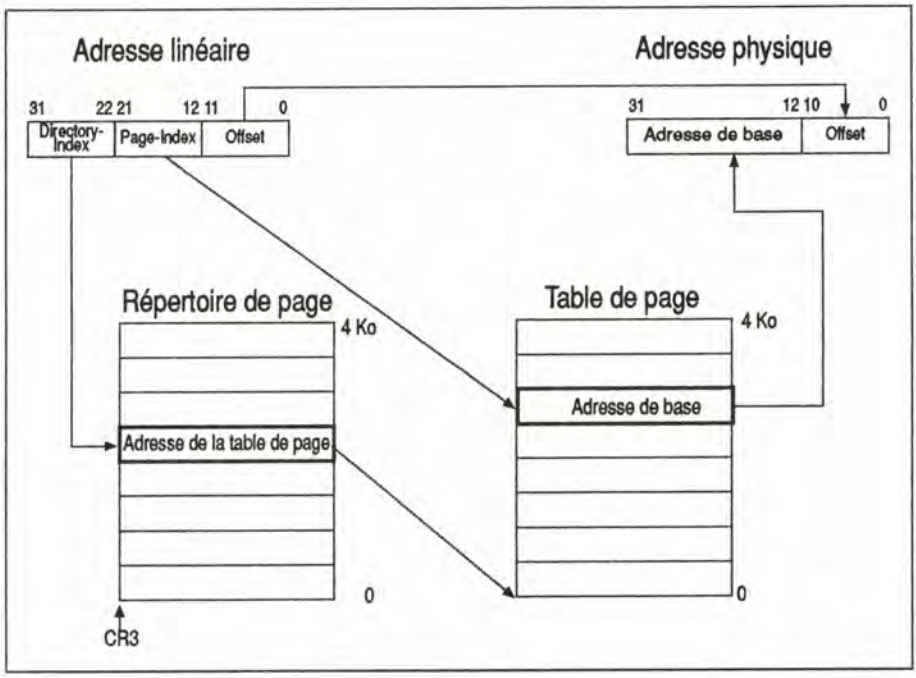
La formation de l'adresse physique ne requiert donc que les 20 bits supérieurs de l'élément de la table des pages. Ces 20 bits sont complétés par les 12 bits inférieurs de l'adresse linéaire qui peuvent être repris tels quels.

A vrai dire, l'opération est un peu plus compliquée car sans précaution la table des pages avec ses 220 éléments de 4 octets prendrait 4 Mo, ce qui représenterait un gaspillage inacceptable. Si l'ordinateur dispose de 4 Mo de mémoire vive, la table ne comporte que 1024 éléments et n'occupe donc que 4 Ko. Malgré tout, à chaque page de l'espace linéaire doit être associée une entrée dans la table des pages. Il se pourrait en effet qu'une mauvaise adresse se glisse dans un descripteur de segment et conduise à une adresse linéaire qui n'est pas prévue dans la table des pages. Le processeur est dans l'impossibilité de détecter cet événement car contrairement aux tables de descripteur globale et locales la table des pages ne comporte pas d'indication de longueur. Notre brave processeur va donc lire un élément supposé de la table des pages alors qu'il s'agit en fait d'une instruction ou d'une donnée qui joue un tout autre rôle. L'adresse physique formée sera fantaisiste et les conséquences qui peuvent en résulter n'ont pas besoin d'être dépeintes.

Partition de la table des pages

Pour éviter ce type de problème sans gaspiller la mémoire vive, on met en place une organisation des tables à deux étages. Le point de départ est le répertoire des pages qui permet de gérer plusieurs petites tables de pages. Comme le montre l'illustration ci-après, les 10 bits supérieurs de l'adresse linéaire jouent le rôle d'un indice d'accès au répertoire des pages. Ce répertoire comportera donc 1024 éléments. Chacun de ces éléments occupant 32 bits, la table tout entière prend 4 Ko.

Les différents éléments constituent des pointeurs sur les adresses des tables de pages dans l'espace mémoire physique. Ce n'est qu'en les mettant à contribution que le processeur apprend l'adresse physique d'une page. Le numéro de l'élément qui est pris dans la table des pages pour le calcul de l'adresse physique est obtenu par les bits 12 à 21 de l'adresse linéaire. Celle-ci est donc finalement divisée en deux parties: un indice de 10 bits se rapportant au répertoire des pages et un deuxième indice de 10 bits se rapportant à la table des pages correspondante. Chaque table de pages, tout comme les tables de répertoire, contient les adresses de 1024 pages et occupe donc 4 Ko.



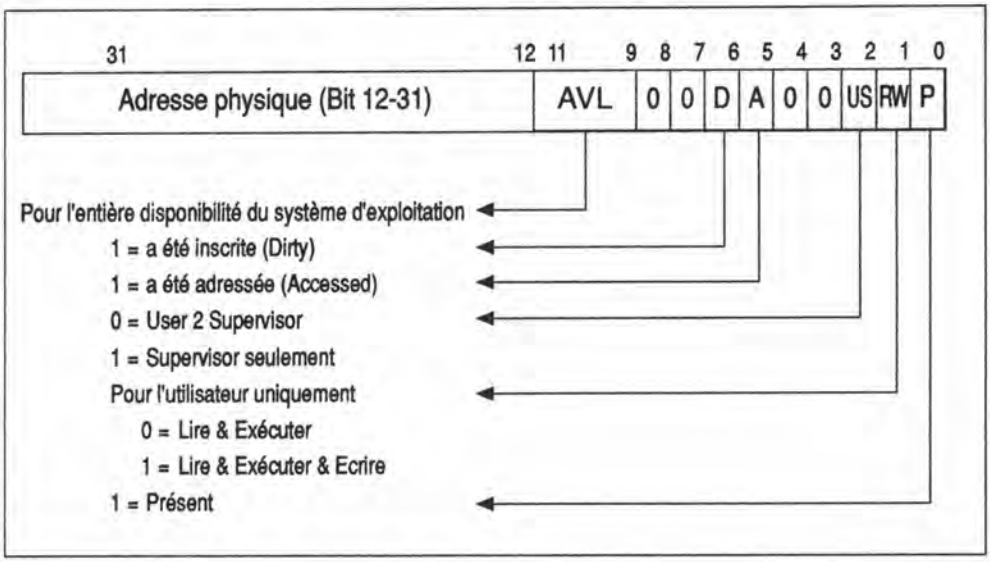
Conversion d'une adresse linéaire en adresse physique par la table des pages

Grâce à cette structure chaque élément du répertoire des pages couvre un domaine de 4 Ko à l'intérieur de l'espace mémoire linéaire. L'avantage de ce procédé est évident: pendant l'amorçage du système d'exploitation il suffit de constituer un seul répertoire de page et un seule table des pages pour gérer les quatre premiers Mo de la mémoire. Dans un premier temps, il n'est sûrement pas nécessaire de disposer de plus de 4 Mo de mémoire. Lorsque les différents composants réclameront de la mémoire, le système d'exploitation va en faire une distribution contigüe et non pas disséminée. C'est pourquoi la totalité de la mémoire allouée va se trouver dans les premiers 4 Mo et sera donc gérée par la table des pages référencée par le premier élément du répertoire des pages.

Tous les autres éléments du répertoire des pages pourront faire l'objet d'un marquage spécial destiné à déclencher une exception en cas de traitement par le processeur. Ce n'est qu'à partir du moment où la demande de mémoire dépasse 4 Mo qu'une nouvelle table des pages devra être créée, son adresse étant mémorisée dans le deuxième élément du répertoire des pages. Finalement, chaque tranche de 4 Mo de mémoire nécessitera une table de 4 Ko. Voilà qui est très inférieur aux 4 Mo requis par une table de page unique qui gérerait à elle seule tout l'espace linéaire.

Bien entendu, en plus des adresses des différentes tables de pages il faut encore stocker l'adresse du répertoire des pages pour qu'elle soit portée à la connaissance du processeur. C'est là la destination du registre CR3 qui est prévu pour mémoriser l'adresse physique dudit répertoire. Le registre CR2 joue aussi un rôle dans la pagination mais

ce rôle est plutôt mineur. C'est en CR2 que le processeur charge l'adresse linéaire en cours en cas d'exception due à une erreur de pagination.



Structure d'un élément de la table des pages

Stratégies de stockage des pages

Le gestionnaire de mémoire virtuelle va se servir des différents indicateurs qui constituent les 20 bits inférieurs des éléments de la table des pages. Le plus important de ces indicateurs est certainement l'indicateur de présence. Il doit être mis à 0 par le système d'exploitation lorsqu'une page a été remisée dans le fichier d'échange. Lorsque le processeur fera appel à cette page, une exception sera générée et le système d'exploitation pourra reprendre le contrôle des opérations pour recharger la page. En même temps il devra mettre à 1 le bit de présence sinon le processeur redéclenchera une exception au prochain accès.

Un gestionnaire de mémoire virtuelle exploitera aussi les bits Dirty et Accessed. Ce dernier est automatiquement mis à 1 par le processeur avant qu'il n'accède à la page référencée par un élément de la table des pages. Dans le répertoire des pages cet indicateur est mis à 1 même si une seule des 1024 pages correspondantes fait l'objet d'un adressage.

Comme le processeur met toujours à 1 cet indicateur sans jamais l'annuler, le gestionnaire de mémoire virtuelle s'en servira pour marquer les pages adressées. Quand une page a été chargée en mémoire ou swappée sur disque, il suffit de mettre l'indicateur Accessed à 0 pour tester plus tard si la page a déjà été adressée.

Cette information est toujours précieuse lorsque la mémoire physique se fait exiguë et que certaines pages doivent être enregistrées dans le fichier d'échange. Pour sélectionner les pages à swapper, on privilégiera celles qui n'ont pas été adressées ces derniers temps et qui de ce fait ont des chances de ne pas être réutilisées dans un avenir immédiat. Il existe cependant des raisons de penser que ce sont justement les pages qui viennent d'être adressées qui doivent être stockées sur fichier. Les tenants de cette théorie prétendent que ces pages viennent d'être exploitées et que ce sont les autres qui attendent leur tour: c'est là une querelle d'école dans laquelle nous n'entrerons pas.

Indépendamment de cette remarque, le bit Dirty fournit un autre élément de décision pour définir la stratégie qui nous préoccupe. Ce bit est mis à 1 en cas d'accès en écriture. Grâce à lui on peut distinguer les pages modifiées de leurs consoeurs qui n'ont pas été retouchées. Ces dernières sont à privilégier pour le swap car elles peuvent encore figurer sous leur forme d'origine dans le fichier d'échange: elles n'ont pas besoin d'être restockées, il suffira de les recharger telles quelles. Le bit Dirty pourra ainsi être intelligemment exploité par une stratégie appropriée.

Mécanismes de protection

Les autres indicateurs qui se trouvent dans les éléments de la table des pages réalisent un mécanisme de protection semblable à celui qui existe au niveau des adresses linéaires grâce aux privilèges inscrits dans les descripteurs de segments. A vrai dire, le système travaille avec moins de finesse : les quatre niveaux de privilèges sont remplacés par une simple distinction entre code de superviseur (supervisor code) ou code utilisateur (user code). Toutes les tâches ayant un niveau de privilège 0, 1 ou 2 sont considérées comme du code de superviseur et sont donc attribuées au système d'exploitation. Les tâches de niveau 3 doivent se contenter du simple statut de code utilisateur.

Lorsque la mise à 1 de l'indicateur superviseur donne à une page le rang de page de supervision, les tâches d'application ne peuvent pas y accéder, toute infraction déclenchant une exception. Si l'indicateur n'est pas à 1, l'accès est autorisé à la fois aux tâches superviseur et aux tâches utilisateur.

En cas d'accès à une tâche d'application, un autre indicateur appelé indicateur de lecture/écriture (Read Write Flag) vient compléter la protection. S'il contient la valeur 0, la page concernée peut être lue et exécutée mais non modifiée. Si sa valeur est 1, la page est complètement ouverte à la lecture comme à l'écriture. Les infractions sont signalées au système d'exploitation par des déclenchements d'exception.

La conversion d'une adresse linéaire en adresse physique coûte évidemment beaucoup de temps, c'est d'ailleurs surtout le chargement des tables de pages en mémoire qui est dispendieux. Le 80386 et ses successeurs disposent à cet effet d'une mémoire cache destinée à stocker les derniers éléments de la table des pages. Dans la littérature spécialisée, cette mémoire cache est parfois désignée sous le nom de TLB: Translation Lookaside Buffer. Elle mémorise systématiquement les 32 derniers éléments lus dans

la table des pages. Cet espace ne recouvre que 128 Ko de la mémoire vive, mais Intel prétend que la mémoire cache ainsi installée est efficace à 98 %.

Mais toute médaille a son revers et la mémoire cache apporte avec elle un lot de problèmes nouveaux. Ils surgissent lorsqu'un élément de la table des pages a été modifié en mémoire centrale alors que la mémoire cache qui contient le même élément n'a pas subi la même modification. Dans les systèmes à base 80386 et i486, c'est le logiciel qui est responsable de l'intégrité de la TLB car le processeur ne s'en préoccupe pas par lui-même. Il dispose cependant d'un petit mécanisme qui permet de déclarer périmé le contenu de la mémoire cache pour que ses éléments puissent être rechargés progressivement. Ce mécanisme consiste à charger dans le registre CR3 l'adresse où commence le répertoire des pages. Si cette adresse ne demande pas de modification et si la seule péremption de la mémoire cache est à signaler, il suffit de remettre dans le registre CR3 son propre contenu:

```
MOV AX,CR3
MOV CR3,AX
```

On maintient ainsi le buffer TLB en bonne santé.

La pagination ne s'impose pas nécessairement pour l'ensemble des tâches, elle peut très bien être activée individuellement pour l'une ou l'autre tâche. En effet à chaque commutation de tâche l'indicateur de pagination peut être mis à 0 ou à 1 dans le registre CR0 et dans le registre CR3 le système peut charger à sa guise une adresse de début d'un répertoire de pages.

Inhibition sélective des ports d'entrée-sortie

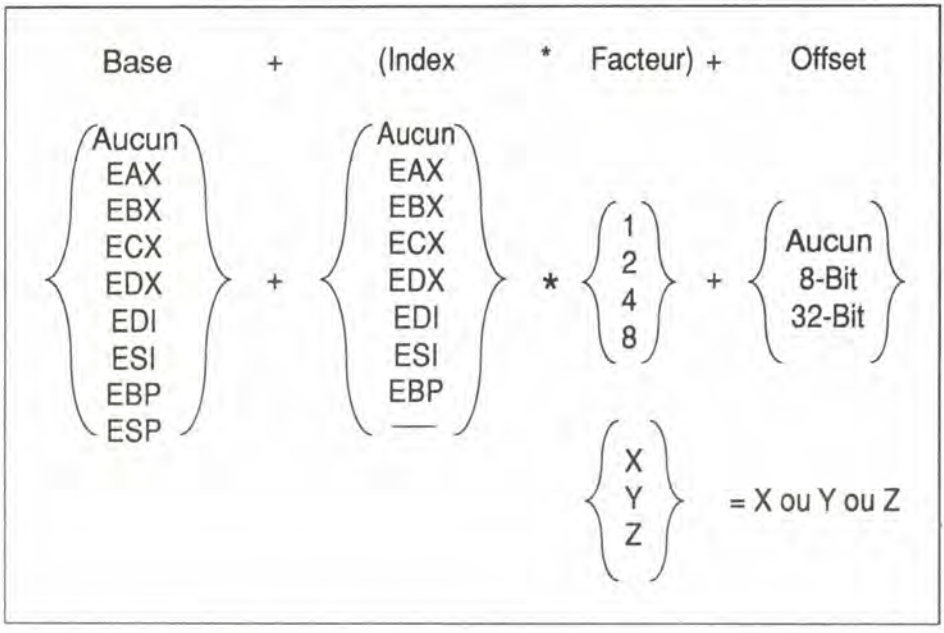
Pour déterminer si un port d'entrée-sortie peut être adressé par une tâche, le 80386 et ses successeurs sont un peu plus sélectifs que le 80286. Avec le 80286 seuls comptaient l'indicateur d'IOPL dans le registre des indicateurs et le niveau de privilège de la tâche en cours. Avec les nouveaux processeurs il est possible de définir une table d'autorisations d'entrée-sortie dans le segment d'état d'une tâche. Cette table est une table de bits pouvant comporter jusqu'à 216 éléments. Elle peut donc gérer 65536 ports indiquant si un accès est possible indépendamment du niveau de privilège ou s'il est nécessaire de tester l'indicateur d'IOPL. La valeur 0 ouvre le libre accès, tandis que la valeur 1 requiert le test d'IOPL.

L'offset de l'adresse de début de la table à l'intérieur du segment d'état de tâche n'est pas fixé a priori mais peut être librement choisi par le système d'exploitation. Mais il doit être indiqué à l'offset 66h de ce même segment d'état de tâche qui du fait de l'élargissement des registres du 80386 comporte au moins 102 octets. A partir de cette adresse de début et de la longueur du segment d'état de tâche qui est inscrite dans son descripteur, le processeur détermine la taille de la table binaire. Les ports qui ne sont pas mentionnés dans cette table sont soumis au test d'IOPL.

Si l'offset de la table binaire est supérieur ou égal à la longueur du segment, c'est qu'en fait la table n'existe pas et c'est le test d'IOPL qui intervient. Si entre l'offset indiqué et la longueur du segment il reste plus d'espace que les 8 Ko maximaux, c'est que tous les ports sont couverts par la table. Si aucune des deux conditions indiquées n'est réalisée, la différence entre l'extrémité terminale du segment d'état de tâche et le début de la table binaire est multipliée par 8 pour donner le nombre de ports représentés.

Des modes d'adressage plus souples

L'introduction du 80386 a permis une extension des modes d'adressage, le processeur se rapprochant de ses concurrents de Motorola. En mode réel comme en mode protégé tous les registres généraux peuvent désormais servir de registre de base ou de registre d'index, on n'est plus limité à certaines combinaisons de SI, DI et BP. L'un des registres d'index peut être multiplié automatiquement par le facteur 2, 4 ou 8 pour permettre un accès rapide aux tables de mots (WORDS), doubles mots (DWORDS) ou quadruples mots (QWORDS). Un offset de 8 ou de 32 bits peut encore être ajouté à l'ensemble comme le montre la figure suivante.



Mode d'adressage du 80386 et de ses successeurs

Plus encore qu'avec les processeurs précédents, il est recommandé avec le 80386 et l'i486 d'aligner les variables en mémoire, en fonction de leur taille, à des frontières de mots, doubles mots ou quadruples mots. L'accès est notablement ralenti si plusieurs lectures ou écritures sont nécessaires pour composer les opérandes.

Nouvelles instructions

Comme le 80286, le 80386 a été doté de quelques instructions ou modes opératoires supplémentaires. Ne sont concernées que les instructions non privilégiées car au niveau de privilège 0 les ordres système n'ont pas varié.

Le tableau suivant récapitule les nouvelles possibilités que recèle le jeu d'instructions du 80386 :

BT,BTC,BTR,BTS	Instructions pour tester et manipuler des bits dans les registres et les variables
BSF,BSR	Instructions pour rechercher des bits à 0 ou à 1 dans les registres et variables
LSS,LFS,LGS	Charge l'un des registres de segment SS,FS ou GS
Jxx 32-bits	Branchement conditionnel (JA, JC, JBE, etc...) avec déplacement de 32 bits
MOVSX, MOVZX	Instruction MOV avec extension automatique du signe ou extension de 0
MOV Drx,Reg	Chargement d'un registre de mise au point
MOV CRx,Reg	Chargement d'un registre de contrôle
MOV Reg,CRx	Chargement du contenu d'un registre de contrôle
SHRD, SHLD	Décalage de doubles mots vers la droite ou vers la gauche
SETxx	Initialisation d'un un registre de 8 bits si la condition indiquée est remplie. Combine les instructions CMP, Jxx et MOV

33.1.4. Le mode virtuel des processeurs 80386 et i486

Alors qu'ils s'affairaient à mettre au point le 80286, les développeurs d'Intel misaient sur le succès rapide du mode protégé et la morte lente du mode réel. Pendant la conception du 80386, les idées durent se plier à l'implacable entêtement des faits, autrement dit à la perennité de DOS. Le mode virtuel 86 (V86) devait constituer la symbiose entre le mode réel et le mode protégé. Il est aujourd'hui présent dans de nombreux types d'extensions système. Son domaine va des émulateurs d'EMS (par exemple EMM386.SYS de Microsoft) aux extensions multitâche (par ex DesqView/386) en passant par la boîte DOS de Windows.

Du point de vue d'un programme tournant en mode V86, le système apparaît comme un ordinateur fonctionnant en mode réel, alors qu'à l'arrière-plan la gestion de la mémoire, la commutation des tâches et les règles de privilège du mode protégé restent actifs. En mode V86, un programme exécuté comme une tâche V86 autonome ne voit rien de ces mécanismes. Il a les yeux fixés sur son espace mémoire de 1 Mo qui est adressé selon les règles habituelles du mode réel, avec la formule Segment+Offset*16. Pour la tâche V86, les contenus des registres de segment sont véritablement des adresses de segments et non des sélecteurs comme en mode protégé. Un programme peut donc charger n'importe quelle valeur dans un registre de segment sans craindre de déclencher une exception.

Pourtant le processeur punit la tâche V86 lorsqu'elle utilise l'un des modes d'adressage sur 32 bits du 80386 pour générer un offset qui se trouve au-delà de la limite des 64 Ko. Dans ce cas il déclenche bien une exception mais cette situation est rarissime car le mode V86 est généralement mis en service pour exécuter des programmes DOS ordinaires qui ne font pas appel aux possibilités étendues du 80386 et de ses successeurs.

Comme il n'existe en mode V86 ni sélecteurs ni tables de descripteurs, les seules adresses exploitées sont linéaires. Le mégaoctet de mémoire d'une telle tâche recouvre donc le premier Mo de la mémoire vive physiquement présente, à moins que la pagination ne soit enclenchée. Si la pagination est active, l'espace mémoire d'une tâche V86 est reproduit à un emplacement quelconque de la mémoire physique par l'intermédiaire du répertoire des pages et d'une table des pages. L'illusion est alors parfaite.

Pendant qu'une tâche est exécutée en mode V86, d'autres tâches peuvent être actives sans être en mode V86. Le mode protégé reste disponible, aussi bien dans la version 16 bits compatible avec le 80286 qu'avec les versions 32 bits du 80386 et de l'i486. Ces modes sont généralement utilisés par un "moniteur de contrôle virtuel" qui gère en arrière plan l'exécution du programme DOS en mode V86 pour lui donner la sensation d'une machine DOS ordinaire. Le détail de ces opérations est évoqué dans les sections consacrées aux émulateurs d'EMS et aux gestionnaires d'exploitation multitâche.

En mode protégé ordinaire, les interruptions et exceptions déclenchées pendant l'exécution d'une tâche V86 sont également traitées. Le processeur se remet en mode protégé où le gestionnaire d'interruptions et d'exceptions est appelé par la porte correspondante de la table des descripteurs d'interruption. L'initialisation de cette table et la mise en service du gestionnaire d'interruptions et d'exceptions n'est pas de la responsabilité de la tâche V86 mais de celle du moniteur de contrôle virtuel qui réglemente le fonctionnement de la tâche.

La commutation du mode protégé en mode V86 peut s'effectuer de plusieurs manières. Le plus souvent elle se fait par une porte de tâche, comme une commutation ordinaire. Le mode d'exploitation de la nouvelle tâche est contrôlé par l'indicateur VM tiré du registre EFLAGS qui est chargé à partir du segment d'état de la nouvelle tâche. Si cet indicateur est à 1, l'exécution se fait en mode V86, sinon c'est le mode protégé ordinaire qui entre en service.

La section consacrée à la commutation de tâche expliquait comment une nouvelle tâche se trouvait amorcée par la création d'un segment d'état de tâche avec une porte de tâche associée. Pour l'exécution d'une tâche en mode V86, il faut encore ajouter la mise à 1 de l'indicateur VM à l'intérieur du segment d'état de tâche pour que l'exécution commence directement en mode V86.

Les tâches V86 se déroulent toujours au niveau de privilège le plus bas (3). Toutes les instructions qui d'une manière ou d'une autre influencent le contenu de l'indicateur d'interruption ou de l'indicateur VM à l'intérieur du registre EFLAGS (PUSHF, POPF, INIT, IRET, CLI et STI) sont soumis au test d'IOPL et ne peuvent être exécutés que si l'indicateur d'IOPL contient la valeur 3. On cherche ainsi à éviter qu'une tâche V86 puisse se commuter en mode protégé. On peut aussi par ce moyen contrôler la modification de l'indicateur d'interruption car les programmes DOS ont tendance à mettre à 0 de temps en temps cet indicateur pour éviter les requêtes INTR. Mais en règle générale ce procédé ne fait pas le bonheur du moniteur de contrôle virtuel car pendant ce temps les demandes d'interruptions des autres tâches s'accumulent et ne peuvent plus être traitées à temps.

C'est pourquoi le mécanisme d'IOPL permet de générer à chaque accès au registre EFLAGS des exceptions qui empêchent la prise en compte de modifications. Mais les exceptions incessantes ralentissant énormément l'exécution du programme, il est recommandé de mettre à l'IOPL 3 les programmes DOS "sûrs".

En mode V86 les commandes IN, OUT, INS et OUTS ne sont pas privilégiées par l'indicateur d'IOPL comme c'est le cas en mode protégé. Mais avant d'exécuter une de ces commandes le processeur teste la table binaire des autorisations d'entrée-sortie dans le segment d'état de la tâche en question pour barrer ou libérer sélectivement les ports. En cas d'accès à un port barré une exception est générée qui rend la main au moniteur de contrôle virtuel. Ce dernier peut alors "virtualiser" certains ports, ce dont nous reparlerons dans la section consacrée au multitâche.

Avec tous ces mécanismes, de la simulation d'un espace d'adressage de mode réel à la pagination en passant par la table binaire des autorisations d'entrée-sortie, le mode V86 est parfaitement équipé pour affronter l'exécution de programmes DOS en multitâche. La section suivante montre comment est réalisée une telle exécution.

33.2. Utilitaires en mode protégé

Le mode V86 présenté précédemment convient parfaitement pour développer des utilitaires en mode protégé qui fonctionnent sous DOS. Ce mode d'exploitation permet en effet de réaliser une sorte de moniteur qui s'exécute en mode protégé mais qui contrôle une (ou plusieurs) machines virtuelles DOS. DOS est donc doté d'un "grand frère" qui contrôle et dirige tous ses pas sans qu'il s'en rende compte.

Les deux sections suivantes montrent le résultat du couplage du mode V86 avec DOS.

33.2.1. Emulateurs d'EMS et programmes de gestion de la mémoire

Les émulateurs d'EMS existent pour toutes sortes de systèmes allant du 8086 à l'i486 en passant par le 80286 et le 80386. Leur rôle est d'initialiser de la mémoire paginée aux normes LIM quand cette mémoire n'existe pas. On les appelle parfois des "limulateurs".

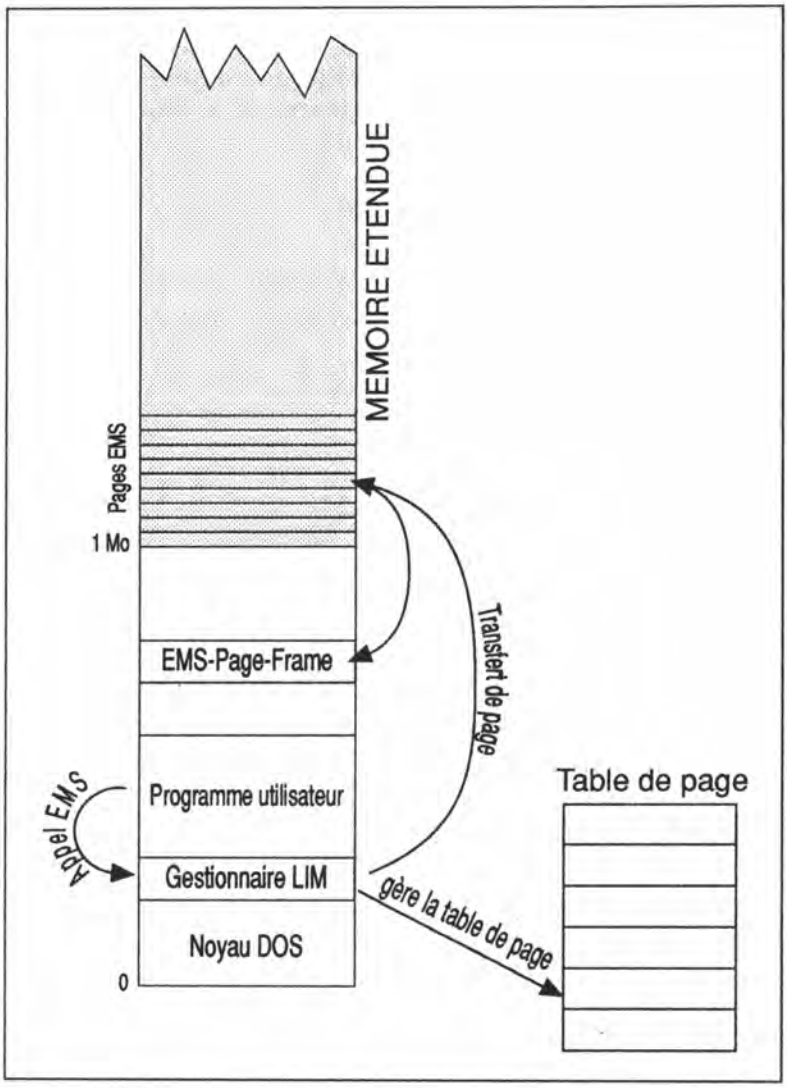
Les limulateurs appliquent différents principes de fonctionnement selon le processeur pour lequel ils ont été développés. D'une génération à l'autre les conditions de mise au point se sont améliorées et leur degré d'efficacité s'est accru. Leur seul point commun reste de préparer une interface logicielle aux spécifications LIM: mais quant à la présence de mémoire EMS ils ne font que tromper les programmes qui font appel à leurs fonctions.

Avec un 8086 l'installation d'un limulateur ne se justifie pas car la mémoire paginée est simulée sous forme d'un fichier sur disque dur. Le temps d'accès est alors excessif. Par ailleurs la fenêtre EMS avec ses 64 Ko de mémoire vive doit être située en dessous de la frontière des 640 Ko, ce qui entraîne un prélèvement douloureux. Les limulateurs pour 8086 ne sont donc pas très efficaces et il faut leur donner une mauvaise note.

Il en est tout autrement avec le 80286. La mémoire EMS émulée peut évidemment être tirée du disque dur, mais ce ne serait pas un choix très heureux. Il vaut mieux se servir de la mémoire étendue située au-delà de la limite de 1 Mo et qui est aujourd'hui presque toujours disponible sur n'importe quel AT. Le limulateur transforme cette mémoire en mémoire paginée en en copiant une partie dans le cadre de page EMS à l'aide d'une fonction spéciale du BIOS. Mais le cadre de page doit être mémorisé en dessous des 640 Ko. Par ailleurs la copie de mémoire étendue en mémoire vive conventionnelle prend pas mal de temps car à chaque fois une commutation en mode protégé est nécessaire, le retour au mode réel étant, avec le 80286, une véritable odyssée à travers les tréfonds de la programmation système.

Plutôt que d'exploiter ainsi de la mémoire paginée, il est souvent plus intéressant, avec les ordinateurs à base de 80286, de faire appel à des cartes d'extension EMS. Il faut dire aussi que de nombreux AT (surtout avec NEAT-BIOS) disposent déjà de la possibilité de configurer de la mémoire étendue en mémoire paginée, ce qui apporte les mêmes avantages.

La véritable percée des limulateurs ne s'est faite qu'avec le processeur 80386. De nombreuses caractéristiques favorisent à ce titre l'exploitation de ce processeur. La mémoire est prélevée dans la mémoire étendue au-delà de la limite de 1 Mo, mais contrairement à ce qui se passe avec le 80286, aucune recopie dans le cadre de page n'est nécessaire.



Fonctionnement d'un limulateur pour le 80386 et ses successeurs

En effet le 80386 et ses successeurs disposent d'un mécanisme de pagination performant grâce auquel il est aisé de reproduire de la mémoire étendue dans le cadre de page EMS tout en plaçant ce dernier entre 640 Ko et 1 Mo, même s'il n'existe pas de mémoire physique à cet endroit. C'est en tout cas ce que font tous les gestionnaires de mémoire et limulateurs un peu sophistiqués tels que 386-to-the-Max, QEMM-386 ou EMM386 de DOS 5.0.

Mais il y a quand même un problème. Car en mode réel, à partir duquel les programmes DOS réclament et exploitent de la mémoire EMS, le mécanisme de pagination du 80386

et de ses successeurs ne fonctionne pas. Il faut donc faire appel au mode V86 ce qui signifie que le limulateur au moment de son lancement bascule l'ordinateur en mode V86. Il doit donc se situer au-dessus de DOS comme moniteur de contrôle virtuel. La section suivante consacrée au multitâche vous en apprendra davantage sur ce sujet.

Programmes de gestion de la mémoire

Les gestionnaires de mémoire comme 386-to-the-Max ou QEMM-386 n'offrent pas seulement des limulateurs mais ils supportent également l'interface XMS pour accéder à la mémoire étendue. En plus -ce qui est encore plus intéressant- ils permettent de déplacer des programmes résidents, des drivers et des parties de DOS pour les disposer au delà de la barrière des 640 Ko, ce qui libère d'autant la mémoire conventionnelle.

Ces programmes sont chargés dans des zones de mémoire que la table des pages situe entre 640 Ko et 1 Mo sans que ces emplacements existent réellement. Le plus difficile n'est pas de programmer les tables de pages en conséquence mais de s'assurer que les zones de mémoire en question ne sont pas déjà revendiquées par des cartes ou d'autres logiciels. Il est clair que des zones de mémoire attribuées à plusieurs acteurs garantissent un plantage système immédiat.

33.2.2. Exploitation multitâche

Depuis que les PC frisent le niveau de performance des postes de travail, les utilisateurs de DOS se font plus pressants en demandant les avantages qu'offrent ce type de matériel, et notamment une possibilité d'exploitation multitâche.

En l'occurrence il s'agit moins d'exécuter simultanément plusieurs programmes que de pouvoir passer de l'un à l'autre sans être obligé de procéder à des clôtures et des réinitialisations. Il existe depuis longtemps des programmes qui effectuent ce genre de "commutation de tâche" pour un PC. Le plus connu est sans doute DESQView de Quarterdeck qui a trouvé un solide concurrent dans le "Commutateur de tâches" du shell de DOS 5.0.

Dans le monde de DOS, les vrais gestionnaires d'exploitation multitâche s'appellent essentiellement DESQView/386 et Windows 3.0 qui permettent l'exécution en parallèle de plusieurs programmes DOS et leur affichage dans des fenêtres écran séparées. Pour parvenir à ce résultat, les problèmes à résoudre sont nombreux et ils ne sont pas toujours du ressort de la programmation système mais plutôt de celle de la théorie des systèmes d'exploitation. Un gestionnaire d'exploitation multitâche n'est au fond rien d'autre qu'un super système d'exploitation qui coiffe DOS pour lancer d'autres systèmes dans les machines virtuelles qu'il contrôle.

Comme cet ouvrage est consacré à la programmation système, je ne souhaiterais pas examiner ici les questions théoriques que soulève le développement d'un tel super

système, quel que soit l'intérêt qui s'y attache. Je voudrais plutôt me tourner vers un problème qui touche la programmation système et pour la résolution duquel le mode V86 est irremplaçable: il s'agit du problème de la "virtualisation du matériel".

Ce n'est que par ce moyen que chaque programme exécuté dans un environnement multitâche peut croire à l'existence de son PC propre où il se trouve tout seul. Par nature les programmes de DOS ne sont pas coopératifs et leur comportement d'égoïste invétéré les amène à écrire directement dans la mémoire d'écran, à programmer sans ménagement le contrôleur d'interruption ou à tirer à eux toute la mémoire disponible.

Virtualisation du matériel

Les problèmes qui surgissent sont particulièrement apparents si l'on considère l'exemple de l'écran. Normalement chaque programme DOS peut supposer qu'il dispose d'un espace d'affichage de 25 lignes de 80 colonnes, c'est-à-dire de la totalité de l'écran pour son usage exclusif. Mais avec un gestionnaire d'exploitation multitâche ce n'est plus le cas. L'espace d'affichage est alors réduit à une petite fenêtre. Le gestionnaire d'exploitation multitâche va rassembler les sorties dans un buffer interne et représenter une partie de ce buffer dans la fenêtre attribuée par l'utilisateur au programme concerné.

Pour que l'utilisateur puisse contempler tous les affichages du programme, le gestionnaire d'exploitation multitâche permet de faire défiler le contenu de la fenêtre : de nouveaux extraits de l'"écran virtuel" deviennent alors visibles. Il est par ailleurs possible d'étendre l'écran virtuel à la totalité de l'écran physique: vous connaissez cette manoeuvre si vous utilisez Windows.

Voilà le point de vue de l'utilisateur. Mais que se passe-t-il dans les coulisses, comment le gestionnaire d'exploitation multitâche collecte-t-il les affichages des différents programmes ?

Tant que le programme utilise les fonctions vidéo du BIOS pour réaliser ses affichages, le problème est simple à résoudre. il suffit que le gestionnaire d'exploitation multitâche déroute l'interruption du BIOS sur une fonction qui lui appartienne et qui enverra les caractères dans un buffer interne (l'écran virtuel) avant de les faire apparaître à l'écran.

Le détournement de l'interruption vidéo du BIOS ne permet pas seulement de capter les affichages provoqués par l'appel direct au BIOS. Toutes les fonctions d'affichage de DOS et les instructions des langages de haut niveau telles que PRINT, printf() ou writeln() (avec DirectVideo=False) se soldent en définitive par des appels au BIOS. Mais que se passe-t-il lorsque le programme en question n'exploite aucune des possibilités mentionnées et qu'il accède directement à la mémoire écran, ce qui est un moyen courant d'accroître la vitesse d'affichage ?

Avec les ordinateurs à base de 8086/8088 ou 80286 la partie est perdue d'avance car il n'existe aucune possibilité d'intercepter les accès directs à une mémoire. Avec le

80386 et ses successeurs, les choses sont différentes car on dispose du mode V86 et du mécanisme de pagination.

Si vous avez lu la section consacrée à la pagination du 80386 et de ses successeurs, vous savez que ce mécanisme reproduit l'adresse linéaire de la mémoire physique. Un gestionnaire d'exploitation multitâche ne peut pas se permettre de renoncer à cette possibilité car il doit donner à tous les programmes DOS en concurrence l'impression de s'exécuter dans le premier Mo de la mémoire vive, même s'ils sont en réalité situés ailleurs.

Mais pour virtualiser la mémoire écran il est encore plus intéressant d'exploiter les possibilités intrinsèques de la pagination pour réaliser une gestion de mémoire virtuelle. C'est le bit de présence qui est particulièrement précieux. Rappelons qu'il s'agit d'une partie de l'indicateur qui fait partie de chaque élément de la table des pages.

Si ce bit est à 0, le processeur sait en accédant à une page que cette dernière n'est pas en mémoire. Il déclenche alors une exception pour que le système reprenne le contrôle des opérations et puisse recharger la page manquante. L'exception est déclenchée même quand il n'existe pas de gestion de la mémoire virtuelle, elle doit toujours être gérée par une routine appropriée du système d'exploitation.

Le gestionnaire multitâche va alors marquer comme étant absentes les pages qui correspondent à la mémoire écran. A chaque accès à la mémoire écran, la routine d'exception du gestionnaire d'exploitation multitâche va être appelée, stockant l'octet ou le mot en voie d'affichage dans l'écran virtuel d'où il sera extrait pour être affiché un peu plus tard.

En pratique cette manière de procéder permet même de se dispenser du détournement de l'interruption vidéo du BIOS car cette dernière en fin de compte effectue également une écriture directe dans la mémoire écran, ce qui va déclencher la routine d'exception du gestionnaire multitâche. Malheureusement, compte tenu des microprocesseurs actuellement disponibles, cette méthode reste un peu dispendieuse en terme de temps d'exécution. C'est pourquoi les gestionnaires d'exploitation multitâche effectuent tout de même le détournement de l'interruption vidéo du BIOS.

Ce que nous venons de voir au sujet de la mémoire écran est aussi valable pour n'importe quelle zone de mémoire qui peut ainsi être prémunie contre les accès directs. En fait les zones concernées ne peuvent pas se restreindre à des mémoires individuelles, elles doivent en principe s'étendre sur au moins une page entière, soit 64 Ko. Mais comme la routine d'exception du gestionnaire multitâche reçoit comme argument l'offset de l'accès, on peut quand même protéger des zones inférieures à 64 Ko. Si la routine d'exception constate que la mémoire est en dehors de la zone à protéger, elle met à 1 le bit de présence, effectue elle-même l'accès et réinitialise ensuite à 0 le bit de présence.

Interception des accès aux ports d'entrée-sortie

Les collisions susceptibles de se produire dans un système multitâche n'ont pas pour seule cause des accès concurrents à une même zone de mémoire. Le même accident risque d'affecter les ports d'entrée-sortie qui servent à commander les cartes matérielles. Revenons encore une fois à notre exemple d'affichage sur l'écran. Que se passe-t-il par exemple si un programme accède directement aux registres de la carte graphique pour mettre en service le mode graphique ou sélectionner une autre palette de couleurs ? Comment le gestionnaire multitâche peut-il s'en rendre compte ?

Ici encore c'est une caractéristique particulière du mode protégé, la table binaire des autorisations d'entrée-sortie, qui va nous être utile. En mode V86 elle est toujours active et permet au gestionnaire multitâche de protéger les ports sensibles contre les accès inconsidérés. Lorsqu'un programme s'adresse à un port interdit, une exception est déclenchée qui avertit le gestionnaire multitâche.

C'est ce que fait Windows en mode étendu lorsqu'une boîte DOS en mode V86 exploite le contrôleur DMA qui intervient dans l'accès aux disquettes et au disque dur. Ce circuit ignore tout de l'existence d'une gestion des pages et il prend donc les adresses qu'on lui communique pour des adresses physiques réelles auxquelles il cherche à accéder. Windows est obligé d'intercepter chaque accès à ce circuit dans le cadre de la boîte DOS pour convertir les adresses indiquées en adresses physiques véritables avant qu'elles ne parviennent au contrôleur.

Les gestionnaires multitâches ont encore bien d'autres problèmes à surmonter mais la combinaison du mode V86, de la pagination et de la table des autorisations d'entrée-sortie en vient généralement à bout. Même des programmes DOS particulièrement peu coopératifs peuvent ainsi être exécutés dans un environnement multitâche, comme le montre d'ailleurs Windows 3.0.

Mais il existe malgré tout un problème insoluble: la dispute pour le mode protégé entre les gestionnaires multitâche et les DOS Extenders.

Si un programme développé à l'aide d'un DOS Extender est lancé en exploitation multitâche, il essayera en vain de se mettre en mode protégé. Ce n'est pas pour rien en effet qu'une tâche en mode V86 se déroule au niveau de privilège le plus bas.

Heureusement il existe des interfaces logicielles appelées DPMI et VCPI qui viennent à la rescousse. Elle sont présentées en section 33.4.

33.3. Les DOS Extenders

D'un côté les données et les programmes ont besoin de mémoire, de l'autre côté la mémoire est présente sous forme étendue. Mais il n'y a pas de rencontre. Un scénario

typique du mode réel sous DOS. Bien sûr, les interfaces logicielles telles que EMS ou XMS donnent accès à davantage de mémoire mais uniquement pour les données et aux prix de mécanismes compliqués difficiles à mettre en harmonie avec la logique interne des programmes. Les développeurs ont depuis longtemps demandé à pouvoir utiliser la mémoire étendue comme de la mémoire vive ordinaire dans leurs programmes DOS. Les DOS Extenders leur donnent enfin satisfaction.

Il s'agit d'outils de développement proposés par différentes sociétés d'édition de logiciels américaines dans une optique de collaboration avec certains compilateurs bien précis. Ce sont surtout les compilateurs C standard qui sont visés par les DOS Extenders mais peut-être aussi Turbo Pascal bien que je ne connaisse pas de produit associé. Dans les sections consacrées à la mise en service concrète de DOS Extenders nous parlerons uniquement de produits qui s'appliquent à des développements en langage C.

Les DOS Extenders sont adaptés spécialement à des compilateurs bien précis car ils manipulent le code machine généré par eux. Leur ambition est de faire fonctionner des programmes DOS dans le mode protégé du 80286, 80386 ou i486. Un programme en mode protégé a à sa disposition l'intégralité de la mémoire vive et non seulement l'espace restreint de 640 Ko que DOS lui-même ne libère pas entièrement.

Mais exécuter des programmes DOS en mode protégé est presque aussi difficile que la quadrature du cercle. Mais l'objectif peut être atteint comme le montrent des logiciels connus qui ont été développés avec des DOS Extenders: parmi eux on compte par exemple, en plus de la version 10.0 d'AutoCAD, la version 3 de Lotus 1-2-3 ou une version spéciale 386 de Paradox. Il n'est pas étonnant que de telles célébrités logicielles aient été candidates à l'intervention des DOS EXtenders. Du fait de leur puissance, ces logiciels comportent beaucoup d'instructions et gèrent une énorme quantité de données et de variables en mémoire centrale.

Mode de fonctionnement d'un DOS Extender

En termes simplifiés, la tâche d'un DOS Extender peut se résumer en une formule sommaire: exécuter le programme en mode protégé mais revenir au mode réel à chaque appel à DOS ou au BIOS. Le DOS Extender a modifié le code de l'application dans le fichier EXE de façon à ce qu'il tourne en mode protégé. Mais il n'a pas pu faire la même chose pour DOS ou le BIOS. Au moment de l'exécution de nombreuses difficultés apparaissent.

Si le principe de fonctionnement est simple, les détails sont d'une grande complexité. Un DOS Extender est livré avec divers utilitaires qui agissent sur le code compilé des compilateurs DOS (par exemple MSC 5.1 ou 6.0). Le programme reçoit d'abord une nouvelle adresse de lancement derrière lequel se cache le code de l'Extender.

Quand on lance le programme à partir de la ligne de commande DOS, c'est en fait le DOS Extender incorporé qui prend le contrôle. Dans un premier temps il copie le

programme en mémoire étendue, sans toutefois le mettre à exécution car il reste encore des travaux préliminaires à effectuer. Le DOS Extender installe alors toujours en mode réel une table de descripteurs globale qui définit les différents segments de code et de données du programme. Les informations nécessaires ont été tirées du fichier OBJ généré par le compilateur, au moment du développement.

C'est également au moment de la compilation qu'est défini l'ordre dans lequel sont créés les différents descripteurs de segments. Dans les références aux segments qui apparaissent dans le code et les données se trouvent déjà des sélecteurs qui ne sont plus modifiés par la suite. Une adaptation des références à l'instar de ce que fait le chargeur de DOS en mode réel n'est pas nécessaire car en fin de compte les adresses des segments étant dans les descripteurs, elles peuvent y être modifiées.

Le programme est prêt pour le décollage, il manque simplement la table des descripteurs d'interruption et le gestionnaire d'interruption que réclament les fonctions de DOS et du BIOS. Une fois qu'ils sont installés, plus rien ne s'oppose à l'exécution en mode protégé. Le DOS Extender commute donc le processeur en mode protégé et lance le programme.

Pendant que le programme tourne, le DOS Extender agit uniquement à l'arrière plan, par le moyen des fonctions de DOS et du BIOS dont le traitement particulier est décrit plus loin. Lorsque le programme se termine d'une façon normale, autrement dit par l'invocation de la fonction 4Ch de DOS, le DOS Extender reprend le contrôle des opérations. Il fait le ménage dans la mémoire centrale et repasse en mode réel, ce qui le ramène au niveau de DOS sans que l'utilisateur ait remarqué la mise en service du mode protégé.

DOS Extenders pour 286 et 386/i486

Le marché des DOS Extenders se divise en deux segments: les DOS Extenders pour 80286 et les DOS Extenders pour 80386 ou i486. Les derniers cités exploitent évidemment les possibilités étendues du processeur 80386 et de ses successeurs (32 bits, segments plus longs) ce qui leur donne un niveau de performance bien supérieur à ce qu'il est possible d'obtenir avec un 80286. Mais leur mise en service demande souvent des modifications à l'intérieur du code source, ce qui n'est pas nécessairement le cas pour un 80286. Nous approfondirons cette question dans les deux sections consacrées d'une part aux DOS Extenders pour 80286 et d'autre part aux DOS Extenders pour 80386/i486. Mais commençons par jeter un coup d'oeil très général sur les problèmes posés par l'exécution d'un programme DOS en mode protégé.

33.3.1. Les exigences du mode protégé

Un programme DOS en mode protégé se voit d'abord confronté à une gestion de mémoire modifiée, où interviennent des sélecteurs, des descripteurs de segment et des

tables de descripteurs. Il n'est plus question d'adresses de segments ou de segments constants de 64 Ko. Ce problème est essentiellement résolu par les manipulations qu'effectue le DOS Extender sur le code compilé avant même le premier lancement du programme.

Les choses deviennent plus compliquées lorsque le programme fait appel à des fonctions de DOS ou du BIOS, ou lorsque des périphériques externes déclenchent des interruptions. Ni le code de DOS, ni la ROM du BIOS ni les gestionnaires d'interruptions dans les programmes résidents n'ont été conçus pour fonctionner en mode protégé. Si on appelle leurs fonctions en mode protégé, on occasionnerait un plantage immédiat du système, dès le premier chargement d'un segment dans l'un des registres de segment.

L'un ou l'autre développeur amateur de portabilité pourrait prétendre que ses programmes ne comportent aucun appel à DOS ou au BIOS sous prétexte qu'ils tournent aussi sur un autre système XYZ, mais une telle prétention est insensée. Car même si un programme ne contient aucun appel explicite aux fonctions de DOS ou du BIOS, il est rempli d'appels cachés. En effet les fonctions de traitement des fichiers, d'affichage à l'écran ou de saisie au clavier offertes par les langages évolués s'appuient toujours sur des appels à DOS ou au BIOS.

Mais que se passe t-il exactement si pendant l'exécution d'un programme une fonction de DOS ou du BIOS est appelée par le moyen de l'interruption logicielle associée ou si le clavier déclenche une interruption matérielle ? Le DOS Extender prend aussitôt les choses en main car avant le démarrage du programme proprement dit il a installé ses propres gestionnaires d'interruption dans la table des descripteurs d'interruption. Le DOS Extender a ainsi manipulé les gestionnaires d'interruption de DOS 20h, 21h, 24h, etc.., les gestionnaires d'interruption du BIOS 10h, 11h, 12h, 13h, 14h, 15h, 16h, etc.. ainsi que les interruptions matérielles 08h, 09h, 0Ah etc...

Pour ce qui est des requêtes matérielles IRQ0 à IRQ7, il faut signaler qu'elles déclenchent normalement les mêmes interruptions que les différentes exceptions du mode protégé, à savoir les interruptions 08h à 0Fh. La plupart des DOS Extenders reprogramment donc le contrôleur d'interruption de façon qu'il déroute les requêtes IRQ0 à IRQ7 sur d'autres interruptions. Ce n'est qu'à ce prix que les interruptions et les exceptions peuvent coexister pacifiquement.

Mais revenons aux gestionnaires d'interruption du DOS Extender qui doivent être exécutés à partir du mode protégé. Leur rôle consiste à passer en mode réel, à appeler le gestionnaire d'interruption d'origine par le moyen de la table des vecteurs d'interruption et à revenir ensuite au mode protégé. L'exécution de l'interruption est alors terminée et le programme interrompu reprend son cours. La commutation temporaire en mode réel est complètement transparente pour le programme.

Un aller-retour pour le mode protégé

Le passage incessant du mode protégé au mode réel et vice-versa coûte beaucoup de temps. C'est surtout le 80286 qui cause souci car s'il est bien possible de passer du mode réel au mode protégé, le retour en sens inverse est un parcours semé d'embûches. Un AT moderne avec une cadence de 16MHz peut faire 2000 commutations par seconde du mode protégé au mode réel, aller-retour. Un 386 à 20 MHz est capable de mener à bien 9000 commutations dans le même temps, soit plus de quatre fois plus. L'amélioration tient moins à l'augmentation de la cadence qu'au support du retour de commutation du mode protégé en mode réel.

2000 ou 9000 commutations, on peut interpréter ces chiffres comme on veut. Mais ils sont quand même faibles comparés aux centaines de milliers d'instructions par seconde d'un processeur. Mais la vraie question est de savoir combien d'appels DOS ou BIOS ou combien d'interruptions matérielles se déclenchent habituellement à chaque seconde. Probablement pas beaucoup, surtout lorsque les programmes consistent davantage à traiter de grandes quantités de données qu'à gérer l'interactivité avec l'utilisateur. Ces sont justement ces programmes-là qui sont candidats à la mise en service de DOS Extenders parce que les 640 Ko standard ne leur suffisent pas. La perte de temps entraînée par les commutations d'un mode à l'autre n'est donc pas a priori un frein à l'usage des DOS Extenders.

Transmission d'adresses de buffers

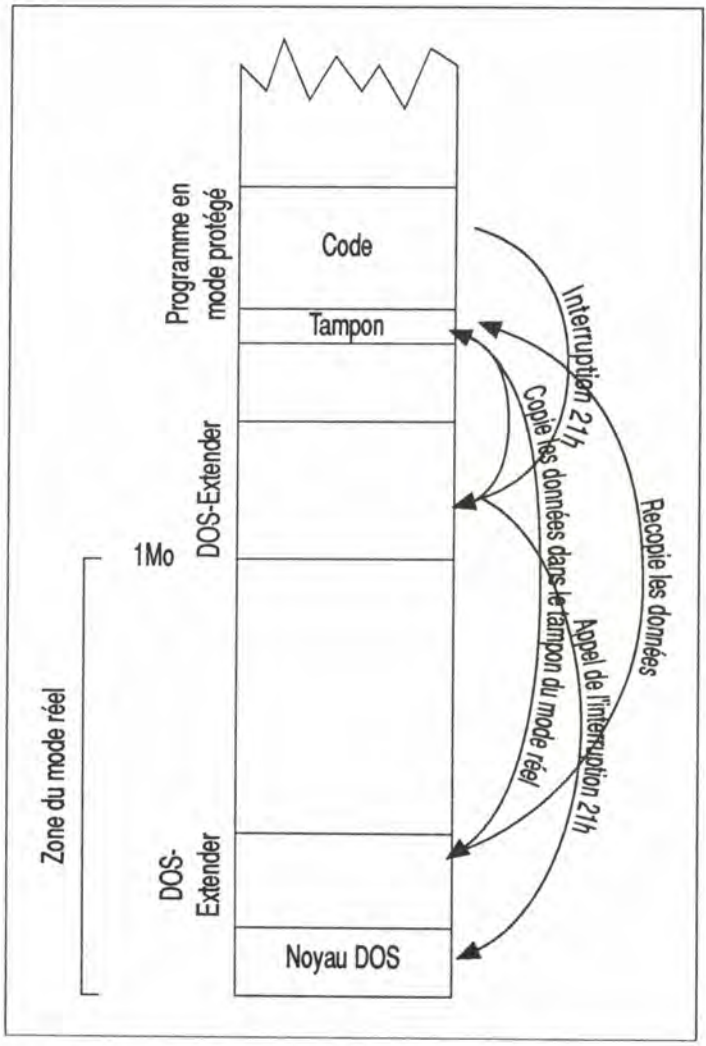
La commutation du mode protégé en mode réel ne résout pas tout, surtout pas pour l'interruption 21h de DOS et ses fonctions. Car ces fonctions vont lire des paramètres dans les différents registres du processeur et la plupart du temps y retourner des résultats. Une fois la commutation en mode réel effectuée, le gestionnaire d'interruption du DOS Extender pourrait transmettre ces informations telles quelles au gestionnaire d'origine de DOS mais la conséquence en serait rapidement un plantage du système. Plus précisément, ce sera le cas si la fonction DOS en question attend comme paramètre une adresse de buffer.

En effet avant d'appeler la fonction de DOS le programme va charger soigneusement l'adresse du buffer dans le registre prévu mais cette adresse ne fournit pas un segment mais un sélecteur. Le sélecteur ne correspond évidemment pas au segment du buffer, c'est ainsi que la fonction de DOS va peut-être accéder au segment 104h alors qu'en fait il s'agissait du sélecteur 104h qui désignait un segment quelconque au-delà de la limite de 1 Mo.

Le DOS Extender est donc obligé, avant d'appeler la fonction, de transformer les références concernant des buffers du mode protégé en adresses de segments. Il ne suffit pas à cet effet de consulter une table des decsripteurs globale ou locale, car le résultat en serait une adresse de base à 24 ou 32 bits, mais sûrement pas une adresse de segment

à16 bits. Par ailleurs l'adresse en mode protégé ne pourra être que rarement exprimée sous forme de segment ordinaire car le programme et ses buffers sont généralement situés au-delà de la limite de 1 Mo, ce qui les rend inaccessibles aux fonctions du mode réel de DOS.

Avant d'appeler la fonction, le DOS Extender va donc recopier le contenu du buffer d'origine dans un buffer provisoire installé en-dessous de la fameuse limite de 1 Mo. Ce déménagement devra être renouvelé à l'issue de l'appel car DOS pourrait avoir modifié le buffer. Le contenu du buffer provisoire devra donc être recopié dans le buffer du mode protégé.



Transmission d'un buffer lors de l'appel d'une fonction de DOS ou du BIOS

Ce déplacement d'informations fait évidemment perdre du temps, surtout lorsqu'on accède à des fichiers qui se caractérisent souvent par des blocs de données importants. Mais après tout, dans ce cas-là, le temps de recopie est négligeable par rapport au temps d'accès aux supports externes. Il ne faut pas oublier que plusieurs puissances de dix séparent un temps de lecture/écriture sur disque du délai d'exécution d'une instruction REP MOSW.

Le DOS Extender encadre donc les différentes fonctions de DOS et les interruptions du BIOS appelés par le programme. Dans le mode de DOS un tel comportement n'est pas une grande nouveauté, car les programmes résidents et les extensions système l'ont adopté depuis longtemps en modifiant à leur guise le fonctionnement de diverses fonctions.

Si la copie d'informations entre buffers en mode réel et buffers en mode protégé est une source de complications et de perte de temps, elle n'est cependant pas la plus grande difficulté rencontrée par les développeurs de DOS Extenders.

IL faut savoir que la plus grande variabilité règne dans ce domaine : toute fonction de DOS n'attend pas forcément une adresse de buffer, la transmission a lieu par différents registres et le codage de la longueur n'est pas toujours le même. En fait le DOS Extender doit être un intime connaisseur des fonctions de DOS. Chacune d'entre elles doit être traitée individuellement: pour l'une les registres EX:BX pointent sur un buffer tandis que pour telle autre ils mémorisent deux handles de fichier qui peuvent être repris directement en mode réel.

L'affaire se complique évidemment lorsqu'un programme exploite des fonctions de DOS non documentées. Dans ce cas le DOS Extender ne connaît pas le rôle des registres et il ne peut pas effectuer la commutation en mode réel. Il faut donc s'en tenir à une règle stricte: une application destinée à être soumise à un DOS Extender ne doit donc pas comporter de fonction DOS non documentée.

Des problèmes similaires surgissent également en liaison avec d'autres interfaces logicielles fréquemment utilisées, par exemple avec la commande de la souris (interruption 33h) ou avec les fonctions du NetBIOS (interruption 5Ch) qui pilotent un réseau. Si le DOS Extender ignore ces fonctions, le programmeur sera responsable de la conversion des sélecteurs en adresses de segments et de la copie des buffers. En général le DOS Extender offre d'ailleurs des fonctions spéciales pour cet usage.

Gestion de la mémoire

Mais revenons aux fonctions de DOS. Il arrive que le DOS Extender ne se contente pas de les compléter mais soit obligé de les remplacer carrément par son propre code. C'est ce qui se passe pour les fonctions qui gèrent la mémoire en allouant, modifiant ou libérant des zones situées en-deçà de la limite des 640 Ko. Pratiquement tous les programmes

- y compris ceux générés par les compilateurs évolués- exploitent ces fonctions pour se réserver de la mémoire.

Mais DOS ne leur donne jamais plus de mémoire qu'il ne s'en trouve dans la limite des 640 Ko. Mais s'il le faisait, la plupart des programmes sauraient l'exploiter, à condition évidemment que cette mémoire se trouve en dessous de la limite de 1 Mo et soit ainsi adressable en mode réel. Mais un programme réalisé à l'aide d'un DOS Extender ne doit pas seulement exploiter la mémoire en dessous de 1 Mo mais aussi celle qui est située au-delà. Et pour identifier les différentes zones de mémoire il ne lui faut pas des adresses de segments mais des sélecteurs.

Tout cela suffit pour que le DOS Extender décide de prendre en charge lui-même la gestion de la mémoire, remplaçant les fonctions de DOS prévues à cet effet par les siennes propres. Lorsqu'un programme demande la quantité de mémoire disponible, ces fonctions peuvent très bien répondre qu'il reste 1 Mo. Il n'est pas possible de dépasser ce chiffre car le schéma de transmission de la fonction concernée ne le permet pas (il faut charger en BX le nombre de paragraphes de 16 octets). Dans l'étape qui suit le programme va réclamer la mémoire requise et probablement aller jusqu'à la limite renseignée par un appel de fonction précédent. Le DOS Extender attribue ainsi 1 Mo de mémoire qui peuvent être adressés par le sélecteur fourni. Comme la plupart des programmes demanderont plusieurs fois de la mémoire à des fins diverses, toute la mémoire physiquement disponible peut leur être allouée.

Il se pose cependant un problème pour la mise à disposition des zones de mémoire qui dépassent 64 Ko. En effet si le code du programme a été créé sur un processeur 80286 ou sur un de ses successeurs, il ne fonctionne qu'avec des registres de 16 bits et ne peut atteindre que des segments de 64 Ko. Lorsque de grandes zones de mémoire sont parcourues, l'adresse de segment doit être modifiée de temps à autre pour sortir des premiers 64 Ko. Mais l'arithmétique des adresses de segment est interdite en mode protégé car on obtient des sélecteurs qui référencent de toutes autres zones de mémoire.

Le DOS Extender éprouve aussi des difficultés à allouer des blocs de plus de 64 Ko à un 80286 car un segment ne peut simplement pas dépasser cette taille. Pour allouer un grand bloc de mémoire, le DOS Extender est obligé de créer plusieurs segments et par conséquent plusieurs descripteurs de segments dans la table des descripteurs globale ou locale. Ces descripteurs à leur tour nécessitent plusieurs sélecteurs que la fonction de DOS ne fournit pas au programme appelant.

Pour les allocations de plus de 64 Ko le 80286 requiert une intervention du DOS Extender qui va au-delà du traitement habituel des fonctions de DOS. La section suivante va examiner ce point particulier.

Ce n'est pas seulement l'arithmétique des segments qui représente un obstacle insurmontable en mode protégé. Les références absolues à certains segments constituent une autre plaie. Le DOS Extender peut encore installer des descripteurs de segment pour pointer sur la table des vecteurs d'interruption et la zone des variables du BIOS en prenant des numéros égaux aux segments mais avec les segments de la mémoire écran

c'est plus difficile. La table des decripteurs globale ou locale - selon le cas- doit être étendue à B800h+1 éléments pour que le segment de la mémoire écran corresponde au descripteur de numéro B800h. La table des descripteurs atteint alors la taille respectable de 360 Ko, avec une majorité d'éléments inutiles étant donné qu'aucun DOS Extender n'a besoin de B800h segments pour exécuter un programme.

Les DOS Extender exploitent donc une autre voie. Ils reproduisent les segments par des descripteurs accolés à la fin de la table des descripteurs avec un indice sans rapport avec les adresses des segments. Ils offrent ensuite des fonctions spéciales qui à partir du sélecteur permettent d'accéder directement à la mémoire associée.

33.3.2. DOS Extenders pour 80286

On estime en général que les DOS Extenders pour 80286 donnent des programmes moins bons que les DOS Extenders pour 80386 et i486 mais que la conversion des programmes existants est plus facile.

Ainsi lorsque la conversion est effectuée avec l'un des DOS Extenders les plus connus pour 80286 (DOS/16M de Rational Systems) il suffit de recompiler le source avec le modèle large, d'enclencher une édition de liens un peu modifiée et de soumettre le tout au DOS Extender. Au bout de cinq minutes on dispose d'un programme qui tourne sur un ordinateur sans afficher "insufficient memory" et en profitant de plus de mémoire que jamais.

Tout fonctionne très bien à condition de respecter les règles décrites précédemment, surtout celles qui concernent le maniement des segments. Mais comment cela est-il possible alors qu'aucune fonction spéciale du DOS Extender n'a été invoquée ? Que se passe-t-il exactement lors de la conversion du programme ?

Réalisation d'un programme en mode protégé avec DOS/16M

Au moment de la compilation il ne se passe pas encore grand-chose. Le programme est traité avec le modèle mémoire large pour que les pointeurs de données et de code soient de type FAR. Les données et le code peuvent ainsi franchir la frontière des 64 Ko, ce qui est une condition minimale d'intervention du DOS Extender.

Le programme objet obtenu après compilation est d'abord un programme ordinaire en mode réel mais l'étape suivante va préparer sa conversion.

Lors du processus d'édition des liens le programme ne va pas être lié aux routines de la bibliothèque du compilateur C mais à des modules additionnels issus du DOS Extender. Le programme reçoit ainsi une nouvel en-tête de démarrage, et à la place des routines de la bibliothèque C sont insérées des routines propres au DOS Extender. Par ailleurs une place est préparée pour l'installation ultérieure des tables de descripteurs. La

nouvelle routine de démarrage prend soin d'arrêter immédiatement l'exécution du programme lorsque le processeur est un 8086 ou si la mémoire étendue est insuffisante.

L'édition de liens génère un fichier EXE qui ne peut plus tourner dans le mode réel de DOS et qui ne n'est pas encore fonctionnel pour le mode protégé. Il doit être soumis à un programme de conversion qui dans le cas du DOS Extender DOS/16M s'appelle MAKEPM et transforme le fichier EXE en un fichier EXP. D'une certaine manière MAKEPM prend en charge la tâche dont s'occupe habituellement le chargeur de DOS: il ajuste les adresses des segments à leur situation physique en mémoire.

Il est vrai qu'un programme en mode protégé n'a pas besoin d'adresses de segments mais uniquement de sélecteurs. MAKEPM remplace les différentes adresses de segments par des numéros de sélecteurs, ces numéros réflétant l'ordre dans lequel les descripteurs de segments seront ultérieurement chargés dans la table des descripteurs.

Une fois soumis à MAKEPM le fichier généré ne peut plus être lancé à partir de la ligne de commande de DOS : d'ailleurs il ne porte plus l'extension EXE. Mais même si on changeait l'extension il ne fonctionnerait pas car pour être lancé il doit être confié à un chargeur spécial. Ce chargeur va extraire les segments du fichier EXE, les placer en mémoire et noter leur emplacement dans la table des descripteurs locale.

Il faut donc finalement deux programmes: le chargeur mis à disposition par le DOS Extender et le programme EXP. Comme la plupart des utilisateurs n'aiment pas cela, la quasi-totalité des DOS EXtenders fournissent des utilitaires qui permettent d'enregistrer le chargeur dans le fichier EXP. Dans le cas de DOS/16M cet utilitaire s'appelle SLICE et crée un fichier EXE tout à fait ordinaire qui comme toute application peut être lancée à partir de la ligne de commande de DOS.

Fonctions de bibliothèque modifiées

La réalisation d'un programme en mode protégé à l'aide d'un DOS Extender est une opération assez étonnante, ne serait-ce que parce qu'elle implique un compilateur en mode réel tel que MSC ou Turbo C. Mais à y regarder de plus près la différence entre instructions machine en code réel et en code protégé n'est pas si importante que ça, abstraction faite de la mise en service des sélecteurs. Les sélecteurs cependant ont la même longueur que des adresses de segments, ils sont mémorisés et traités de la même façon. Autrement dit les ajustements nécessaires sont raisonnables et peuvent être effectués par un "post-processeur" comme MAKEPM qui ne touche pas vraiment au code original.

Il en est tout autrement des fonctions de bibliothèque qui sont proposées par le compilateur et sont incluses dans le programme au moment de l'édition des liens. Certaines de ces fonctions contreviennent aux principes du mode protégé, elles sont donc remplacées par d'autres. Parmi elles se trouvent les fonctions qui modifient le code (notamment les fonctions d'interruption int86() et int86x() telles qu'elles sont implémen-

tées par les compilateurs C sous MS-DOS). Ou encore toutes les fonctions qui jouent sur l'artithmétique des segments.

L'une des fonctions les plus importantes lorsqu'on se préoccupe de la portabilité en C est la fonction malloc(). Elle est l'instrument central de l'allocation dynamique de mémoire. Contrairement à l'ancienne version, la nouvelle fonction malloc() du DOS Extender est en mesure de distribuer progressivement toute la mémoire étendue au programme qui le requiert. Elle renvoie à cet effet le sélecteur et l'offset de la zone de mémoire allouée. Lorsqu'un programme réserve de la mémoire par la fonction malloc() et non par les fonctions du DOS, ces dernières n'entrent pas en service.

Détection des erreurs de pointage

La programmation en C exploite abondamment les pointeurs, mais c'est là précisément une chance. Car les plantages et dysfonctionnements d'un programme en C résultent souvent de pointeurs erronés qui référencent des zones de mémoire situées au-delà de leur portée. Si on suppose qu'à chaque appel de malloc() un seul objet mémoire est alloué, les erreurs de ce type sont plus faciles à détecter en mode protégé. Il suffit que malloc() à chaque appel crée un nouveau segment ainsi que son descripteur associé qui contiendra entre autres la taille mémoire requise par l'appelant. Si au moment de l'accès à ce segment un pointeur va trop loin ou adopte un numéro de sélecteur invalide, une exception est immédiatement déclenchée ce qui permet de repérer la ligne du programme en cause.

Ce principe n'est toutefois guère applicable en pratique car la table des descripteurs globale ne peut mémoriser que 8191 decripteurs au plus et il faut retrancher de ce nombre tous ceux qui sont exploités pour d'autres segments de mémoire. Si on fait trop souvent appel à la fonction malloc(), la table des descripteurs globale se remplit rapidement et en conséquence la mémoire restante ne peut plus être attribuée à la suite du programme. Il est vrai qu'une table de descripteurs locale de 8192 segments peut être rajoutée mais la plupart des DOS Extenders ne misent que sur la seule table des descripteurs globale.

La fonction malloc() va donc être obligée de compacter plusieurs zones de mémoire en un seul segment qui croît à chaque appel. Dès qu'il est plein ou qu'un bloc requis n'y rentre plus, un nouveau segment est créé par addition d'un nouveau descripteur dans la table des descripteurs globale. Plusieurs objets mémoire se partagent donc le même sélecteur avec des offsets différents. Les vrais problèmes ne se posent qu'à partir du moment où un programme demande plus de 64 Ko en une fois, mais cette requête est déjà en dehors du champ de malloc() en mode réel.

Malgré la restriction à des blocs de 64 Ko, l'allocation de mémoire par la nouvelle fonction malloc() est déjà en elle-même fascinante par l'impression de facilité qu'elle donne. La plupart des programmes C épuisent la fonction malloc() jusqu'à ce qu'elle ne puisse plus distribuer de mémoire. Avec le mode protégé, le processus d'épuisement

est beaucoup plus long, le programme est donc capable de traiter automatiquement beaucoup plus de données. Cette constatation justifie largement l'attrait de la transformation des programmes C ordinaires par un DOS Extender pour 80286.

S'il est impossible de gérer par le moyen décrit des blocs de plus de 64 Ko, le DOS Extender fournit d'autres fonctions à cet effet. Ces fonctions établissent plusieurs sélecteurs, un par bloc de 64 Ko. Les zones ainsi allouées ne peuvent pas être parcourues par des pointeurs obéissant à l'arithmétique ordinaire, elles nécessitent des fonctions qui en cas de débordement de l'offset référencent le sélecteur suivant ou précédent.

Les possibilités étendues du 80386

Nous avons signalé plus haut que les programmes réalisés avec un DOS Extender pour 80386 étaient plus rapides en mode protégé que leurs homologues réduits à l'usage d'un 80286. La section suivante en explique les raisons.

33.3.3. DOS Extenders pour 80386

Les DOS Extenders pour 80386 donnent de meilleurs résultats que les DOS Extenders pour 80286, nous l'avons déjà dit à plusieurs reprises. Il existe plusieurs raisons à l'accroissement de prerformances mais elles tiennent toutes à un nombre magique: 32.

Le 80386 est un processeur à 32 bits ce qui s'exprime par plusieurs facettes très importantes pour les langages évolués:

- ✓ les segments peuvent avoir jusqu'à 4 Go. On peut donc se passer de segments de code et de données multiples
- ✓ les entiers 32 bits peuvent être mémorisés et traités dans un registre unique
- ✓ les instructions sur chaînes de caractères qui servent à copier, parcourir ou traiter des zones de mémoire peuvent travailler sur la base de mots doubles DWORD ce qui se traduit par une vitesse acrrue

Il existe encore d'autres raisons par lesquelles le 80386 manifeste sa supériorité sur le 80286:

- ✓ le mécanisme de pagination donne au DOS Extender la possibilité de munir le programme en mode protégé d'une gestion de mémoire virtuelle complètement transparente. Les limitations de mémoire s'en trouve supprimées.
- ✓ les possibilités d'adressage ont été démultipliées. Chaque registre peut servir de registre d'index et être multiplié automatiquement par 2, 4 ou 8. L'accès aux

tableaux s'en trouve considérablement accéléré. Le compilateur peut désormais stocker davantage de variables locales en permanence dans des registres.

- ✓ le jeu d'instructions a été étendu par l'adjonction de commandes puissantes pour le traitement des tables binaires. Il en résulte une amélioration notable de performance.

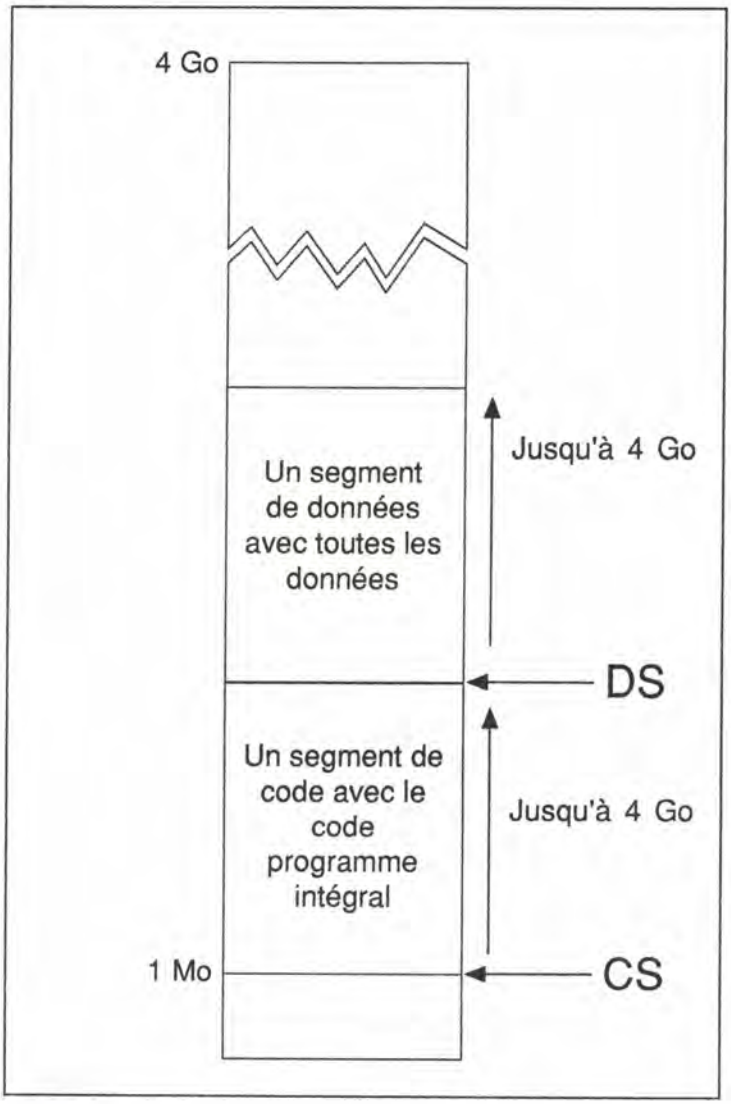
- ✓ les branchements conditionnels peuvent porter sur de grandes distances. Jusqu'ici pour atteindre le même résultat il fallait combiner plusieurs instructions qui alourdissaient et ralentissaient les programmes.

Il est vrai que toutes ces possibilités ne sont pas exploitées par les compilateurs DOS ordinaires qui souvent négligent même les instructions introduites par le 80286. Sans parler des difficultés de portabilité dans le domaine des pointeurs, les offsets ont maintenant 32 bits et non plus 16 ce qui représente un problème parmi d'autres.

Le modèle Flat

Aucun compilateur DOS ne supporte le modèle mémoire qui est celui des véritables compilateurs 386 et accroît la puissance d'exécution des programmes en mode protégé: le modèle plat (Flat model).

Ce modèle vaut pour les programmes qui ne comportent que deux segments: un segment de code avec l'ensemble des instructions et un segment de données avec toutes les variables et constantes. Le monde DOS connaît bien un modèle de ce type: il s'appelle TINY mais la taille de ses segments est limitée à 64 Ko. Le modèle plat du 80386 permet d'utiliser deux segments s'étendant conjointement sur 4 Go, ce qui est propre à éteindre toute envie de répartir le code et les données sur plusieurs segments.



Programme en modèle Flat

Avec ce modèle on obtient une vitesse d'exécution accrue parce que malgré la taille impressionnante des segments, les pointeurs sont en fin de compte de type NEAR, autrement dit on manipule uniquement un offset sans sélecteur. Ce dernier se trouve en effet en DS pour les données et en CS pour le code. Le temps des distinctions entre pointeurs NEAR et FAR est donc révolu.

Même si les offsets s'étendent non plus sur 16 bits mais sur 32 bits, l'avantage précédent n'est pas remis en cause. Par ailleurs dans le modèle plat les perpétuels chargements des registres de segments qui grevaient le mode protégé par rapport au mode réel n'ont

plus de raison d'être. Les problèmes suscités par l'arithmétique des pointeurs disparaissent: les pointeurs peuvent désormais être incrémentés et décrémentés comme des nombres ordinaires (en tant queforme de doubles mots DWORDs) sans qu'il faille craindre des modifications du sélecteur.

Mais ce n'est pas seulement à cause de la vitesse que le modèle plat a été notamment retenu pour la version 2.0 d'OS/2. Les DOS Extenders pour 80386 sont souvent utilisés pour convertir de gros programmes issus du monde Unix, par exemple des programmes à vocation mathématique ou statistique. Sous Unix la segmentation de la mémoire est une chose inconnue: c'est pourquoi la mise en service d'un modèle mémoire semblable apporte une simplification appréciable.

Les compilateurs sont d'ailleurs eux-mêmes tirés du monde Unix grâce à un Dos Extender. Sous Unix il existe en effet des compilateurs dont les facultés d'optimisation dépassent nettement celles qui se trouvent sous DOS et qui d'ailleurs ne se donnent pas toujours la peine de fonctionner. (Ainsi la version C du compilateur 5.1 de Microsoft a tendance à effectuer certaines prétendues optimisations en éliminant des lignes de code essentielles).

Les compilateurs DOS ont en fait des oeillères lorsqu'ils entreprennent leur optimisation: ils se contentent de prendre en considération à l'intérieur d'une fonction étroitement localisée une petite séquence d'instructions machine et ils essayent ensuite de mettre à la place des instructions plus rapides ou moins nombreuses. Les compilateurs Unix sont capables d'optimiser globalement un programme, ce qui demande une compréhension approfondie de sa structure. Ils doivent savoir quelles sont les fonctions appelées, à quel endroit elles se situent et quelles sont les variables exploitées. Ces renseignements ne sont pas seulement difficiles à obtenir, alourdissant la taille des compilateurs, ils amènent aussi en mémoire des quantités de données très importantes (le compilateur est très gourmand en mémoire).

Les compilateurs 836 les plus connus s'appellent High C386 (de Metaware) et Watcom C/386 (de Watcom). Le DOS Extender le plus fameux pour 80386 est le 386 | DOS Extender de Phar Lap qui est capable de fonctionner avec les deux compilateurs précédents. Son mode de fonctionnement rentre tout à fait dans le cadre des techniques décrites précédemment à propos des DOS Extenders pour 80286. Mais pour ce qui est des DOS Extenders dédiés au 80386, les appels à DOS et au BIOS ne sont pas toujours effectués en mode réel mais en mode V86, ce dernier comportant quelques avantages.

Appel des fonctions DOS

Un changement décisif a affecté l'émulation des fonctions de DOS. Dans un programme en mode protégé, les buffers dont l'adresse est à transmettre à certaines fonctions DOS se trouvent souvent au-delà des 64 premiers Ko du segment des données. Ces buffers ne sont pas à la portée de l'offset de 16 bits attendu par les fonctions DOS. Les fonctions

DOS émulées exploitent donc des offsets de 32 bits, les 16 bits supérieurs étant fournis par la partie étendue du registre correspondant. Voici un exemple pour bien comprendre ce point.

La fonction 09h DOS sert à émettre une chaîne de caractères sur le périphérique de sortie standard. Elle attend à cet effet dans les registres DS:DX l'adresse de la chaîne. La fonction DOS émulée exploite quant à elle les registres DS:EDX pour pouvoir traiter également un offset situé au-delà de la limite des 64 Ko. Le contenu du buffer indiqué est copié par le DOS Extender dans une zone temporaire situé en-dessous des 640 Ko. A cet endroit l'offset est à nouveau inférieur à 64 Ko et la fonction DOS originale peut à nouveau être appelée avec l'adresse du buffer en DS:DX.

Les registres étendus n'entrent pas seulement en service pour transmettre des adresses de buffers, ils servent aussi à mémoriser certaines indications de longueur. En font notamment usage les différentes fonctions de fichier lorsqu'il s'agit de manier le nombre de caractères à lire ou à écrire. Au lieu du registre CX c'est ECX qui va mémoriser le nombre d'octets à traiter. Mais les fonctions originales de DOS ne peuvent traiter que 64 Ko à la fois, le DOS Extender répète donc plusieurs fois des appels sur des blocs de 64 Ko jusqu'à ce que le nombre de caractères souhaité soit atteint. Ceci n'est qu'un exemple parmi toutes sortes de fonctions qui jusqu'ici agissaient avec des indications de longueur sur 16 bits et qui désormais grâce au DOS Extender peuvent travailler sur des nombres de 32 bits.

Intégration d'une gestion de mémoire virtuelle

Mais revenons au DOS Extender de Phar Lap cité précédemment. Il peut être mis en service avec un autre produit de la maison PharLap appelé 386 | VMM. Il s'agit d'un gestionnaire de mémoire virtuelle qui met à la disposition d'un programme protégé un espace de mémoire pratiquement illimité. Toutes les barrières imposées par la mémoire disparaissent bien que les échanges de mémoire consomment du temps et diminuent l'efficacité du programme. Cette gestion ne se justifie vraiment que pour des programmes qui nécessitent plus de mémoire qu'il n'en reste à la disposition des utilisateurs dans les ordinateurs sur lesquels ils travaillent.

Citons à titre d'exemple le logiciel Interleaf Pusbisher d'IBM. Il s'agit d'un logiciel de PAO issu du monde Unix. Avant d'être soumis à la gestion de mémoire virtuelle de PharLap, il exigeait au moins 6 Mo de méomire. Il se contente à présent de 2 Mo.

Il est préférable de planifier l'intervention de la mémoire virtuelle dès le stade de l'analyse du programme. Si vous allouez un tableau de 8 Mo et si vous le parcourez à plusieurs reprises, vous ne devrez pas vous étonner de voir clignoter tout le temps la diode du disque dur et il ne faudra pas vous plaindre de la lenteur de l'exécution.

Pour le programmeur habitué à DOS, travailler avec des compilateurs 386 réclame une certaine reconversion. Par exemple les variables entières de type INT sont maintenant

stokées sur 32 bits. Pour continuer à traiter des entiers de 16 bits, il faudra recourir au type SHORT.

De nombreuses fonctions se voient attribuer de nouvelles capacités. Ainsi malloc() peut désormais allouer d'un seul coup un bloc pouvant aller jusqu'à 4 Go. Les autres fonctions qui opèrent sur des blocs de données (notamment les fonctions de traitement des fichiers) bénéficient des mêmes améliorations.

Les bibliothèques des compilateurs demandent un certain apprentissage car elles émanent du monde Unix. Mais il est vrai qu'elles se recoupent largement avec les fonctions de DOS de sorte qu'on reste quand même en territoire connu.

Mais finalement le travail en vaut la peine. Les différents logiciels commerciaux qui existent à la fois en version mode réel et en version "386" le démontrent amplement car les versions "386" sont généralement développées avec un DOS Extender. Elles sont à la fois plus rapides et plus puissantes par leurs capacités. En conclusion, les DOS Extenders sont des instruments fort utiles pour développer des logiciels de grande envergure exploitant de nombreux calculs.

33.4. DPMI et VCPI

On dit parfois que la multiplication des cuisiniers n'améliore pas pour autant la soupe, et cette sentence pourrait très bien s'appliquer au mode protégé. Les conflits de ressources opposent des cuisiniers dont aucun n'accepte facilement de déposer sa louche.

Les utilitaires en mode protégé convoitent avant tout deux types de ressources: la mémoire étendue et le mode protégé lui-même. Chaque utilitaire s'imagine volontiers que la mémoire étendue lui appartient entièrement et que le mode d'exploitation du processeur est de son seul ressort. Qui pourrait d'ailleurs dire le contraire puisqu'il n'existe pas d'instance centrale qui gèrerait ces ressources. Après tout nous nous trouvons encore dans le monde DOS qui du point de vue du système d'exploitation ignore les termes de "mémoire étendue" et de "mode protégé"

C'est pourquoi les différents cuisiniers ne peuvent pas vraiment coexister. Les émulateurs d'EMS empêchent tout démarrage d'un programme développé avec un DOS Extender ou désireux de passer au mode virtuel 86 pour se lancer dans le multitâche. Après tout les choses sont très bien ainsi car sinon l'étape suivante consisterait à partager égalitairement la mémoire étendue, ce qui enlèverait beaucoup d'intérêt au mode protégé.

Si ces problèmes peuvent intéresser le programmeur système et lui donner matière à réflexion, l'utilisateur n'en a généralement cure. Il a investi son argent dans la version "386" d'un logiciel coûteux. Le voilà cependant qui constate que son programme ne fonctionne pas avec son utilitaire de gestion de la mémoire habituel. Ce gestionnaire

est pourtant indipensable car la mémoire conventionnele est accaparée par une infinité de drivers de périphériques. Et en plus le même programme refuse de fonctionner en multitâche alors qu'il serait si pratique de visualiser en même temps les données d'une feuille de calcul. Adieu le rêve de la station de travail sous DOS.

Mais la scène n'est pas aussi désespérée que cela car depuis les premiers jours où apparurent les utilitaires en mode protégé la situation a beaucoup évolué. Les éditeurs de logiciels se sont assis à une table et ont défini des interfaces logicielles pour assurer la coexistence pacifique des utilitaires en mode protégé. Il en a résulté deux standards: le plus ancien est appelé VCPI (Virtual Control Programming Interface) et il est surtout exploité par les DOS Extenders et les gestionnaires de mémoire. Le plus récent s'appelle DPMI (Dos Protected Mode Interface), c'est sur lui que mise Microsoft dans Windows 3.

Si on compare les deux standards il s'avère incontestablement que DMPI est meilleur. Il est vrai qu'il est trop jeune pour s'imposer à travers les DOS Extenders et les traditionnels utilitaires en mode protégé. Mais par rapport au bricolage de VCPI, le standard VPMI est bien plus professionnel.

Les principes des deux interfaces vont être exposés ci-dessous mais ne vous attendez à une exploration détaillée de toutes les notions et fonctions en jeu. Une telle étude dépasserait le cadre du présent ouvrage. Mon intention est plutôt de vous montrer la complexité des mécanismes qui ont dû être inventés pour maintenir artificiellement DOS en vie.

33.4.1. VCPI

L'interface VCPI a été présentée en 1989 par un consortium de sociétés éditrices de logiciels sous la conduite de PharLap (développeur de DOS Extenders) et de Quarter-Deck (auteur de DESQView et QEMM). Elle se préoccupe essentiellement des problèmes posés par la coexistence de DOS Extenders, de gestionnaires d'exploitation multitâche et de gestionnaires de mémoire lorsqu'ils interviennent en même temps sur un système à base de 80386 ou i486. Le 80286 ne joue aucun rôle dans ces spécifications.

Les premières difficultés commencent avec l'installation d'un gestionnaire de mémoire qui dote l'ordinateur d'une mémoire virtuelle EMS. Pour exploiter les possibilités de pagination du processeur, la machine DOS est commutée en mode V86 ce qui la relègue au niveau de privilège 3. Si à partir de ce niveau on lance un gestionnaire multitâche ou une application créée par un DOS Extender, le processeur ne peut pas passer en mode protégé et il est impossible d'installer une quelconque table des descripteurs. Les mécanismes de protection du processeur s'y opposent.

Mais VCPI offre justement à ces programmes un moyen de commutation en mode protégé tout en prenant des dispositions pour leur assurer une coexistence harmonieuse en mémoire étendue. Presque tous les gestionnaires de mémoire connus (QEMM, 386-to-the-Max, etc...) supportent VCPI tout comme le DOS Extender de PharLap et l'environnement multitâche de DESQView/386 de QuarterDeck. Même DOS dans sa

version 5.0 se plie à VCPI lorsqu'il fournit le driver EMM386 qui gère l'accès aux blocs de mémoire supérieure.

Lorsqu'on est un programmeur qui développe une application avec un DOS Extender ou qui réalise des logiciels fonctionnant avec DesQView, on n'entre pas en contact avec VCPI. Ce sont le DOS Extender et l'API de DESQView qui s'en chargent. L'interface VCPI ne s'impose que pour les programmes qui voudraient exploiter le mode protégé sans ces utilitaires.

Client et serveur

Vous connaissez peut-être déjà les notions de client et de serveur pour avoir côtoyé des réseaux. Le modèle client-serveur y est en effet très utilisé. Sous VCPI ces notions ont une signification quelque peu différente malgré certaines analogies.

On appel serveur l'application qui offre des fonctions VCPI à différents clients. Sous VCPI le serveur est toujours constitué par un gestionnaire de mémoire (par ex EMM836.EXE) car ce type de programme est installé dès le lancement du système par l'intermédiaire du fichier CONFIG.SYS. Il est donc déjà présent au premier appel d'un DOS Extender ou d'un environnement multitâche.

Les DOS Extenders et les gestionnaires multitâche deviennent ainsi clients du programme VCPI préinstallé et sont en quelque sorte hébergés sous le même toit. Il est vrai qu'un DOS Extender ou un gestionnaire multitâche pourraient très bien être des serveurs mais il se trouve que seuls les gestionnaires de mémoire sont automatiquement présents du démarrage du système jusqu'à son extinction et qu'ils constituent donc le meilleur choix. Les autres utilitaires en mode protégé n'ont pas cet avantage. Ainsi le premier arrivé prend en charge l'ensemble des opérations, c'est donc le gestionnaire de mémoire qui va jouer au chef d'orchestre.

Les services offerts

Le serveur VCPI offre à ses clients 13 fonctions qui couvrent tous les domaines où il existe un risque de collision entre les utilitaires en mode protégé:

- ✓ 3 fonctions sont consacrées à l'initialisation de l'interface VCPI
- ✓ 4 fonctions proposent une gestion rudimentaire de la mémoire étendue
- ✓ 3 fonctions permettent d'accéder au premier registre de contrôle CR0 et aux registres de mise au point du processeur
- ✓ 2 fonctions sont une aide à la programmation du contrôleur d'interruption
- ✓ 1 fonction est responsable de la commutation entre mode V86 et mode protégé

Ces fonctions sont implémentées comme une extension du gestionnaire de mémoire EMS car tout gestionnaire de mémoire doit évidemment supporter la mémoire EMS avec son interface. La base de départ est constituée par la version 4.0 du standard LIM. Les fonctions s'appellent par l'interruption EMS 67h et portent toutes le numéro DEh (ce qui évoque très curieusement le nom de DESQView). La fonction VCPI choisie est définie par un code de fonction à placer en AL, et qui doit être compris entre 00h et 0Ch.

Le résultat retourné au client est mémorisé en AH, comme c'est le cas habituellement pour les fonctions EMS. La valeur 0 signifie "OK", toute autre valeur signale une erreur d'exécution de la fonction appelée.

Il n'est possible de déclencher l'interruption 67h que dans le mode réel ou le mode V86. En mode protégé le processeur n'exploite pas la table des vecteurs d'interruption dans laquelle se trouve le pointeur qui référence le gestionnaire EMS. En mode V86 la communication entre client et serveur doit être introduite par différents appels de fonctions qui assurent au client un accès au serveur malgré le mode protégé.

Initialisation d'une liaison client-serveur

Les fonctions VCPI ne peuvent naturellement être exploitées que s'il existe un serveur VCPI. Il en est d'ailleurs de même pour les fonctions EMS ordinaires. Un client potentiel doit donc commencer par tester la présence d'un serveur VCPI ce qui veut d'abord dire: s'assurer qu'un driver EMS est disponible.

Une page EMS doit ensuite être requise à l'aide des fonctions EMS ordinaires. Pourquoi ? Parce que certains émulateurs de mémoire paginée restent latents en mode réel tant qu'aucune mémoire EMS n'est demandée. C'était le cas jusqu'à la version 5.0 de DOS de l'émulateur EMMM386 de Microsoft. Ce n'est qu'après commutation en mode V86 que de nombreux émulateurs d'EMS se font reconnaître comme serveurs VCPI. Il faut donc maintenir l'allocation de la page jusqu'à la fin du programme pour éviter le retour au mode réel et l'abandon des fonctionnalités VCPI.

Mais même si une page EMS a été allouée, il reste encore à s'assurer que l'émulateur d'EMS supporte l'interface VCPI. Il suffit pour cela d'appeler la fonction VCPI 00h avec la valeur DE00h en AX. Si seule l'interface EMS ordinaire est présente, le driver EMS retourne le code erreur 84h en AH. Un serveur VCPI répondra par contre par le code 00h. Vous trouverez alors en BX le numéro de version de l'interface VCPI, sous forme binaire, chiffre majeur en BH et chiffre mineur en BL.

Il faut ensuite invoquer la fonction 01h, qui est relativement complexe et qu'on ne peut comprendre qu'en liaison avec la gestion de la mémoire en mode protégé avec un émulateur EMS.

Exploitation des tables de pages

Le serveur VCPI et le client qui s'exécute en mode protégé possèdent tous deux une table des descripteurs propre car ils contrôlent entièrement les registres GDTR, LDTR, IDTR et TR. Comme les tables de descripteurs ne contiennent que des adresses linéaires, et non pas des adresses physiques, aucun conflit n'est encore possible. Partager l'espace physique signifie cependant soit exploiter une structure de table de pages commune au serveur et au client (le registre CR3 restant inchangé), soit travailler au moins partiellement avec des tables de pages identiques.

L'interface VCPI utilise la deuxième possibilité. Le client VCPI doit initialiser sa collaboration par un appel à la fonction VCPI 01h qui, entre autres, remplit sa table de pages. Cet appel doit être effectué en mode réel avant toute commutation en mode protégé.

Lors de l'appel de la fonction, le client doit transmettre au serveur par l'intermédiaire des registres ES:DI l'adresse physique d'un bloc de mémoire de 4 Ko. Ce bloc servira ultérieurement à l'installation de la première table de pages qui reproduira les premiers 4 Mo de l'espace mémoire linéaire. Le répertoire des pages que doit référencer le premier élément de la table des pages est à constituer par le client avant sa commutation en mode protégé.

La future table des pages est initialisée par le serveur de telle sorte que les 256 premiers éléments qui reflètent le premier mégaoctet de l'espace linéaire dans l'espace physique créent une relation de 1 à 1 entre les deux espaces. En clair cela veut dire que le client peut accéder au premier mégaoctet de son espace physique par l'intermédiaire du premier mégaoctet de l'espace linéaire, ce qui ouvre la voie au BIOS, à la mémoire écran et à la mémoire conventionnelle de DOS. Le serveur VCPI procède exactement de même: la zone en question peut donc être exploitée pour échanger des données entre client et serveur.

Mais avec la table des pages en service le serveur VCPI peut installer encore davantage de mémoire linéaire en mémoire physique. La fonction VCPI 01h renvoie à cet effet dans le registre DI l'offset référençant le premier élément libre de la table des pages. C'est à partir de cette adresse que le client pourra installer ultérieurement ses propres éléments dans la table.

En soustrayant l'offset d'origine de la valeur renvoyée en DI, on obtient la longueur courante de la table des pages. Comme chaque élément en occupe 4 octets, une division par 4 donne le nombre de pages référencées et l'indice de la prochaine page libre. De cet indice on peut déduire la première adresse libre dans l'espace linéaire: il suffit de le multiplier par 4 Ko. Cette information est importante lorsqu'on alloue de la mémoire, comme le montrera la section suivante.

La fonction 01h attend aussi en DS:SI un argument constitué par un pointeur. A partir de l'emplacement référencé, le serveur stocke trois descripteurs de segments occupant

24 octets au total. Le client devra plus tard incorporer ces descripteurs dans sa table globale. Ils pourront être recopiés à n'importe quel endroit de la GDT, à moins que l'on ne préfère englober dans la GDT la zone pointée par DS:SI. Il faudra cependant mémoriser la situation de ces descripteurs au sein de la GDT c'est-à-dire leur indice car ultérieurement un sélecteur sur le premier élément devra être établi.

Ce premier élément décrit le segment de code par lequel le client peut atteindre les fonctions EMS du serveur en mode protégé. Comme il ne s'agit que d'un descripteur de segment et non d'une porte d'appel, l'appel du serveur nécessite en plus un offset. Ce dernier est renvoyé en BX par la fonction 01h.

L'appel du serveur devra en fin de compte se faire par un FAR CALL qui se réfère au sélecteur du segment de code du serveur et adopte comme offset la valeur retournée en BX.

La gestion de la mémoire dans l'interface VPCI

Un client VCPI peut généralement supposer que le serveur VCPI a pris en charge la totalité de la mémoire étendue disponible au moment de son lancement. S'il reste encore un peu de mémoire disponible -le plus souvent à la demande expresse de l'utilisateur- elle est presque toujours épuisée par une disquette virtuelle, un programme de cache ou quelque chose de semblable. Les appels de l'interruption 15h du BIOS ou de l'interface XMS pour exploiter de la mémoire étendue resteront habituellement insatisfaits. Le client VCPI en mode protégé ne peut réclamer de la mémoire que par le serveur VCPI, abstraction faite des petits résidus qu'un programme en mode réel (en fait le mode V86) peut encore tirer des fonctions de DOS.

La gestion de la mémoire VPCI est fondée sur des pages de 4 Ko gérées par le mécanisme de pagination des processeurs 80386 et au-delà. La mémoire ne peut ainsi être allouée et libérée que par meultiples de 4 Ko. Les quatre fonctions VCPI qui interviennent peuvent être appelées à la fois en mode protégé et en mode V86, ce qui n'est pas le cas de toutes les fonctions VCPI.

La fonction 02h indique l'adresse physique de la plus haute page que le serveur VCPI puisse attribuer. Avant de l'appeler, il faut charger la valeur DE02h dans le registre AX. Si à l'issue de l'appel le même registre contient le nombre 0, c'est que la fonction s'est exécutée correctement. Le registre EDX donne alors l'information recherchée.

Cette information est cependant à manier avec précaution. Certains serveurs VCPI retournent l'adresse de la dernière page théoriquement possible et non celle de la dernière page physiquement disponible. La fonction 02h renvoie alors systématiquement la valeur FFF000h (dernière page avant la limite de 16 Mo) et non pas quelque chose comme 3FF000h si l'ordinateur est équipé de 4 Mo.

Il est donc prudent de renoncer à la mise en service de la fonction 02h et de se rabattre sur la fonction 03h qui indique en EDX le nombre de pages disponibles. C'est là le maximum qui puisse ensuite être réclamé à la fonction 04h en une seule fois. A chaque appel la fonction 04h retourne l'adresse physique de la page demandée mais uniquement si le registre AH est à 00h. Toute autre valeur en AH dénote une erreur d'allocation, le résultat en EDX étant alors dénué de signification.

Pour adresser cette mémoire qui peut provenir de zones diverses, le client VCPI doit faire deux choses:

① Il doit créer un descripteur de segment dans sa table de descripteurs globale ou locale pour pouvoir accéder la mémoire allouée par un segment. Le segment en question peut dépasser 4 Ko lorsque plusieurs pages sont rassemblées dans un segment qui les englobe.

② Il doit créer un élément dans la table des pages qui permette de refléter l'espace linéaire de 4 Ko dans l'espace de la mémoire physique. L'adresse linéaire qui est contenue dans le descripteur de segment mentionné plus haut décide de la situation de l'élément dans la table des pages. Comme une table des pages a déjà été installée pour les quatre premiers mégaoctets (elle doit être transmise à la fonction 01h), il est recommandé de choisir une adresse linéaire à l'intérieur de de ces 4 Mo. Il ne faut pas oublier à ce propos que le serveur VCPI à l'appel de la fonction 01h a déjà créé quelques entrées dans la table des pages au moins pour installer le premier mégaoctet de l'espace linéaire. La première adresse linéaire libre et l'indice dans la table des pages correspondante peuvent être calculés à l'aide la formule indiquée précédemment.

Si le client VCPI n'a plus besoin des pages allouées, il peut les restituer avec la fonction 05h, mais il n'a plus le droit d'y accéder par la suite même si sa table des pages en garde la possibilité. En plus de son numéro la fonction attend comme argument en EDX l'adresse physique de la page.

VCPI et mémoire EMS

Le client VCPI peut non seulement réclamer de la mémoire par l'interface VCPI mais aussi par les fonctions EMS ordinaires, par blocs de 16 Ko. L'accès à ces pages EMS se fait par le cadre de page dont l'emplacement se situe en dessous de la limite de 1 Mo, comme c'est la règle pour la mémoire paginée.

Pour connaître l'origine des ces pages, on peut mettre en service la fonction VCPI 06h. Pour chaque zone de mémoire située en-deçà de 1 Mo, elle donne la situation de la page associée dans l'espace physique. On peut ainsi détecter des zones de mémoire qui n'existent pas directement sous forme de RAM mais que le gestionnaire de mémoire a associé à de la mémoire réelle par la table des pages. Ce mécanisme s'appelle le "backfilling". DOS 5.0 l'exploite pour caser ses blocs de mémoire supérieure.

Lorsqu'on appelle la fonction 06h, il faut lui fournir en CX le numéro de la page testée en-dessous de 1 Mo. On l'obtient en divisant l'adresse linéaire par 4096, ce qui revient à décaler l'adresse de 12 bits vers la droite. Le résultat renvoyé en EDX est l'adresse physique de la page mémoire représentée à cet endroit.

Si le contenu de EDX est égal à CX * 4096, aucune mémoire étrangère n'est insérée à l'adresse indiquée. L'adresse linéaire de ces pages de 4 Ko est alors identique à leur adresse physique.

En fait tout l'espace de 1 Mo ne peut pas être interrogé par ce moyen. les zones dans lesquelles se trouvent le BIOS ou ses extensions sont exclues. Dans ce cas la fonction 06h renvoie en AH le code erreur 8Bh alors que normalement il doit s'y trouver la valeur 00h

Accès aux registres de mise au point et d'état

L'accès aux différents registres de mise au point n'est possible que sous le contrôle étroit du serveur VCPI, du moins avec le mode V86 qui fonctionne au niveau de privilège 3.

Grâce aux fonctions VCPI 08h et 09h, les différents registres de mise au point peuvent être lus et chargés à partir du mode V86. Les programmes habituels ne se servent pas de ces registres mais ils valent leur pesant d'or pour un logiciel du genre Turbo Debugger de Borland qui exploite également l'interface VCPI.

Les deux fonctions opèrent simultanément sur les huit registres de mise au point. Les contenus des registres sont transmis par un buffer référencé par ES:DI. Comme chaque registre de mise au point s'étend sur 32 bits, ce buffer doit avoir une capacité de 32 octets. DR0 est mémorisé dans le premier double mot du buffer, DR1 passe dans le second, et ainsi de suite. La fonction 08h copie les registres dans le buffer alors que la fonction 09 h fait exactement l'inverse.

Si ces fonctions peuvent être très utiles, la fonction 07h par contre est un anachronisme. Elle renvoie le contenu du registre d'état en EBX. Ce registre peut effectivement indiquer le mode d'exploitation courant et l'état du mécanisme de pagination mais les mêmes informations sont fournies par l'instruction non privilégiée SMSW qui recopie le contenu du registre en question dans un autre registre du processeur ou dans une mémoire.

Interception et fixation des vecteurs d'interruption

Lorsque nous avons parlé des exceptions en mode V86 nous avons déjà indiqué qu'il était recommandé de reprogrammer le premier contrôleur d'interruption pour éviter que des exceptions ne se mélangent aux interruptions matérielles IRQ0 à IRQ7.

Si un client VCPI souhaite ainsi reprogrammer le premier contrôleur d'interruption, il devra commencer par lire son contenu actuel. Il peut se servir à cet effet de la fonction VCPI 0Ah qui renvoie en BX l'interruption de base du premier contrôleur et en CX l'interruption de base du second contrôleur. Si les deux contrôleurs n'ont plus leurs valeurs par défaut 08H et 70h c'est que le serveur VCPI a déjà détourné ces interruptions. Tout nouveau détournement est alors strictement prohibé.

Mais si tel n'est pas le cas, le client VCPI peut reprogrammer le premier contrôleur d'interruption et même le deuxième s'il le juge utile. L'accès aux contrôleurs devra se faire manuellement après inhibition de toutes les interruptions. Avant que l'indicateur d'interruption ne soit à nouveau mis à 1, le client doit appeler la fonction VCPI 0Bh pour informer le serveur de la reprogrammation du contrôleur. De façon analogue à ce qui se passe avec la fonction 0Ah mais en sens inverse, l'adresse de base du premier contrôleur devra être mise en BX et celle du deuxième en CX.

Changement de mode d'exploitation

L'une des tâches les plus importantes du serveur VCPI consiste à passer du mode V86 au mode protégé et vice-vesra. C'est la fonction VCPI 0Ch qui en est responsable. En mode V86 elle est appelée par une interruption EMS tout à fait ordinaire. En mode protégé elle est accessible par le segment de code et l'offset renvoyés par le serveur lors de l'appel de la fonction VCPI 01h.

Lorsque cette fonction est appelée à partir du mode V86, elle attend en ESI l'adresse linéaire d'une structure de données dont l'organisation est décrite par le tableau suivant. C'est de là qu'elle tire les valeurs qui seront chargées dans les différents registres système:

Structure de données pour passer du mode V86 au mode protégé en appelant la fonction VCPI 0Ch

Offset	Signification	Type
00h	CR3 (Adresse où commence le répertoire des pages)	DWORD
04h	Adresse linéaire de la variable en dessous de 1 Mo qui contient le pointeur FAR (FWORD) pour le registre GDTR	DWORD
08h	Adresse linéaire de la variable en dessous de 1 Mo qui contient le pointeur FAR (FWORD) pour le registre IDTR	DWORD
0Ch	Sélecteur pour le registre LDTR	WORD
0Eh	Sélecteur pour le registre TR	WORD
10h	Point d'entrée dans le code de programme commutation en mode protégé avec sélecteur et offset de 32 bits	FWORD

Il est très important que cette structure soit installée en dessous de la limite de 1 Mo car ce n'est que dans cette zone que les espaces d'adressage du serveur VCPI et de son client sont identiques, comme nous l'avons remarqué à l'examen de la fonction 01h.

La première étape de la fonction consiste à charger dans le registre CR3 la valeur prévue dans la structure de données. A partir de ce moment ce ne sont plus les tables de pages du serveur mais celles du client qui sont valables. Le serveur ne peut plus lire que des informations dans les zones qui se recouvrent c'est-à-dire en dessous de 1 Mo.

Le registre GDTR est ensuite chargé. La table de descripteurs globale ainsi référencée peut se trouver n'importe où dans l'espace d'adressage du client VCPI. Elle doit évidemment avoir été initialisée auparavant en mode V86 mais la supposition en est déjà faite lors de l'appel de la fonction VCPI 01h.

C'est ensuite au tour des registres LDT, IDTR et TR d'être chargés. Puis la commutation en mode protégé est effectuée et le programme se branche à l'endroit indiqué. A ce moment toutes les interruptions sont encore inhibées et elle doivent encore le rester car des six registres de segment seul le registre CS contient déjà un sélecteur valable. Il est donc urgent de créer une pile appropriée et de mettre des sélecteurs valables dans les registres DS, ES, FS et GS. Ensuite les interruptions peuvent à nouveau être autorisées.

Si avant le chargement des registres de segment une interruption survenait, la sauvegarde et la restauration des registres à l'intérieur du gestionnaire d'interruption (PUSHA, PUSH DS, PUSH ES ...POP ES, POP DS, POPA) conduirait par la suite à la prise en compte de sélecteurs invalides, ce qui finirait par déclencher une exception au niveau du processeur.

Le retour au mode V86 se fera par la fonction 0Ch mais avec d'autres contenus de registres. Le point d'entrée sera indiqué par le sélecteur de segment de code et l'offset retournés par la fonction 01h.

Avant d'appeler la fonction 0Ch le client VCPI devra mémoriser une structure de données dans la RAM physique située en dessous de 1 Mo. Cette structure contiendra les valeurs que le serveur VCPI doit charger en CS,DS, ES,FS, GS, SS, EIP et ESP après la commutation en moe V86. L'indication de CS:EIP fixe l'adresse où l'exécution du programme doit reprendre en mode V86. Les registres SS:ESP décrivent le nouvel emplacement de la pile.

Notez bien que les valeurs pour les différents registres ne sont pas des sélecteurs mais des adresses de base (adresse physique divisée par 16)., comme le réclament le mode réel et le mode V86. Si le programme en mode V86 n'a pas été développé spécialement pour le 386 et ses successeurs, les registres FSet GS peuvent recevoir un contenu quelconque car ils ne jouent aucun rôle dans l'exécution du programme.

Même si les registres ne prennent qu'un mot, la structure de données mentionée plus haut en prévoit deux pour chacun. Le premier mot va mémoriser une adresse de segment tandis que le second qui lui succède reste inutilisé.

Structure de données pour passer du mode protégé au mode V86 en appelant la fonction VCPI 0Ch		
Offset	**Signification**	**Type**
00h	réservé	QWORD
08h	EIP après commutation en mode V86	DWORD
0Ch	CS après commutation en mode V86	DWORD
10h	réservé (registre EFLAGS)	DWORD
14h	ESP après commutation en mode V86	DWORD
18h	SS après commutation en mode V86	DWORD
1Ch	ES après commutation en mode V86	DWORD
20h	DS après commutation en mode V86	DWORD
24h	FS après commutation en mode V86	DWORD
28h	GS après commutation en mode V86	DWORD

Pour que le serveur VCPI puisse accéder à la table précédente après l'appel de la fonction 0Ch, il faut transmettre son adresse linéaire dans les registres SS:ESP. Par ailleurs le client doit préalablement à l'appel charger en DS un sélecteur référençant un descripteur de segment de données qu'il a lui-même installé dans sa table de descripteurs globale ou locale. Ce descripteur doit contenir une adresse de base nulle et représenter une taille de 1 Mo.

Après commutation en mode V86, le contenu des registres standard (EBX, ECX etc...) est préservé. Seul le contenu de EAX se trouve modifié dans un sens indéfini. Les registres du mode protégé GDTR, LDTR etc... ont été rechargés par le serveur qui y a remis les adresses ou les sélecteurs de ses tables de descripteurs. Mais après tout ce n'est plus l'affaire du programme en mode V86.

33.4.2. DPMI

L'interface du mode protégé de DOS (DOS Protected Mode Interface = DMPI) n'était pas destiné a priori à fixer des règles standard de collaboration entre gestionnaires d'exploitation multitâche et autres utilitaires en mode protégé. Il s'agissait à l'origine d'un produit développé à titre interne par Microsoft pour les besoins de Windows 3. Le but poursuivi était de permettre l'exécution en mémoire étendue des applications Windows. Pour des raisons inconnues Microsoft décida de publier et d'étendre cette interface. Début 1990 un comité DMPI fut créé avec différents fabricants de logiciels. Cette association regroupe en plus de Microsoft les sociétés Borland, Intel, Eclipse, IBM, Lotus, Phar Lap, Quarterdeck et Rational Systems: on y reconnaît certains protagonistes de VCPI.

On peut en déduire que les DOS Extenders, gestionnaires multitâche et gestionnaires de mémoire proposés à l'avenir par ces participants supporteront non seulement VCPI mais aussi DMPI, à moins que VCPI ne passe carrément à la trappe. Les DOS Extenders de Phar Lap exploitent déjà le nouveau standard.

Différences avec VCPI

La force agissante qui détermine cet engagement trouve son origine dans le marché ouvert par la diffusion explosive de Windows. Par ailleurs les spécifications DPMI sont supérieures à celles de VCPI, à la fois du point de vue théorique et dans le domaine de la réalisation concrète. En deux mots, DPMI est plus "propre" et plus général que son concurrent.

L'interface DPMI couvre toute la gamme des services que nécessitent les programmes DOS en mode protégé pour vivre pacifiquement en compagnie d'autres programmes en mode protégé. Elle assure notamment:

- ✓ la gestion des tables de descripteurs d'un programme en mode protégé
- ✓ la gestion et l'allocation de la mémoire étendue
- ✓ la gestion des interruptions et exceptions
- ✓ la communication avec les programmes en mode réel et les gestionnaires d'interruption
- ✓ l'accès à différents registres du processeur
- ✓ la virtualisation DMA

La liste de ces services montre déjà que les clients DPMI doivent céder une grande partie de leurs droits de mode protégé à l'hôte DPMI. C'est la seule possibilité envisageable pour que les différents programmes ne se querellent pas. Le modèle de protection du processeur garantit alors qu'un client potentiel n'utilise pas ces services pour fouiller par exemple de sa propre initiative dans sa table des descripteurs locale.

Contrairement au serveur VCPI, l'hôte DPMI s'exécute à un niveau de privilège supérieur à celui de son client, ce qui lui confère le contrôle de tous les clients et de leurs activités. Microsoft a toujours reproché à l'interface VCPI de ne pas exploiter ce type de protection, et il est tout à fait vrai que sa présence dans VCPI aurait été un atout de première valeur.

Clients et hôte

Le comité DPMI ne parle jamais de "serveur", il utilise le terme "hôte". La signification est la même. Je suppose qu'on a voulu s'écarter du modèle client-serveur qui en informatique évoque autre chose.

Dans l'interface DPMI le programme hôte est celui qui met les services DMPI à la disposition des différents clients. Jusqu'à ce jour il n'existe qu'un seul hôte DPMI: Windows 3.0 ou 3.1 qui est conforme aux spécifications de la version 0.9 de DPMI. Entretemps le comité DPMI a publié la version 1.0 qui diffère sur quelques points de la version 0.9 mais nous en reparlerons plus tard.

DPMI 16 ou 32 bits

Contrairement à VCPI, DPMI est également conçu pour le processeur 80286 même si Windows 3 n'en tient pas compte. Les propriétés de l'hôte DPMI ne s'activent que dans le mode étendu de Windows qui ne s'exécute que sur les ordinateurs pourvus d'un 80386 ou d'un processeur postérieur et d'une quantité suffisante de mémoire étendue. Comme les systèmes à base de 80286 sont en perte de vitesse, il est peu probable qu'il y ait jamais un hôte DPMI pour 80286.

Mais à cause de l'existence du 80286, un hôte DPMI peut se présenter sous deux formes: 16 bits et 32 bits. Les hôtes DPMI 16 bits sont conçus pour le processeur 80286 et ne supportent que des segments de 64 Ko. Un hôte DPMI 32 bits fonctionne par contre avec des segments 32 bits qui peuvent atteindre 4 Go en modèle plat. Un tel hôte n'est exécutable que sur un ordinateur à base de 80386 ou i486.

Mis à part certains compléments qui concernent la gestion de la mémoire virtuelle avec les tables de pages, un hôte DPMI de 32 bits présente les mêmes fonctions que son homologue à 16 bits. La différence tient essentiellement au transfert des pointeurs, plus précisément au transfert des offsets. Quand un hôte de 16 bits attendra un offset en BX, son compère 32 bits l'attendra en EBX. Ce n'est qu'à ce prix que peuvent être communiqués les adresses des buffers dont l'offset se trouve au delà des 64 premiers Ko d'un segment.

Les clients qui sont destinés à collaborer avec des hôtes 16 bits peuvent ainsi tourner sous le contrôle d'hôtes 32 bits, pour peu que les 16 bits supérieurs des registres du 80386 ne soient pas altérés. Mais l'inverse n'est pas possible.

La plupart des DOS Extenders et des gestionnaires de mémoire n'exploiteront pas cette compatibilité ascendante, car il est trop tentant pour eux de ne fonctionner qu'avec des hôtes 32 bits. Pour savoir si en tant que client DPMI on a devant soi un hôte 16 ou 32 bits il suffit de tester l'indicateur d'état qui signale la disponibilité de l'hôte.

Destination du standard DPMI

Tout comme VCPI, l'interface DMPI ne s'adresse pas au programmeur d'applications ordinaires mais aux développeurs de DOS Extenders et de gestionnaires de mémoire. Ces derniers sont censés exploiter ses services pour que leurs programmes en mode

protégé s'exécutent également dans un environnement multitâche comme Window 3.0. Les premiers DOS Extenders qui font usage de cette possibilité sont déjà sur le marché. Mais un tel DOS Extender ou le programme qu'il génère doit aussi fonctionner comme hôte DPMI. Car si le programme est lancé à partir de DOS, sans que jusqu'ici aucun autre hôte DPMI n'ait été activé (tous les utilisateurs ne travaillent pas forcément sous Windows), il doit mettre ses services à sa propre disposition.

Exécution à l'intérieur d'une machine virtuelle

L'hôte DPMI exécute ses programmes clients dans une machine dite virtuelle incarnée sous Windows 3 par la boîte DOS. Plusieurs clients DPMI peuvent tourner à l'intérieur d'une même machine virtuelle (MV). tel sera par exemple le cas si le gestionnaire de mémoire exploite les services de l'hôte DPMI dans un boîte DOS de Windows et s'il est lancé au niveau de DOS à partir d'un programme d'application qui a été créé avec un DOS Extender. En tant que composants d'une même machine virtuelle, les différents programmes en mode réel se partagent un espace d'adressage qui reproduit une machine DOS de 1 Mo plus la HMA.

Entre les différents clients d'une même MV il n'existe pas d'exploitation multitâche mais elle est possible entre les différentes MV. Ce n'est qu'ainsi que Windows est capable d'exécuter simultanément plusieurs programmes de DOS dans plusieurs boîtes DOS. Le multitâche est de type préemptif: au bout d'un certain temps l'hôte DPMI interrompt l'exécution dans une MV pour passer à une autre exécution dans une autre MV.

La commutation se fait après expiration d'un certain délai, mais les clients peuvent apporter leur soutien à ce processus en indiquant par une fonction spéciale (Int 2Fh, fonction 1680h) qu'ils sont inoccupés, dans l'attente par exemple d'une saisie au clavier. L'hôte DPMI peut alors passer directement le contrôle à la MV suivante sans gaspiller du temps d'attente.

C'est dans la gestion de la machine virtuelle que réside la plus grande différence entre les versions 0.9 et 1.0 de DPMI. Dans la version 0.9, tous les clients d'une même VM se partagent une table de descripteurs locale (LDT) et une table des vecteurs d'interruption (IDT). Ce mécanisme n'est pas sans danger car les différents clients ont ainsi accès à leurs segments mutuels en exploitant leurs descripteurs. L'hôte DPMI ne peut rien faire contre. Par ailleurs ledit mécanisme rend impossible l'exécution simultanée de clients 16 bits et clients 32 bits parce que leurs descripteurs ne peuvent pas être rassemblés dans un même tableau.

La version 1.0 de DPMI affecte à chaque client d'une même MV ses propres LDT et IDT, ce qui met en difficulté Windows. Car beaucoup de programmes DOS en mode protégé qui jusqu'ici tournaient impeccablement dans la boîte DOS de Windows s'étaient accordés à exploiter la même LDT avec d'autres programmes: ils doivent être réécrits. C'est peut-être la raison pour laquelle l'hôte DPMI de Windows soutient comme précédemment l'ancienne version 0.9.

Fonctions de l'interface DPMI

L'interface DPMI prend en charge toute une série de tâches qui vont être brièvement présentées ici. Vous trouverez ensuite dans les sections suivantes l'étude individuelle des fonctions les plus importantes. Le point de départ sera la version DPMI 0.9.

Il existe 13 fonctions différentes qui se consacrent au traitement de la table des descripteurs locale du client et des descripteurs de segments qu'elle contient. Rappelons que dans l'interface VCPI le traitement de cette table des descripteurs était aux mains du client, ce qui ouvrait la porte à bien des problèmes. Ici il en est différemment mais il ne faut pas se cacher que les fonctions DMPI laissent encore quelques possibilités d'abus.

Sous DPMI chaque tâche ne peut accéder qu'à sa propre table de descripteurs locale, la table des descripteurs globale étant hors de portée. Cette dernière est réservée à l'hôte DPMI seul.

Pour ce qui est de la gestion de la mémoire, l'interface DPMI supporte trois groupes de fonctions différentes. Le premier s'occupe de l'allocation et de l'accès à la mémoire DOS conventionnelle en dessous de la limite de 1 Mo. Comme l'a expliqué le chapitre consacré aux DOS Extenders, cette mémoire est requise par les programmes en mode protégé lorsqu'ils font appel à des fonctions DOS qui attendent des informations dans des buffers.

Le deuxième groupe de fonctions de gestion de la mémoire se consacre à la mémoire étendue, la véritable patrie des programmes en mode protégé. Comme il faut s'y attendre, ces fonctions permettent d'allouer ou de libérer des blocs de mémoire, ou encore de modifier leur taille.

Le troisième groupe de fonctions sert à gérer la mémoire virtuelle. Celle-ci n'est pas forcément fournie par tous les hôtes DPMI, car elle ne peut pas fonctionner sur les systèmes à base de 80286 en raison de l'absence de pagination. Les fonctions concernées permettent notamment de geler une page pour qu'elle ne puisse pas être swappée hors de la mémoire.

Un autre groupe important s'occupe d'appeler les routines en mode réel et les gestionnaires d'interruption à partir du mode protégé. En invoquant les fonctions de DOS ou du BIOS, l'hôte DPMI décharge énormément son client. Mais inversement l'appel de routines en mode protégé à partir d'un gestionnaire d'interruption en mode réel est également possible grâce aux call backs.

Les spécifications du standard DMPI décrivent aussi des fonctions pour travailler sur les registres de mise au point du 80386 et de ses successeurs, pour interroger et tester des gestionnaires d'interruption en mode réel ou protégé, pour inhiber ou libérer des interruptions matérielles et naturellement aussi pour initialiser un client et effectuer une commutation en mode protégé.

Voici sous forme de tableau une récapitulation des différents services qu'un hôte DPMI peut offrir à un de ses clients :

Récapitulation des services de l'interface DPMI	
Interruption 2FH appel en mode réel	
1680h	Client inoccupé, transmettre l'exécution
1686h	Lire le mode d'exploitation (égal en mode protégé)
1687h	Déterminer si DPMI disponible
Interruption 31h appel en mode protégé uniquement	
Gestion des LDT	
0000h	Allouer un descripteur de segment de LDT
0001h	Libérer un descripteur de segment de LDT
0002h	Reproduire un segment de mode réel dans le descripteur de segment
0003h	Lire l'incrément pour les sélecteurs
0004h	Empêcher le swap d'un segment
0005h	Autoriser le swap d'un segment
0006h	Lire l'adresse de base d'un segment
0007h	Fixer l'adresse de base d'un segment
0008h	Fixer la longueur d'un segment
0009h	Fixer les droits d'accès et le type de segment
000Ah	Créer un alias pour un segment de code
000Bh	Lire un descripteur de segment
000Ch	Charger un descripteur de segment
000Dh	Réclamer un sélecteur
Accès à la mémoire DOS	
0100h	Allouer de la mémoire DOS
0101h	Libérer de la mémoire DOS
0102h	Modifier la taille d'un bloc de mémoire
Gestion des interruptions et des exceptions	
0200h	Lire l'adresse d'un gestionnaire d'interruption en mode réel
0201h	Installer un gestionnaire d'interruption
0202h	Lire l'adresse d'un gestionnaire d'exceptions
0203h	Installer un gestionnaire d'exceptions
0204h	Lire l'adresse d'un gestionnaire d'interruption en mode protégé
0205h	Installer un gestionnaire d'interruption en mode protégé
0900h	Inhiber l'indicateur d'interruption virtuel

Récapitulation des services de l'interface DPMI	
0901h	Libérer l'indicateur d'interruption virtuel
0902h	Lire l'indicateur d'interruption virtuel
Appel des routines en mode réel	
0300h	Simuler une interruption en mode réel
0301h	Appeler une routine en mode réel
0302h	Appeler une routine en mode réel
0303h	Générer un call back
0304h	Retourner un call back
Fonctions diverses	
0400h	Lire le numéro de version
Fonctions pour accéder à la mémoire étendue	
0500h	Lire des informations sur l'occupation de la mémoire
0501h	Allouer de la mémoire étendue
0502h	Libérer un bloc de mémoire
0503h	Modifier la taille d'un bloc de mémoire
0800h	Convertir une adresse physique en adresse linéaire
Fonctions de gestion des interruptions et des exceptions	
0600h	Protéger une zone de mémoire contre le swap
0601h	Déverrouiller une zone de mémoire
0602h	Déverrouiller une zone de mémoire en mode réel
0603h	Reverrouiller une zone de mémoire en mode réel
0604h	Lire la taille d'une page
0702h	Privilégier une zone de mémoire pour le remisage
0703h	Protéger une zone de mémoire contre l'écriture
Support des registres de mise au point du 80386 et de ses successeurs	
0B00h	Définir un point d'inspection (watchpoint)
0B01h	Effacer un point d'inspection
0B02h	Lire l'état d'un point d'inspection
0B03h	Réinitialiser l'état d'un point d'inspection

Initialisation du client et commutation en mode protégé

Tout appel DPMI doit être précédé d'une interrogation de l'hôte sinon les services DPMI sont inaccessibles. L'interrogation doit se faire en mode réel, après le lancement du client potentiel au niveau de DOS.

C'est la fonction 1678h qui va intervenir. L'hôte DPMI l'a intégrée lors de son installation dans l'interruption multiplexeur 2Fh de DOS. Si elle retourne en AX la valeur 0, un hôte DPMI est effectivement installé. Le bit 0 du registre BX indique alors s'il s'agit d'un hôte DPMI de 16 bits ou d'un hôte DPMI de 32 Bits. dans ce dernier cas le bit 0 est à 1.

En même temps le registre CL indique le type de processeur et le registre DX le numéro de version de l'hôte DPMI. En SI se trouve une information particulièrement importante. Elle donne en paragraphes la taille d'un bloc de mémoire dont l'hôte DPMI a besoin pour ses tâches de gestion et qui doit être alloué par le client avant la commutation en mode protégé.

La commutation proprement dite se fait par une routine dont l'adresse est renvoyée en ES:DI par la fonction multiplexeur. Le branchement doit s'effectuer par une instruction FAR CALL, le registre ES contenant le segment du bloc de mémoire mis à la disposition de l'hôte pour ses besoins de gestion.

Si la routine de l'hôte DMPI appelée par ES:DI renvoie un indicateur de retenue à 1, le programme se trouve toujours en mode réel parce que la commutation en mode protégé a échoué pour une raison que nous n'analyserons pas. Si l'indicateur de retenue est à 0, le programme est passé en mode protégé et peut maintenant appeler tous les services disponibles par l'intermédiaire de l'interruption 31h. Il faut noter qu'en cas d'erreur la plupart des fonctions DPMI rendent la main avec l'indicateur de retenue mis à 1, mais qu'elles ne livrent pas de code d'erreur. Voià un point d'amélioration possible pour les futures extensions de l'interface.

Même commuté en mode protégé le programme se trouve toujours en dessous de la barrière des 640 Ko, où il est exécuté normalement, les adresses de segments en CS, DSet SS étant remplacées par des sélecteurs. Ces derniers référencent des segments de 64 Ko dont les descripteurs ont été automatiquement installés par l'hôte. Les sélecteurs en DS et SS sont identiques lorsque les deux segments étaient confondus dans le mode réel.

ES représente un descripteur qui correspond au PSP du programme et mémorise une longueur de segment de 100h octets. Lorsque GS et FS sont disponibles, ils reçoivent la valeur 0.

Dès que la commutation en mode protégé est réussie, un DOS Extender va par exemple chercher à charger en mémoire étendue le programme qu'il a généré pour l'y mettre à exécution. Il faudra qu'il alloue de la mémoire étendue en la demandant à l'hôte DPMI

et qu'il fasse enregistrer les descripteurs de segments associés dans la table de descripteurs locale. La section suivante donnera plus d'informations à ce sujet.

L'exécution du programme en mode protégé se termine par l'appel à partir du mode protégé de la fonction terminale de DOS 4Ch (Int 21h). L'hôte DPMI repasse alors automatiquement en mode réel, retire de la mémoire le programme d'origine en mode réel (le chargeur) et revient au message d'attente de DOS.

Gestion de la table des descripteurs locale du client

Avec ses 14 fonctions, la gestion de la table des descripteurs locale paraît richement dotée - une peu trop à mon avis car on aurait pu regrouper certains appels. Ces fonctions sont néanmoins indispensables car en matière de gestion de la mémoire étendue l'interface DPMI se contente de distribuer des blocs de mémoire sans y associer de descripteur de segment. C'est le client qui est responsable de cette tâche. Mais le client ne peut pas accéder à sa table de descripteurs pour des raisons de privilège: il est donc obligé de se servir des fonctions de l'hôte DPMI.

La première à intervenir est la fonction 0000h grâce à laquelle le client peut créer un ou plusieurs descripteurs de segments dans sa table locale."Créer" veut dire ici que l'hôte installe une série de descripteurs de segments de données dans la table de descripteurs locale, ces descripteurs présentant tous une adresse de début de segment nulle et une longueur identique. Ce n'est que par des appels réitérés que le client peut fixer les adresses de début, les longueurs et les types souhaités (a priori le type est celui des segments de données mais il se peut que le client ait besoin de segments de code).

Le résultat fourni par la fonction 0000h est un sélecteur sur le descripteur créé. Si plusieurs descripteurs ont été créés, leurs "numéros" peuvent être lus par addition de la valeur retournée par la fonction 0003h. Compte tenu de la structure compliquée des sélecteurs, il est faux de supposer que le sélecteur renvoyé s'incrémente de 1 à chaque fois pour retrouver progressivement les sélecteurs suivants.

Avant la clôture du programme, tous les sélecteurs ainsi créés devront être restitués. C'est la fonction 0001h qui s'en charge pour un sélecteur à la fois.

Pour adresser un segment de mémoire en dessous de la limite de 1 Mo en mode protégé, il est également nécessaire de disposer d'un descripteur de segment et d'un sélecteur associé. Il suffit à cet effet de transmettre à la fonction 0002h l'adresse du segment en mode réel. Ladite fonction génère aussitôt un descripteur de segment qui référence un bloc de 64 Ko et renvoie le sélecteur qui pointe sur ce descripteur.

Un descripteur de segment créé par la fonction 000h ou 0002h peut être modifié par l'une des fonctions 0007h, 0008h ou 0009h. On peut ainsi fixer individuellement l'adresse de début, la longueur et les droits d'accès ou l'attribut du segment. Pour ce qui est de l'adresse de début et de la longueur, le client doit se préoccuper de ne pas accéder

à des segments qu'il n'a pas alloués et de ne pas faire se chevaucher les allocations. Ces vérifications ne sont pas faites par les fonctions citées. Leur absence peut conduire aux abus qui ont été évoqués plus haut. Les informations sont lues par l'instruction machine LSL (Load Segment Limit) et LAR (Load Access Rights) ou par la fonction 0006h qui retourne l'adresse de base d'un segment.

Pour charger dans un buffer un descripteur complet de segment, il faut se servir de la fonction 000Bh. Inversement c'est la fonction 000Ch qui charge un descripteur avec le contenu d'un buffer. L'hôte DPMI vérifie que le client ne s'accorde pas dans cette opération une priorité meilleure que celle qui lui est échue, à savoir 3.

Pour charger un programme en mémoire étendue, la fonction 000Ah s'avère très utile. A un segment de code donné elle associe un descripteur de segment de données où sont mémorisées l'adresse de début et la taille du segment de code et elle renvoie ensuite un sélecteur de référence.

Allocation de mémoire étendue

La gestion et la distribution de la mémoire étendue s'effectuent par le moyen de quatre fonctions numérotées 0500h, 0501h, 0502h et 0503h. La fonction 0500h renvoie quant à elle des informations d'état, de la quantité de mémoire libre jusqu'au fichier d'échange d'une gestion de mémoire virtuelle.

L'allocation de mémoire étendue est à la charge de la fonction 0501h à laquelle on doit fournir en BX:CX la taille demandée. Comme la mémoire de l'hôte DPMI est répartie en blocs de 4 Ko (en raison de la mémoire virtuelle, presque toujours en service dans le cas des "32 bits"), il est conseillé de faire des requêtes qui soient des multiples de 4 Ko.

Si la demande de mémoire a pu être satisfaite, les registres BX:CX contiennent à l'issue de l'appel l'adresse linéaire du bloc alloué. Les registres SI:DI fournissent un handle nécessaire pour traiter le même bloc par d'autres fonctions DPMI.

Après avoir invoqué avec succès la fonction 0501h le client est bien en possession du bloc requis mais il ne peut pas encore y accéder. Il lui manque le descripteur de segment associé comme nous l'avons déjà mentionné précédemment. Ce descripteur doit être créé par la fonction 000h puis rempli avec l'adresse de début et la taille appropriée.

Ce mécanisme un peu fastidieux a quand même des contreparties avantageuses: le client est par exemple libre de diviser la zone de mémoire créée en plusieurs segments contigus. Une telle démarche est même recommandable car l'interface DPMI ne dispose pas d'un collecteur de résidus qui rassemble les blocs libérés pour en faire des blocs de taille supérieure. Un client DPMI aura intérêt à demander la totalité de la mémoire nécessaire en un seul appel à la fonction 0501h puis d'établir les segments souhaités dans la zone attribuée.

Pour libérer un bloc de mémoire étendue on se sert de la fonction 0502h. Le bloc un question est identifié par le handle renvoyé lors de l'allocation par la fonction 0501h.

La dernière fonction de la bande des quatre qui gère la mémoire porte le numéro 0503h. Elle est destinée à modifier un bloc de mémoire déjà alloué. La modification concerne évidemment non pas son contenu mais sa taille, ce qui entraînera parfois son déplacement à l'intérieur de la mémoire étendue.

Comme argument d'entrée il faut fournir à la fonction le handle du bloc de mémoire et sa nouvelle taille. En retour elle renvoie la nouvelle adresse de début et un nouveau handle. Si le bloc de mémoire a été réellement déplacé, le client est responsable de l'ajustement de l'adresse de début dans le ou les descripteurs(s) de segment qu'il a créé(s) pour y accéder.

Allocation de mémoire DOS

Pour allouer et gérer la mémoire DOS, l'hôte DPMI offre les fonctions 0100h, 0101h, 0102h qui sont les équivalents des fonctions bien connues de DOS 48h, 49h et 4Ah.

Contrairement à la fonction d'allocation de mémoire étendue, la fonction 0100h renvoie tout de suite un descripteur de segment pour accéder au bloc alloué, de sorte que le client n'a pas à s'en occuper. Il faut lui fournir en entrée la taille en paragraphes de la zone souhaitée.

Le sélecteur renvoyé servira notamment à libérer la mémoire par la suite à l'aide de la fonction 0101h.

Il sert aussi d'identificateur pour agrandir ou réduire le bloc de mémoire: il faut alors le charger en DX avant d'appeler la fonction 0102h. BX devra contenir la nouvelle taille souhaitée pour le bloc de mémoire, exprimée en paragraphes.

Ce sont surtout les DOS Extenders qui ont besoin de blocs de mémoire de DOS lorsque des appels aux fonctions de DOS surviennent en mode protégé et que certaines informations doivent être transmises dans des buffers.

Les accès aux fichiers entrent souvent dans ce cas de figure. Mais sur ce point précis les spécifications de l'interface DPMI sont facilement améliorables. En règle générale on peut en effet supposer que l'hôte DPMI tourne sur une machine à base de 80386 ou i486 dont le mécanisme de pagination dessert également la gestion de la mémoire. Avec ce mécanisme et la disponibilité de fonctions DPMI appropriées il aurait été facile de représenter des blocs de mémoire étendue à l'intérieur de la mémoire conventionnelle en dessous de la limite de 1 Mo et de les rendre ainsi accessibles à DOS. Le client aurait ainsi évité de perdre son temps à copier des données.

Gestion de la mémoire virtuelle

La gestion de la mémoire virtuelle est assurée par les fonctions 0600h, 0601h, 0602h, 0603h et 0604h. Ces fonctions ne sont implémentées que sur les hôtes DPMI 32 bits car les processeurs 80286 ignorent les tables de pages et par conséquent toute mémoire virtuelle. Elles servent à interdire le swap de certaines zones de mémoire ("page-locking") ou à les désigner comme candidates au swap c'est-à-dire à l'enregistrement sur fichier d'échange. Les tâches sont prises en charge par les fonctions 0600h et 0601h. La première protège contre le stockage de la zone de mémoire dont l'adresse et la taille lui sont communiquées. Seront concernées toutes les pages qui sont situées totalement ou partiellement dans la zone indiquée. Ce verrouillage se justifie par exemple pour un gestionnaire d'interruption qui doit se déclencher immédiatement quand on l'appelle. Si la protection d'une page n'est plus indispensable, elle peut être déverrouillée par la fonction 0601h.

Alors que les fonctions 0600h et 0601h concernent la mémoire étendue allouée par le client au moyen de la fonction 0501h, les fonctions 0602h et 0603h s'appliquent à la mémoire située en dessous de la limite de 1 Mo qui a été allouée avec la fonction 0100h. Notez que l'ordre des deux fonctions est inversé par rapport à 0600h et 0601h: la fonction 0602h déverrouille les pages, tandis que la fonction 0603h les verrouille.

Dernière fonction du groupe, la fonction 0604h renvoie la taille d'une page.

Gestionnaire d'interruption et appel de routines en mode réel

La gestion des interruptions pose de graves problèmes lorsqu'on programme des DOS Extenders ou d'autres utilitaires en mode protégé, surtout lorsque les interruptions sont matérielles. Comment réagir lorsqu'une telle interruption se produit brusquement en mode protégé, alors que le gestionnaire compétent n'est disponible qu'en mode réel ? S'il est vrai que le standard VCPI laisse le programmeur livré à lui-même pour résoudre cette question, l'interface DPMI offre divers services pour faire face à l'urgence de la situation. Mais avant de décrire le détail de ces fonctions, je voudrais expliquer les idées qui les soutiennent. Il faut être conscient qu'il ne peut exister de conflit entre les exceptions et les interruptions matérielles car le premier contrôleur d'interruption est systématiquement reprogrammé par l'hôte. Le bas niveau de privilège du client le garantit contre la tentation de se livrer à la même activité.

C'est aussi l'hôte DPMI qui en cas d'interruption matérielle la déclenche en mode protégé même si elle est survenue pendant l'exécution d'un programme en mode réel ou en mode V86. Dans ce processus le gestionnaire standard DPMI reprend d'abord le contrôle des opérations puis il le rend au premier client qui a enregistré un gestionnaire pour cette interruption par le moyen de la fonction DPMI appropriée. La rencontre d'une instruction machine IRET destinée à rendre la main à l'appelant met aussi fin à la gestion des interruptions car les gestionnaires d'interruption des autres clients

n'arrivent plus à prendre leur tour (nous sommes dans un environnement multitâche). Il est donc recommandé avant d'installer un gestionnaire d'interruption matérielle de prendre connaissance du gestionnaire précédent et de l'appeler dans le cadre d'une routine d'interruption propre.

Si tous les gestionnaires appliquent ce principe il en résulte une sorte de chaîne et aucun gestionnaire d'interruption n'est brimé. Chaque client DPMI peut cependant se prémunir contre l'action des interruptions matérielles en appelant des fonctions appropriées. Car l'hôte DPMI gère pour chaque client une sorte d'indicateur d'interruption virtuel à l'intérieur du registre des indicateurs. Cet indicateur identifie les clients qui doivent être gardés à l'abri des interruptions matérielles alors qu'elles atteignent les autres.

Les interruptions logicielles déclenchées en mode réel se comportent différemment. Seules trois de ces interruptions parviennent jusqu'au mode protégé pour peu que le client ait prévu des handlers à cet effet: l'interruption du timer du BIOS, l'interruption Ctrl-C de DOS (23h) et l'interruption d'erreur critique (24h) qui fait aussi partie de DOS. Mais les interruptions logicielles peuvent aussi être déclenchées en mode protégé par exemple lorsqu'un DOS Extender gère une instruction machine INT. Tant qu'aucun client DPMI n'a installé un gestionnaire pour cette interruption elle entraîne une commutation en mode réel suivie de l'exécution de l'interruption et du retour en mode protégé.

Ce mécanisme est la plupart du temps pleinement justifié: car la majorité des interruptions logicielles sert à appeler des fonctions de DOS ou du BIOS. Mais à la réflexion les choses sont un peu moins simples. Ces fonctions reçoivent en effet des arguments par les registres du processeur. Les registres généraux ne posent pas de problème. Mais les contenus des registres de segments sont détruits au moment de la commutation en mode réel car ils sont interprétés comme des adresses alors que ce sont des sélecteurs.

C'est pour cela que les DOS Extenders s'immiscent dans les différentes interruptions logicielles et à l'intérieur-même du mode protégé convertissent en adresses de segments les sélecteurs dans les registres de segments, avant de transmettre l'interruption au mode réel.

Mais il est temps d'examiner les principales fonctions DPMI en matière d'interruption. Il existe d'abord six fonctions qui permettent d'installer des gestionnaires d'interruption en mode réel, des gestionnaires d'interruption et d'exception en mode protégé et de lire les adresses des gestionnaires déjà installés. Ces fonctions sont numérotées de 0200h à 0205h.

Avec la fonction DPMI 0300h l'hôte offre la possibilité de simuler une interruption en mode réel. Les gestionnaires d'interruptions logicielles en mode protégé en font un large usage. Ils interceptent les interruptions logicielles (BIOS, DOS, etc), testent le numéro de la fonction et sur la base de cette information transforment les pointeurs transmis par les registres du processeur en les adaptant au mode réel, après avoir préalablement recopié les données adressées dans un buffer DOS situé en dessous de la limite de 1 Mo.

Les fonctions 0301h et 0302h jouent un rôle similaire. Mais elles ne simulent pas des interruptions logicielles: elles appellent directement une routine en mode réel située en dessous de la limite de 1 Mo et dont l'adresse est connue. La différence entre les deux fonctions tient à la façon dont elles attendent que les routines se terminent. Avec la fonction 0301h la routine en mode réel doit se terminer par l'instruction en langage machine VAR RET, tandis qu'avec la fonction 0302h, la même fin doit se caractériser par une instruction IRET.

La situation inverse dans laquelle une routine en mode protégé doit être invoquée à partir du mode réel est également envisageable. Il suffit de penser par exemple à un driver de souris qui appelle une certaine routine dès que la souris est déplacée ou que l'état de la souris soit modifié. La routine en question doit évidemment être établie en mode réel, mais elle peut avoir à appeler une routine en mode protégé pour que l'événement souris soit pris en compte dans le cadre d'un programme en mode protégé (créé par un DOS Extender).

Grâce à la fonction 0303h l'hôte DPMI offre le moyen de générer un call back. Il s'agit d'une petite routine que l'hôte dépose de sa propre initiative en mémoire vive et qui lorsqu'on l'invoque, déclenche la routine en mode protégé qu'on lui indique. Mais seulement après commutation préalable en mode protégé.

Pour gérer les indicateurs d'interruption virtuels on dispose des fonctions 0900h, 0901h et 0902h. Elles se chargent de garder à distance les redoutables interruptions matérielles lorsqu'on le souhaite.

Accès aux registres de mise au point

Pour que chaque client DPMI puisse profiter un peu des registres de mise au point du 80386 et de ses successeurs, l'hôte DPMI centralise l'accès à ces registres au moyen de quatre fonctions. Chaque client peut installer ses propres points d'arrêt et points d'inspection (watchpoints). Les adresses de ces points sont chargées par l'hôte DPMI dans les registres de mise au point chaque fois que le client concerné entre en exécution.

La fonction 0B00h permet de définir un point d'inspection. Il faut lui fournir tous les paramètres que nécessite la mise en service des registres de mise au point: adresse linéaire du point d'arrêt, taille (Byte, Word, DWord) et type d'opération (lire la mémoire, y écrire ou l'exécuter). En retour on obtient un handle de point d'inspection à moins que le nombre de points déjà installés soit excessif ou qu'un paramètre invalide n'ait été transmis.

Un point d'inspection est effacé par la fonction 0B01h. Il faut communiquer à cette fonction le handle obtenu lors de la définition du point.

Tant qu'un point d'inspection est encore actif, son état peut être lu par la fonction 0B02h. On peut ainsi tester si le point a été atteint ou non. L'indicateur correspondant peut ensuite être remis à 0 par la fonction 0B03h pour permettre une nouvelle détection.

Annexes

A. Description des fonctions des interruptions du BIOS

Les interruptions 10h à 1Ah permettent d'accéder aux diverses fonctions qui rendent le BIOS en ROM disponible pour assurer la communication entre un programme et le matériel. Il s'agit non seulement des fonctions autorisant l'accès au matériel vidéo, au clavier, aux disques durs et unités de disquettes, mais aussi pour le contrôle des données de configuration ainsi que la programmation des interfaces série et parallèle et l'horloge alimentée par piles.

Voici d'abord la liste des différentes interruptions et leurs utilisations. Notez que les fonctions de l'interruption 13h sont citées deux fois selon qu'elles interviennent sur des unités de disquettes ou disques durs. Ainsi, choisissez le paragraphe qui vous intéresse sachant que vous souhaiter utiliser ces fonctions pour accéder à des disquettes ou des disques durs.

Sauf indication supplémentaire, les diverses fonctions conviennent à tous les types de PC. Mais il existe toute une série de fonctions ne concernant que les XT, d'autres les AT. Nous avons omis d'énumérer les innombrables extensions du BIOS développées par des sociétés telles que Compaq à l'origine du BIOS en ROM pour la bonne raison qu'elles ne sont pas reconnues comme des standards. Il en est de même des fonctions étendues du PS/2 d'IBM, mais elles sont cependant entièrement compatibles avec les fonctions présentées dans ce contexte.

Interruption 10h, Fonction 00h BIOS
Ecran : Fixer le mode vidéo

Cette fonction permet de sélectionner et initialiser un mode vidéo. L'écran est alors entièrement effacé. De ce fait, cette fonction peut aussi être appelée pour vider l'écran d'une façon très simple même lorsqu'on ne souhaite pas changer le mode vidéo.

Entrée : AH = 00h
AL = mode vidéo

0	40*25 car., noir et blanc	(Carte couleur)
1	40*25 car., couleur	(Carte couleur)
2	80*25 car., noir et blanc	(Carte couleur)
3	80*25 car., couleur	(Carte couleur)
4	320*200 points graph., 4 coul.	(Carte couleur)
5	320*200 points graph., 4 coul. (Couleurs affichées en noir et blanc)	(Carte couleur)
6	640*200 points graph., 2 coul.	(Carte couleur)
7	Mode interne de la carte mono	(Carte mono)
8	Graphique 160*200, 16 couleurs	(seulement PC-Junior)

9	Graphique 320*200, 16 couleurs (seulement PC-Junior)
0Ah	Graphique 640*200, 4 couleurs (seulement PC-Junior)
0Dh	Graphique 320*200, 16 couleurs (à partir de EGA)
0Eh	Graphique 640*200, 16 couleurs (à partir de EGA)
0Fh	Graphique 640*350, monochrome (à partir de EGA)
10h	Graphique 640*350, 16 couleurs (à partir de EGA)
11h	Graphique 640*480, 2 couleurs (à partir de MCGA)
12h	Graphique 640*480, 16 couleurs (VGA)
12h	Graphique 640*480, 256 couleurs (8514)
13h	Graphique 320*200, 256 couleurs (à partir de MCGA)

Sortie : aucune sortie

Remarques :
- Les couleurs pour les modes 4, 5 et 6 peuvent être fixées à l'aide de la fonction 11.
- Le contenu des registres BX, CX, DX et des registres SS, CS et DS n'est pas modifié. Le contenu de tous les autres registres, et notamment celui des registres SI et DI, peut avoir été modifié.

Interruption 10h, Fonction 01h BIOS
Ecran : Définition de l'apparence du curseur

Les lignes de départ et de fin du curseur de l'écran sont fixées en appelant cette fonction. Cette fonction ne dépend pas de la page écran actuellement affichée sur l'écran.

Entrée :
AH	= 01h
CH	= Ligne de départ du curseur
CL	= Ligne de fin du curseur

Sortie : aucune sortie

Remarques :
- Les valeurs autorisées pour les lignes de départ et de fin dépendent de la carte vidéo installée. Les valeurs suivantes sont autorisées :

Cartes écran monochromes :	0 - 13
Cartes écran couleur :	0 - 7

- Le BIOS fixe au départ les valeurs suivantes :

Cartes écran monochromes :	lignes 11 et 12
Cartes écran couleur :	lignes 6 et 7

- Il ne faut pas fixer des valeurs non autorisées à l'aide de cette fonction car cela peut avoir des conséquences imprévisibles (en général, une disparition du curseur).

- Le contenu des registres BX, CX, DX et des registres de segment SS, CS et DS n'est pas modifié par cette fonction. Le contenu de tous les autres registres, et notamment des registres SI et DI, peut avoir été modifié.

Interruption 10h, Fonction 02h BIOS
Ecran : Positionnement du curseur

Cette fonction permet de déplacer le curseur qui fixe la position de la sortie de caractères sur l'écran, à l'aide d'une des fonctions de sortie de caractères du BIOS.

Entrée : AH = 02h
BH = Numéro de la page écran
DH = Ligne de l'écran
DL = Colonne de l'écran

Sortie : aucune sortie

Remarques :
- Le curseur clignotant de l'écran n'est déplacé par cette fonction qu'à condition que la page écran appelée soit la page actuelle de l'écran.
- La ligne de l'écran est une valeur entre 0 et 24.
- La colonne de l'écran est une valeur comprise entre 0 et 79 (en 80 colonnes) ou entre 0 et 39 (en 40 colonnes), suivant le mode vidéo fixé.
- Une méthode pour faire disparaître le curseur clignotant consiste à le placer dans une position hors du champ de l'écran (par exemple colonne 0, ligne 25).
- Le numéro de la page écran dépend aussi du nombre de pages écran disponibles sur la carte vidéo utilisée.
- Le contenu des registres BX, CX, DX et des registres de segment SS, CS et DS n'est pas modifié par cette fonction. Le contenu de tous les autres registres, et notamment des registres SI et DI, peut avoir été modifié.

Interruption 10h, Fonction 03h BIOS
Ecran : Lecture de la position du curseur

Lecture de la position du curseur de texte dans une page écran et lecture des lignes de départ et de fin du curseur clignotant de l'écran.

Entrée : AH = 03h
BH = Numéro de la page écran

Sortie : DH = Ligne de l'écran dans laquelle figure le curseur
DL = Colonne de l'écran dans laquelle figure le curseur
CH = Ligne de départ du curseur clignotant de l'écran
CL = Ligne de fin du curseur clignotant de l'écran

Remarques :
- Le numéro de la page écran dépend aussi du nombre de pages écran disponibles sur la carte vidéo utilisée.
- Les ligne et colonne de l'écran se réfèrent au système de coordonnées de texte.
- Le contenu du registre BX et des registres de segment SS, CS et DS n'est pas modifié par cette fonction. Le contenu de tous les autres registres, et notamment des registres SI et DI, peut avoir été modifié.

Interruption 10h, Fonction 04h BIOS
Ecran : Lecture de la position du crayon optique

Si possible, la position du crayon optique sur l'écran est lue.

Entrée : AH = 04h

Sortie : AH = 0 : position du crayon optique ne peut être testée pour le moment.
AH = 1 : position du crayon optique a pu être obtenue, auquel cas
DH = Ligne écran du crayon optique (mode texte)
DL = Colonne écran du crayon optique (mode texte)
CH = Ligne écran du crayon optique (mode graphique)
BX = Colonne écran du crayon optique (mode graphique)

Remarques :
- L'appel de cette fonction doit être répété jusqu'à ce qu'un 1 soit renvoyé dans le registre AH car ce n'est que dans ce cas que les coordonnées peuvent être lues dans les autres registres.
- Les coordonnées indiquées se réfèrent naturellement au mode vidéo actuel et à sa résolution horizontale et verticale.
- Les coordonnées du crayon optique ne peuvent pas être déterminées très précisément, surtout en mode graphique. La coordonnée Y (ligne) est toujours une valeur paire de sorte qu'il n'est pas possible de distinguer si le crayon optique se trouve sur la ligne 8 ou 9. En mode graphique 320*200 points, la coordonnée X (colonne) est

toujours un multiple entier de 4 et en mode graphique 640*200 points toujours un multiple entier de 8.

- Le contenu du registre CL et des registres de segment SS, CS et DS n'est pas modifié par cette fonction. Le contenu de tous les autres registres, et notamment des registres SI et DI, peut avoir été modifié.

Interruption 10h, Fonction 05h BIOS
Ecran : Sélection de la page actuelle de l'écran

Sélection de la page écran actuelle, c'est-à-dire de celle qui doit être affichée sur l'écran (mode texte uniquement).

Entrée : AH = 05h
AL = Numéro de la page écran

Sortie : aucune sortie

Remarques :
- Le numéro de la page écran dépend aussi du nombre de pages écran disponibles sur la carte vidéo
- Lorsqu'une nouvelle page écran est activée le curseur clignotant de l'écran est toujours fixé sur la position du curseur de texte sur cette page.
- Le fait de passer d'une page écran à l'autre n'affecte pas le contenu de ces pages.
- Il n'est pas nécessaire qu'une page écran soit activée pour pouvoir y écrire.
- Le contenu des registres BX, CX, DX et des registres de segment SS, CS et DS n'est pas modifié par cette fonction. Le contenu de tous les autres registres, et notamment des registres SI et DI, peut avoir été modifié.

Interruption 10h, Fonction 06h BIOS
Ecran : Faire défiler des lignes de texte vers le haut (scrolling)

Faire défiler d'une ou plusieurs lignes vers le haut ou effacer une partie de la page écran actuelle

Entrée : AH = 06h
AL = Nombre de lignes dont la fenêtre doit être décalée vers le haut (0 signifie effacer la fenêtre)

CH = Ligne écran du coin supérieur gauche de la fenêtre
CL = Colonne écran du coin supérieur gauche de la fenêtre
DH = Ligne écran du coin inférieur droit de la fenêtre
DL = Colonne écran du coin inférieur droit de la fenêtre
BH = Couleur (attribut) pour les lignes vides

Sortie : aucune sortie

Remarques :
- Seule la page écran actuelle peut être modifiée par cette fonction
- Le fait d'effacer la zone de l'écran (nombre de lignes = 0) revient à la remplir d'espaces (code ASCII 32).
- Le contenu des lignes expulsées de l'écran lors du défilement est définitivement perdu et ne peut être récupéré.
- Pour vider l'écran tout entier, il est préférable d'employer la fonction 0 de cette interruption.
- Le contenu des registres BX, CX, DX et des registres de segment SS, CS et DS n'est pas modifié par cette fonction. Le contenu de tous les autres registres, et notamment des registres SI et DI, peut avoir été modifié.

Interruption 10h, Fonction 07h BIOS
Ecran : Faire défiler des lignes de texte vers le bas (scrolling)

Faire défiler d'une ou plusieurs lignes vers le haut ou effacer une partie de la page écran actuelle

Entrée : AH = 07h
AL = Nombre de lignes dont la fenêtre doit être décalée vers le bas (0 signifie effacer la fenêtre)
CH = Ligne écran du coin supérieur gauche de la fenêtre
CL = Colonne écran du coin supérieur gauche de la fenêtre
DH = Ligne écran du coin inférieur droit de la fenêtre
DL = Colonne écran du coin inférieur droit de la fenêtre
BH = Couleur (attribut) pour les lignes vides

Sortie : aucune sortie

Remarques :
- Seule la page écran actuelle peut être modifiée par cette fonction
- Le fait d'effacer la zone de l'écran (nombre de lignes = 0) revient à la remplir d'espaces (code ASCII 32).
- Le contenu des lignes expulsées de l'écran lors du défilement est définitivement perdu et ne peut être récupéré.

- Pour vider l'écran tout entier, il est préférable d'employer la fonction 0 de cette interruption.
- Le contenu des registres BX, CX, DX et des registres de segment SS, CS et DS n'est pas modifié par cette fonction. Le contenu de tous les autres registres, et notamment des registres SI et DI, peut avoir été modifié.

Interruption 10h, Fonction 08h BIOS
Ecran : Lecture d'un caractère/d'une couleur

Lecture du code ASCII et de la couleur (attribut) du caractère figurant dans la position actuelle du curseur.

Entrée : AH = 08h
BH = Numéro de la page écran

Sortie : AL = Code ASCII du caractère
AH = Couleur (attribut)

Remarques :
- Le numéro de la page écran dépend aussi du nombre de pages écran disponibles sur la carte vidéo
- Cette fonction peut également être appelée en mode graphique. Dans ce cas, le modèle de bits du caractère sur l'écran sera comparé avec les modèles de bits des caractères définis dans la ROM de caractères de la carte vidéo et avec les caractères qui ont été stockés dans une table en RAM dont l'adresse figure à l'interruption 1Fh. Si le caractère ne peut être identifié, le registre AL contiendra la valeur 0 après appel de la fonction.
- Le contenu des registres BX, CX, DX et des registres de segment SS, CS et DS n'est pas modifié par cette fonction. Le contenu de tous les autres registres, et notamment des registres SI et DI, peut avoir été modifié.

Interruption 10h, Fonction 09h BIOS
Ecran : Ecriture d'un caractère/d'une couleur

Ecrire un caractère dans une couleur déterminée dans la position actuelle du curseur (dans la page écran spécifiée).

Entrée : AH = 09h
BH = Numéro de la page écran
CX = Nombre d'écritures successives du caractère

AL = Code ASCII du caractère
BL = Attribut

Sortie : aucune sortie

Remarques :

- Si le caractère indiqué doit être sorti plusieurs fois (auquel cas la valeur du registre CX est supérieure à 1), il faut, en mode graphique, que tous les caractères rentrent dans la page écran actuelle.
- Les codes de commande "bell", "carriage return", etc..., ne sont pas identifiés comme tels mais sortis comme des codes ASCII normaux.
- Cette fonction permet également de sortir des caractères en mode graphique. Dans ce cas, les modèles des caractères portant les codes 0 à 127 sont tirés dans une table en ROM et les modèles des caractères portant les codes 128 à 255 sont tirés d'une table en RAM qui doit avoir été installée préalablement avec l'instruction GRAFTABL du DOS.
- En mode de texte, le contenu du registre BL définit l'octet d'attribut du caractère. En mode graphique, il détermine la couleur du caractère. En mode graphique 640*200 points, les valeurs 0 et 1, en mode graphique 320*200 points, les valeurs 0 à 3 sont possibles pour définir les différentes couleurs de la palette de couleurs sélectionnée.
- Si le mode graphique est activé lors de la sortie de caractères et si le bit 7 du registre BL est fixé, le modèle de caractère est combiné par un 'ou exclusif' avec les points graphiques figurant sous ce caractère.
- Cette fonction ne fait pas avancer le curseur vers la prochaine position de l'écran.
- Le contenu des registres BX, CX, DX et des registres de segment SS, CS et DS n'est pas modifié par cette fonction. Le contenu de tous les autres registres, et notamment des registres SI et DI, peut avoir été modifié.

Interruption 10h, Fonction 0Ah BIOS
Ecran : Ecriture d'un caractère

Un caractère est écrit dans la position actuelle du curseur dans la page écran spécifiée. La couleur de l'ancien caractère dans cette position de l'écran est maintenue.

Entrée : AH = 0Ah
BH = Numéro de la page écran

CX = Nombre d'écritures successives du caractère
AL = Code ASCII du caractère

Sortie : aucune sortie

Remarques :

- Si le caractère indiqué doit être sorti plusieurs fois (auquel cas la valeur du registre CX est supérieure à 1), il faut, en mode graphique, que tous les caractères rentrent dans la page écran actuelle.
- Les codes de commande "bell", "carriage return", etc..., ne sont pas identifiés comme tels mais sortis comme des codes ASCII normaux.
- Cette fonction permet également de sortir des caractères en mode graphique. Dans ce cas, les modèles des caractères portant les codes 0 à 127 sont tirés dans une table en ROM et les modèles des caractères portant les codes 128 à 255 sont tirés d'une table en RAM qui doit avoir été installée préalablement avec l'instruction GRAFTABL du DOS.
- En mode de texte, le contenu du registre BL définit l'octet d'attribut du caractère. En mode graphique, il détermine la couleur du caractère. En mode graphique 640*200 points, les valeurs 0 et 1, en mode graphique 320*200 points, les valeurs 0 à 3 sont possibles pour définir les différentes couleurs de la palette de couleurs sélectionnée.
- Si le mode graphique est activé lors de la sortie de caractères et si le bit 7 du registre BL est fixé, le modèle de caractère est combiné par un 'ou exclusif' avec les points graphiques figurant sous ce caractère.
- Cette fonction ne fait pas avancer le curseur vers la prochaine position de l'écran.

Le contenu des registres BX, CX, DX et des registres de segment SS, CS et DS n'est pas modifié par cette fonction. Le contenu de tous les autres registres, et notamment des registres SI et DI, peut avoir été modifié.

Interruption 10h, Fonction 0Bh, Sous-fonction 0 BIOS
Ecran : Sélection des couleurs de cadre et de fond

Cette fonction sert à sélectionner les couleurs de cadre et de fond pour les modes graphique et de texte.

Entrée : AH = 0Bh
BH = 0
BL = Couleurs de cadre/de fond

Sortie : aucune sortie

Remarques :
- En mode graphique, la valeur de couleur transmise définit aussi bien la couleur du cadre de l'écran que celle du fond de l'écran. En mode de texte, la couleur du fond est définie séparément pour chaque caractère, de sorte que la valeur de couleur transmise définit uniquement la couleur du cadre de l'écran.
- La valeur de couleur transmise peut être comprise entre 0 et 15 et peut donc représenter les 16 couleurs possibles.
- Le contenu des registres BX, CX, DX et des registres de segment SS, CS et DS n'est pas modifié par cette fonction. Le contenu de tous les autres registres, et notamment des registres SI et DI, peut avoir été modifié.

Interruption 10h, Fonction 0Bh, Sous-fonction 1 BIOS
Ecran : Sélection de la palette de couleurs

Sélection d'une des deux palettes de couleurs pour le mode graphique 320*200 points.

Entrée : AH = 0Bh
BH = 1
BL = Numéro de la palette de couleurs

Sortie : aucune sortie

Remarques :
- Deux palettes de couleurs sont disponibles. Elles portent les numéros 0 et 1 et contiennent les couleurs suivantes :

  Palette 0 : vert, rouge, jaune
  Palette 1 : turquoise, magenta, blanc
- Le contenu des registres BX, CX, DX et des registres de segment SS, CS et DS n'est pas modifié par cette fonction. Le contenu de tous les autres registres, et notamment des registres SI et DI, peut avoir été modifié.

Interruption 10h, Fonction 0Ch BIOS
Ecran : Ecrire un point graphique

Fixer la valeur de couleur pour un point écran en mode graphique.

Entrée : AH = 0Ch
DX = Ligne de l'écran

CX = Colonne de l'écran
AL = Valeur de couleur

Sortie : aucune sortie

Remarques :
- La valeur de couleur se réfère au mode graphique actuel.
- En mode 640*200 points, seules les valeurs 0 et 1 sont autorisées.
- En mode 320*200 points, les valeurs 0 à 3 sont autorisées. Elles génèrent une couleur en fonction de la palette de couleurs sélectionnée. La valeur 0 correspond à la couleur sélectionnée pour le fond, 1 à la première couleur de la palette sélectionnée, 2 à la seconde, etc...
- Le contenu des registres BX, CX, DX et des registres de segment SS, CS et DS n'est pas modifié par cette fonction. Le contenu de tous les autres registres, et notamment des registres SI et DI, peut avoir été modifié.

Interruption 10h, Fonction 0Dh BIOS
Ecran : Lire un point graphique

Lire la couleur d'un point écran en mode graphique.

Entrée : AH = 0Dh
DX = Ligne de l'écran
CX = Colonne de l'écran

Sortie : AL = Valeur de couleur

Remarques :
- La valeur de couleur se réfère au mode graphique actuel.
- En mode 640*200 points, seules les valeurs 0 et 1 sont possibles.
- En mode 320*200 points, les valeurs 0 à 3 sont autorisées. Elles génèrent une couleur en fonction de la palette de couleurs sélectionnée. La valeur 0 correspond à la couleur sélectionnée pour le fond, 1 à la première couleur de la palette sélectionnée, 2 à la seconde, etc...
- Le contenu des registres BX, CX, DX et des registres de segment SS, CS et DS n'est pas modifié par cette fonction. Le contenu de tous les autres registres, et notamment des registres SI et DI, peut avoir été modifié.

Interruption 10h, Fonction 0Eh BIOS
Ecran : Ecrire un caractère

Un caractère est écrit dans la position actuelle du curseur dans la page écran actuelle. La couleur de l'ancien caractère dans cette position de l'écran est maintenue.

Entrée : AH = 0Eh
AL = Code ASCII du caractère
BL = Couleur de premier plan du caractère (en mode graphique uniquement)

Sortie : aucune sortie

Remarques :
- Cette fonction n'interprète pas les différents codes de commande tels que "bell" et "carriage return" comme des codes ASCII normaux mais comme des codes de commande particuliers. La sortie du caractère "bell" produira par exemple un bip.
- Après sortie d'un caractère à l'aide de cette fonction, la position du curseur est incrémentée de sorte que le caractère suivant sera sorti dans la position suivante de l'écran. Lorsque la dernière position de l'écran est atteinte, l'écran défile d'une ligne vers le haut et la sortie se poursuit dans la première colonne de la dernière ligne de l'écran.
- La couleur de premier plan se réfère au mode graphique actuel.
- En mode 640*200 points, seules les valeurs 0 et 1 sont possibles.
- En mode 320*200 points, les valeurs 0 à 3 sont autorisées. Elles génèrent une couleur en fonction de la palette de couleurs sélectionnée. La valeur 0 correspond à la couleur sélectionnée pour le fond, 1 à la première couleur de la palette sélectionnée, 2 à la seconde, etc...
- Le contenu des registres BX, CX, DX et des registres de segment SS, CS et DS n'est pas modifié par cette fonction. Le contenu de tous les autres registres, et notamment des registres SI et DI, peut avoir été modifié.

Interruption 10h, Fonction 0Fh BIOS
Ecran : Lecture du mode vidéo

Lire le numéro du mode vidéo actuel, le nombre de caractères par ligne et le numéro de la page écran actuelle.

Entrée : AH = 0Fh

Sortie : AL = Mode vidéo
(voir fonction 00h)
AH = Nombre de caractères par ligne
BH = Numéro de la page écran actuelle

Remarques :
- Le contenu des registres BL, CX, DX et des registres de segment SS, CS et DS n'est pas modifié par cette fonction. Le contenu de tous les autres registres, et notamment des registres SI et DI, peut avoir été modifié.

Interruption 10h, Fonction 13h BIOS (A partir de AT)
Ecran : Sortie d'une chaîne de caractères

Une chaîne de caractères est sortie sur l'écran, dans une page écran déterminée, à partir d'une position spécifiée de l'écran. Les caractères sont retirés d'un buffer dont l'adresse doit être transmise à la fonction.

Entrée : AH = 13h
AL = Mode de sortie (0 - 3)
0 = Attribut dans BL, conserver position du curseur
1 = Attribut dans BL, actualiser position du curseur
2 = Attribut dans le buffer, conserver position du curseur
3 = Attribut dans le buffer, actualiser position du curseur
BL = Octet d'attribut des caractères (modes 0 et 1 seulement)
CX = Nombre de caractères à sortir
DH = Ligne de l'écran
DL = Colonne de l'écran
BH = Page écran
ES = Adresse de segment du buffer
BP = Adresse d'offset du buffer

Sortie : aucune sortie

Remarques :
- En modes 1 et 3, la position du curseur est fixée à la suite du dernier caractère de la chaîne de caractères sortie, de sorte que, lors des appels ultérieurs d'une fonction BIOS de sortie de caractères, les caractères seront sortis à la suite de la chaîne de caractères. Ce n'est pas le cas en modes 0 et 2.
- En modes 0 et 1, le buffer contient uniquement les codes ASCII des caractères à sortir. La couleur de tous les caractères de la chaîne de caractères est dans ce cas fixée par le registre BL. En modes 2 et 3, par contre, chaque caractère est suivi, dans le buffer, de l'octet d'attribut correspondant de sorte que chaque caractère est doté d'un attribut individuel. Bien que la taille en octets de la chaîne de

caractères soit deux fois supérieure au nombre de caractères à sortir, c'est bien ce nombre de caractères ASCII qui doit être spécifié dans le registre CX et non la longueur effective de la chaîne.

- Les codes de commande spéciaux tels que "bell" et "carriage return" sont interprétés comme des codes de commande et non comme des codes ASCII normaux.
- Lorsqu'est atteinte la dernière position de l'écran, l'écran défile d'une ligne vers le haut et la sortie se poursuit sur la première colonne de la dernière ligne de l'écran.
- Le contenu des registres BX, CX, DX et des registres de segment SS, CS et DS n'est pas modifié par cette fonction. Le contenu de tous les autres registres, et notamment des registres SI et DI, peut avoir été modifié.

Interruption 11h BIOS
Déterminer la configuration

Déterminer quelle est la configuration du système telle qu'elle a été constatée lors de l'opération de lancement du système.

Entrée : aucune entrée

Sortie : AX = la configuration

Pour le PC et le XT :

Bit 0	vaut 1 si le système dispose d'un ou plusieurs lecteurs de disquette
Bit 1	inutilisé
Bits 2 et 3	Mémoire RAM sur la carte mère 00 = 16 Ko 01 = 32 Ko 10 = 48 Ko 11 = 64 Ko
Bits 4 et 5	Mode vidéo lors du lancement du système 00 : inutilisé 01 : 40*25 car. sur carte couleur 02 : 80*25 car. sur carte couleur 03 : 80*25 car. sur carte mono
Bits 6 et 7	indiquent le nombre de lecteurs de disquette si le bit 0 vaut 1 00 = 1 lecteur de disquette 01 = 2 lecteurs de disquette 10 = 3 lecteurs de disquette

	11 = 4 lecteurs de disquette
Bit 8	vaut 0 s'il y a un circuit DMA
Bits 9 à 11	Nombre de cartes RS232 connectées
Bit 12	vaut 1 si l'adaptateur de jeux est connecté
Bit 13	inutilisé
Bits 14 et 15	indiquent le nombre d'imprimantes

Pour l'AT :

Bit 0	vaut 1 si le système dispose d'un ou plusieurs lecteurs de disquette
Bit 1	vaut 1 si un coprocesseur mathématique est implanté sur le système
Bits 2 et 3	inutilisés
Bits 4 et 5	Mode vidéo lors du lancement du système 00 : inutilisé 01 : 40*25 car. sur carte couleur 02 : 80*25 car. sur carte couleur 03 : 80*25 car. sur carte mono
Bits 6 et 7	indiquent le nombre de lecteurs de disquette si le bit 0 vaut 1 00 = 1 lecteur de disquette 01 = 2 lecteurs de disquette 10 = 3 lecteurs de disquette 11 = 4 lecteurs de disquette
Bit 8	sans signification
Bits 9 à 11	Nombre de cartes RS232 connectées
Bits 12 et 13	inutilisés
Bits 14 et 15	indiquent le nombre d'imprimantes

Remarques :

- Pour pouvoir interpréter la signification des différents bits du mot de configuration, il faut connaître le type du PC (PC/XT ou AT).
- La taille de la mémoire indiquée dans les bits 2 et 3 du mot de configuration du PC/XT se réfère uniquement à la carte mère. L'interruption 12h permet de connaître la taille totale de la mémoire.
- Le mode vidéo indiqué dans les bits 4 et 5 correspond au mode vidéo qui était activé lors de la mise en marche du système. Pour connaître le mode vidéo actuel, c'est d'une autre fonction, la fonction 15 de l'interruption 10h qu'il convient donc de se servir.
- Seul le contenu du registre AX est modifié à la suite de l'appel de cette fonction.

Interruption 12h BIOS
Déterminer la taille de la mémoire

Entrée : aucune entrée

Sortie : AX = Taille de la mémoire en Ko

Remarques :
- Le PC et le XT ne peuvent recevoir plus de 640 Ko de RAM alors que l'AT peut encore recevoir jusqu'à 14 Mo de mémoire RAM au-delà de la limite de 1 Mo. La taille de cette mémoire supplémentaire n'est pas prise en compte dans la taille de mémoire indiquée par cette fonction. Pour connaître le volume de la mémoire au-delà de la limite de 1 Mo, vous pouvez utiliser la fonction 88h de l'interruption 15h. Cette fonction n'existe d'ailleurs que sur l'AT.
- Seul le contenu du registre AX est modifié par l'appel de cette fonction.

Interruption 13h, Fonction 00h BIOS
Disquette : Réinitialisation du lecteur de disquette

L'appel de cette fonction déclenche une réinitialisation du contrôleur de disquette ainsi que des lecteurs de disquette connectés. Il convient d'effectuer une réinitialisation (Reset) après toute opération disquette à la suite de laquelle une erreur a été signalée.

Entrée : AH = 00h
DL = 0 ou 1

Sortie : Flag Carry = 0 : Opération exécutée, dans ce cas AH=0
Flag Carry = 1 : Erreur, dans ce cas AH=Code d'erreur

Remarques :
- La valeur dans le registre DL ne sert à rien en fait car la réinitialisation est effectuée sur tous les lecteurs de disquette. Elle est cependant utilisée sur le XT et l'AT pour signaler si la réinitialisation doit concerner les lecteurs de disquette ou le disque dur.
- Les codes d'erreur suivants peuvent se présenter :

  01h : Numéro de fonction non autorisé
  02h : Marque d'adresse non trouvée
  03h : Tentative d'écriture sur disquette protégée contre l'écriture
  04h : Secteur appelé non trouvé
  08h : Débordement DMA
  09h : Transfert de données par-delà la limite de segment
  10h : Erreur de lecture

20h : Erreur sur le contrôleur de disque
40h : Piste non trouvée
80h : Erreur de Time Out, le lecteur ne réagit pas

- Le contenu des registres BX, CX, DX, SI, DI, BP et des registres de segment SS, CS et DS n'est pas modifié par cette fonction. Le contenu de tous les autres registres peut avoir été modifié.

Interruption 13h, Fonction 01h BIOS
Disquette : Lire l'état disquette

Cette fonction permet de consulter l'état de la dernière opération disquette.

Entrée : AH = 01h
DL = 0 ou 1

Sortie : Flag Carry =0 : Opération exécutée, dans ce cas AH=0
Flag Carry =1 : Erreur, dans ce cas AH=Code d'erreur

Remarques :

- La valeur dans le registre DL ne sert à rien en fait car l'état est toujours renvoyé pour le lecteur de disquette auquel il avait été accédé en dernier. Il est cependant utilisé sur le XT et l'AT pour distinguer si c'est l'état du lecteur de disquette ou du disque dur qui doit être consulté.
- Codes d'erreur, cf fontion 00h
- Le contenu des registres BX, CX, DX, SI, DI, BP et des registres de segment n'est pas modifié par cette fonction. Le contenu de tous les autres registres peut avoir été modifié.

Interruption 13h, Fonction 02h BIOS
Disquette : Lecture

Cette fonction permet de lire un ou plusieurs secteurs sur la disquette pour les transférer dans un buffer.

Entrée : AH = 02h
DL = Numéro du lecteur de disquette
DH = Numéro de la face de disquette (0 ou 1)
CH = Numéro de piste
CL = Numéro de secteur
AL = Nombre de secteurs à lire
ES = Adresse de segment du buffer
BX = Adresse d'offset du buffer

Sortie : Flag Carry =0 : Opération exécutée, dans ce cas AH=0
Flag Carry =1 : Erreur, dans ce cas AH=Code d'erreur

Remarques :
- Le nombre de secteurs à lire, qui est spécifié dans le registre AL, est limité par le fait qu'un seul appel de fonction ne permet de lire que des secteurs consécutifs sur une même piste d'une même face de disquette.
- Codes d'erreur, cf. Fonction 00h.
- Le contenu des registres BX, CX, DX, SI, DI, BP et des registres de segment n'est pas modifié par cette fonction. Le contenu de tous les autres registres peut avoir été modifié.

Interruption 13h, Fonction 03h BIOS
Disquette : Ecriture

Cette fonction permet d'écrire un ou plusieurs secteurs sur la disquette. Les données à transférer sont tirées d'un buffer.

Entrée :
AH = 03h
DL = Numéro du lecteur de disquette
DH = Numéro de la face de disquette (0 ou 1)
CH = Numéro de piste
CL = Numéro de secteur
AL = Nombre de secteurs à écrire
ES = Adresse de segment du buffer
BX = Adresse d'offset du buffer

Sortie : Flag Carry =0 : Opération exécutée, dans ce cas AH=0
Flag Carry =1 : Erreur, dans ce cas AH=Code d'erreur

Remarques :
- Le nombre de secteurs à écrire, qui est spécifié dans le registre AL, est limité par le fait qu'un seul appel de fonction ne permet d'écrire que sur des secteurs consécutifs sur une même piste d'une même face de disquette.
- Codes d'erreur, cf. Fonction 00h.
- Le contenu des registres BX, CX, DX, SI, DI, BP et des registres de segment n'est pas modifié par cette fonction. Le contenu de tous les autres registres peut avoir été modifié.

Interruption 13h, Fonction 04h BIOS
Disquette : Vérification

Cette fonction permet de comparer un ou plusieurs secteurs de la disquette avec les données d'un buffer en mémoire. Cela permet par exemple, à la suite d'une instruction d'écriture, de vérifier si les données ont été correctement transférées sur la disquette.

Entrée :
- AH = 04h
- DL = Numéro du lecteur de disquette
- DH = Numéro de la face de disquette (0 ou 1)
- CH = Numéro de piste
- CL = Numéro de secteur
- AL = Nombre de secteurs à vérifier
- ES = Adresse de segment du buffer
- BX = Adresse d'offset du buffer

Sortie : Flag Carry =0 : Opération exécutée, dans ce cas AH=0
Flag Carry =1 : Erreur, dans ce cas AH=Code d'erreur

Remarques :
- Le nombre de secteurs à vérifier, qui est spécifié dans le registre AL, est limité par le fait qu'un seul appel de fonction ne permet de vérifier que des secteurs consécutifs sur une même piste d'une même face de disquette.
- Codes d'erreur, cf. Fonction 00h.
- Le contenu des registres BX, CX, DX, SI, DI, BP et des registres de segment n'est pas modifié par cette fonction. Le contenu de tous les autres registres peut avoir été modifié.

Interruption 13h, Fonction 05h BIOS
Disquette : Formatage

En appelant cette fonction, vous pouvez formater une piste complète d'une face de la disquette. Il faut communiquer à la fonction, à cet effet, l'adresse d'un buffer contenant certaines informations sur les secteurs à formater.

Entrée :
- AH = 05h
- DL = Numéro du lecteur de disquette
- DH = Numéro de la face de disquette (0 ou 1)
- CH = Numéro de piste
- AL = Nombre de secteurs à vérifier
- ES = Adresse de segment du buffer
- BX = Adresse d'offset du buffer

Sortie : Flag Carry =0 : Opération exécutée, dans ce cas AH=0
Flag Carry =1 : Erreur, dans ce cas AH=Code d'erreur

Remarques :

- Le nombre de secteurs à formater, qui est spécifié dans le registre AL, est limité par le fait qu'un seul appel de fonction ne permet de formater que des secteurs consécutifs sur une même piste d'une même face de disquette.
- En ce qui concerne les AT et les systèmes équipés d'un processeur 80386 ou 80486, il faut appeler la fonction 17h avant d'appeler cette fonction pour déterminer le type de la disquette à formater.
- Le buffer dont l'adresse est fournie dans ES:BX contient pour chaque secteur à formater une entrée composée de 4 octets consécutifs :

  1. Numéro de piste
  2. Numéro de face
  3. Numéro logique du secteur
  4. Nombre d'octets dans ce secteur
     0 : 128 octets
     1 : 256 octets
     2 : 512 octets (Standard PC)
     3 : 1024 octets

  Les numéros de piste et de face doivent être identiques pour tous les secteurs alors que le numéro de secteur doit être au contraire différent pour chacun. Ce numéro de secteur peut être décalé par rapport au numéro de secteur physique.
- Codes d'erreur, cf. Fonction 00h.
- Le contenu des registres BX, CX, DX, SI, DI, BP et des registres de segment n'est pas modifié par cette fonction. Le contenu de tous les autres registres peut avoir été modifié.

Interruption 13h, Fonction 06h BIOS
Disque dur

Cette fonction sert à accéder aux disques durs et non aux unités de disquettes.

Interruption 13h, Fonction 07h BIOS
Disque dur

Cette fonction sert à accéder aux disques durs et non aux unités de disquettes.

Interruption 13h, Fonction 08h BIOS (à partir de AT)
Disquette : Déterminer le format

Cette fonction permet de déterminer la capacité d'une unité de disquettes et le format des disquettes. On se réfère ici aux capacités physiques de l'unité et non au format de la disquette qui y est insérée.

Entrée : AH = 08h
DL = Numéro de l'unité de disquettes

Sortie : Flag Carry = 0 : Opération exécutée, dans ce cas

AH = 0
BL = Type de l'unité
01h = 5,25", 360 Ko
02h = 5,25", 1,2 Mo
03h = 3,5", 720 Ko
04h = 3,5", 1,44 Mo
DH = Numéro de page supérieur (toujours 1)
CH = Numéro de piste supérieur
CL = Numéros de secteurs supérieurs

Flag Carry = 1 : Erreur, dans ce cas AH = Code d'erreur

Remarques :
- Le contenu des registres SI, BP et des registres de segment CS, DS et SS n'est pas modifié par cette fonction. Le contenu de tous les autres registres peut avoir été modifié.

Interruption 13h, Fonctions de 09h à 14h BIOS
Disque dur

Ces fonctions servent à accéder aux disques durs et non aux unités de disquettes.

Interruption 13h, Fonction 15h BIOS (A partir de AT)
Disquette : Déterminer le type de lecteur

L'AT soutient non seulement les anciens lecteurs 320/360 Ko mais aussi les nouveaux lecteurs 1,2 Mo. Ces nouveaux lecteurs disposent, contrairement aux précédents, de la possibilité de détecter un changement de disquette. Cette fonction vous permet de tester si le lecteur utilisé dispose de cette possibilité.

Entrée : AH = 15h
DL = Numéro du lecteur de disquette (0 ou 1)

Sortie : Flag Carry =0 : Operation exécutée, dans ce cas
AH =type du lecteur
0 : Périphérique absent
1 : Lecteur ne détecte pas les changements de disquette
2 : Lecteur détecte les changements de disquette
3 : Disque dur (voir remarques)
1 : Erreur

Remarques :
- Il se peut que l'unité testée soit un disque dur car l'AT comporte un contrôleur qui peut gérer, au choix, 2 lecteurs de disquette et un disque dur ou bien un lecteur de disquette et 2 disques durs. Dans ce dernier cas, le premier disque dur porte le numéro 1 et peut donc être testé à l'aide de cette fonction.
- Le contenu des registres BX, CX, DX, SI, DI, BP et des registres de segment n'est pas modifié par cette fonction. Le contenu de tous les autres registres peut avoir été modifié.

Interruption 13h, Fonction 16h BIOS (A partir de AT)
Disquette : Détecter un changement de disquette

L'AT soutient non seulement les anciens lecteurs 320/360 Ko mais aussi les nouveaux lecteurs 1,2 Mo. Ces nouveaux lecteurs disposent, contrairement aux précédents, de la possibilité de détecter un changement de disquette. Si vous disposez d'un lecteur de disquette de ce type, cette fonction vous permet de tester si un changement de disquette est intervenu depuis le dernier accès à la disquette.

Entrée : AH = 16h
DL = Numéro du lecteur de disquette (0 ou 1)

Sortie : AH = 0 : Pas de changement de disquette
AH = 6 : Disquette changée depuis le dernier accès disquette

Remarques :
- Le contenu des registres BX, CX, DX, SI, DI, BP et des registres de segment n'est pas modifié par cette fonction. Le contenu de tous les autres registres peut avoir été modifié.

Interruption 13h, Fonction 17h BIOS (A partir de AT)
Disquette : Fixer format de disquette

Cette fonction permet de configurer un lecteur par rapport à un format de disquette précis. Cette fonction est déjà utilisée dans les versions 3.0 et 3.1 de DOS, mais à partir de la version 3.2, cette tâche est remplie par la fonction 18h. Les programmes doivent également éviter cette fonction d'autant qu'elle ne convient pas aux lecteurs 3,5" qui se sont largement répandus dans le monde PC.

Entrée : AH = 17h
AL = Format
1: formater en 320/360 Ko sur un lecteur 320/360 Ko
2: formater en 320/360 Ko sur un lecteur 1,2 Mo
3: formater en 1,2 Mo sur un lecteur 1,2 Mo
4: formater en 720 Ko sur un lecteur 720 Ko

Sortie : Flag Carry =0 : Opération exécutée
Flag Carry =1 : Erreur

Remarques :
- Codes d'erreur, cf. Fonction 00h.
- Le contenu des registres BX, CX, DX, SI, DI, BP et des registres de segment n'est pas modifié par cette fonction. Le contenu de tous les autres registres peut avoir été modifié.

Interruption 13h, Fonction 18h BIOS (à partir de AT)
Disquette : Fixer format de disquette

Cette fonction remplace la fonction 17h. Elle doit être appelée pour fixer le format des disquettes. Mais elle ne doit être appelée qu'après avoir inséré une disquette dans le lecteur spécifié, sinon elle signale une erreur.

Entrée : AH = 18h
CH = Numéro de piste
CL = Numéro de secteur
DL = Numéro du lecteur (0 ou 1)

Sortie : Flag Carry = 0 : Opération exécutée
Flag Carry = 1 : Erreur

Remarques :
- Dans ce cas, cette fonction doit être appelée avant le premier appel de la fonction de formatage 05h pour que le BIOS puisse être configuré au format de disquette souhaité.
- Codes d'erreur, cf. Fonction 00h.

- Le contenu des registres BX, CX, DX, SI, DI, BP et des registres de segment n'est pas modifié par cette fonction. Le contenu de tous les autres registres peut avoir été modifié.

Interruption 13h, Fonction 00h BIOS
Disque dur : Reset

Si vous appelez cette fonction, un Reset est effectué aussi bien sur le contrôleur de disque dur que sur les lecteurs de disque dur connectés. Il convient d'effectuer une réinitialisation après toute opération disque dur à la suite de laquelle une erreur a été signalée.

Entrée : AH = 00h
DL = 80h ou 81h

Sortie : Flag Carry =0 : Opération exécutée
Flag Carry =1 : Erreur, dans ce cas AH=Code d'erreur

Remarques :

- Le premier lecteur de disque dur porte le numéro 80h, le second le numéro 81h.
- La valeur dans le registre DL ne sert à rien en fait car une réinitialisation est effectuée sur tous les lecteurs de disque dur. Elle est utilisée sur le XT et sur l'AT pour distinguer si la réinitialisation doit concerner les lecteurs de disquette ou de disque dur.
- Les codes d'erreur suivants peuvent apparaître :

  01h : Numéro de fonction non autorisé ou lecteur appelé absent
  02h : Marque d'adresse non trouvée
  04h : Secteur non trouvé
  05h : Erreur lors de réinitialisation contrôleur
  07h : Erreur lors d'initialisation du contrôleur
  09h : Erreur de transmission DMA : transfert de données par-delà la limite de segment
  0Ah : Secteur défectueux
  10h : Erreur de lecture
  11h : Erreur de lecture corrigée par ECC
  20h : Contrôleur défectueux
  40h : Opération de recherche infructueuse
  80h : Lecteur ne répond pas (Time Out)
  AAh : Lecteur n'est pas prêt
  CCh : Erreur en écriture

- Le contenu des registres BX, CX, DX, SI, DI, BP et des registres de segment n'est pas modifié par cette fonction. Le contenu de tous les autres registres peut avoir été modifié.

Interruption 13h, Fonction 01h BIOS
Disque dur : Consulter état du disque dur

Cette fonction permet de connaître l'état de la dernière opération sur disque dur.

Entrée : AH = 01h
DL = 80h ou 81h

Sortie : Flag Carry =0 : Opération exécutée
Flag Carry =1 : Erreur, dans ce cas AH=Code d'erreur

Remarques :
- Le premier lecteur de disque dur porte le numéro 80h, le second le numéro 81h.
- La valeur dans le registre DL ne sert à rien en fait car l'état est toujours renvoyé pour le disque dur auquel il a été accédé en dernier lieu. Elle est utilisée sur le XT et sur l'AT pour distinguer si c'est l'état d'un lecteur de disquette ou de disque dur qui doit être obtenu.
- Codes d'erreur, cf. Fonction 00h.
- Le contenu des registres BX, CX, DX, SI, DI, BP et des registres de segment n'est pas modifié par cette fonction. Le contenu de tous les autres registres peut avoir été modifié.

Interruption 13h, Fonction 02h BIOS
Disque dur : Lecture

Cette fonction permet de lire un ou plusieurs secteurs sur le disque dur, pour les transférer dans un buffer.

Entrée : AH = 02h
DL = Numéro de disque dur (80h ou 81h)
DH = Numéro de tête de lecture/écriture
CH = Numéro de cylindre
CL = Numéro de secteur
AL = Nombre de secteurs à lire (1 - 128)
ES = Adresse de segment du buffer
BX = Adresse d'offset du buffer

Sortie : Flag Carry =0 : Opération exécutée
Flag Carry =1 : Erreur, dans ce cas AH=Code d'erreur

Remarques :
- Le premier lecteur de disque dur porte le numéro 80h, le second le numéro 81h.

- Les 8 bits du registre CH ne permettant d'adresser que 256 cylindres, les bits 6 et 7 du numéro de secteur (registre CL) représentent les bits 8 et 9 du numéro de cylindre et permettent ainsi d'adresser jusqu'à 1023 cylindres.
- Si plusieurs secteurs sont lus et que le dernier secteur d'un cylindre est atteint, la lecture se poursuit sur le premier secteur du même cylindre sur la prochaine tête de lecture/écriture. Une fois la dernière tête de lecture/écriture atteinte, la lecture se poursuit sur le premier secteur du cylindre suivant sur la première tête de lecture/écriture.
- Codes d'erreur, cf. Fonction 00h.
- Le contenu des registres BX, CX, DX, SI, DI, BP et des registres de segment n'est pas modifié par cette fonction. Le contenu de tous les autres registres peut avoir été modifié.

Interruption 13h, Fonction 03h BIOS
Disque dur : Ecriture

Cette fonction permet d'écrire sur un ou plusieurs secteurs du disque dur. Les données à transférer sont tirées d'un buffer mis en place dans le programme d'appel.

Entrée :
AH = 03h
DL = Numéro de disque dur (80h ou 81h)
DH = Numéro de tête de lecture/écriture
CH = Numéro de cylindre
CL = Numéro de secteur
AL = Nombre de secteurs à écrire (1 - 128)
ES = Adresse de segment du buffer
BX = Adresse d'offset du buffer

Sortie : Flag Carry =0 : Opération exécutée
Flag Carry =1 : Erreur, dans ce cas AH=Code d'erreur

Remarques :
- Le premier lecteur de disque dur porte le numéro 80h, le second le numéro 81h.
- Les 8 bits du registre CH ne permettant d'adresser que 256 cylindres, les bits 6 et 7 du numéro de secteur (registre CL) représentent les bits 8 et 9 du numéro de cylindre et permettent ainsi d'adresser jusqu'à 1023 cylindres.
- Si on écrit sur plusieurs secteurs et que le dernier secteur d'un cylindre est atteint, l'écriture se poursuit sur le premier secteur du même cylindre sur la prochaine tête de lecture/écriture. Une fois la dernière tête de lecture/écriture atteinte, l'écriture se poursuit sur

le premier secteur du cylindre suivant sur la première tête de lecture/écriture.

- Codes d'erreur, cf. Fonction 00h.
- Le contenu des registres BX, CX, DX, SI, DI, BP et des registres de segment n'est pas modifié par cette fonction. Le contenu de tous les autres registres peut avoir été modifié.

Interruption 13h, Fonction 04h BIOS
Disque dur : Vérification

Cette fonction permet de vérifier un ou plusieurs secteurs du disque dur. Contrairement à la fonction disquette équivalente, cette fonction ne compare cependant pas les données du disque dur avec des données en mémoire. 4 octets de contrôle sont en effet systématiquement sauvegardés avec chaque secteur. Ils permettent ainsi de vérifier le contenu d'un secteur, même bien après qu'il ait été sauvegardé.

Entrée :
AH = 04h
DL = Numéro de disque dur (80h ou 81h)
DH = Numéro de tête de lecture/écriture
CH = Numéro de cylindre
CL = Numéro de secteur
AL = Nombre de secteurs à vérifier (1 - 128)
ES = Adresse de segment du buffer
BX = Adresse d'offset du buffer

Sortie :
Flag Carry =0 : Opération exécutée
Flag Carry =1 : Erreur, dans ce cas AH=Code d'erreur

Remarques :

- Le premier lecteur de disque dur porte le numéro 80h, le second le numéro 81h.
- Les 8 bits du registre CH ne permettant d'adresser que 256 cylindres, les bits 6 et 7 du numéro de secteur (registre CL) représentent les bits 8 et 9 du numéro de cylindre et permettent ainsi d'adresser jusqu'à 1023 cylindres.
- Si on vérifie plusieurs secteurs et que le dernier secteur d'un cylindre est atteint, la vérification se poursuit sur le premier secteur du même cylindre sur la prochaine tête de lecture/écriture. Une fois la dernière tête de lecture/écriture atteinte, la vérification se poursuit sur le premier secteur du cylindre suivant sur la première tête de lecture/écriture.
- Codes d'erreur, cf. Fonction 00h.

- Le contenu des registres BX, CX, DX, SI, DI, BP et des registres de segment n'est pas modifié par cette fonction. Le contenu de tous les autres registres peut avoir été modifié.

Interruption 13h, Fonction 05h BIOS
Disque dur : Formatage

Cette fonction permet de formater un cylindre complet, c'est-à-dire 17 secteurs du disque dur. La fonction a cependant besoin pour cela que le programme d'appel lui communique dans un buffer un certain nombre d'informations supplémentaires.

Entrée :
AH = 05h
DL = Numéro de disque dur (80h ou 81h)
DH = Numéro de tête de lecture/écriture
CH = Numéro de cylindre
CL = 1
AL = 17
ES = Adresse de segment du buffer
BX = Adresse d'offset du buffer

Sortie : Flag Carry =0 : Opération exécutée
Flag Carry =1 : Erreur, dans ce cas AH=Code d'erreur

Remarques :

- Le premier lecteur de disque dur porte le numéro 80h, le second le numéro 81h.
- Les 8 bits du registre CH ne permettant d'adresser que 256 cylindres, les bits 6 et 7 du numéro de secteur (registre CL) représentent les bits 8 et 9 du numéro de cylindre et permettent ainsi d'adresser jusqu'à 1023 cylindres.
- Comme c'est toujours un cylindre complet qui est formaté à la fois, le premier secteur à formater, dans le registre CL, est toujours le secteur 1. Pour la même raison, le nombre de secteurs à formater, dans le registre AL, est toujours 17 car les disques durs standard travaillent avec 17 secteurs par cylindre.
- Le buffer dont l'adresse est fournie en ES:BX doit avoir une taille de 512 octets au moins. Les 17 secteurs à formater d'un cylindre n'occupent toutefois que les 34 premiers octets de ce buffer. Les informations correspondant à chaque secteur physique sont chaque fois stockées dans deux octets consécutifs. Le premier octet n'a aucune signification avant appel de la fonction mais indique après l'appel de fonction si le secteur a pu être formaté (00h) ou non (80h). Le second octet spécifie le numéro de secteur logique. Il doit être inscrit dans le buffer par le programme d'appel avant appel de la fonction.

- Codes d'erreur, cf. Fonction 00h.
- Le contenu des registres BX, CX, DX, SI, DI, BP et des registres de segment n'est pas modifié par cette fonction. Le contenu de tous les autres registres peut avoir été modifié.

Interruption 13h, Fonction 06h BIOS
Disque dur : Formatage XT

Auparavant, cette fonction servait à formater les disques durs XT. A l'heure actuelle, elle ne joue aucun rôle dans la programmation.

Interruption 13h, Fonction 07h BIOS

Auparavant, cette fonction servait à formater les disques durs XT. A l'heure actuelle, elle ne joue aucun rôle dans la programmation.

Interruption 13h, Fonction 08h BIOS (XT AT)
Disque dur : Déterminer format

En appelant cette fonction, un programme peut connaître le format ou les caractéristiques du disque dur.

Entrée : AH = 08h
DL = Numéro de disque dur (80h ou 81h)

Sortie : Flag Carry =0 : Opération exécutée, dans ce cas
DL = Nombre de disques durs connectés
DH = Nombre de têtes de lecture/écriture (0=première tête)
CH = Numéro de cylindre
CL = Numéro de secteur
Flag Carry =1 : Erreur, dans ce cas AH=Code d'erreur

Remarques :
- Le premier lecteur de disque dur porte le numéro 80h, le second le numéro 81h.
- Les 8 bits du registre CH ne permettant d'adresser que 256 cylindres, les bits 6 et 7 du numéro de secteur (registre CL) représentent les bits 8 et 9 du numéro de cylindre et permettent ainsi d'adresser jusqu'à 1023 cylindres.

- La formule suivante permet de calculer la capacité totale du disque dur en octets :

  Capacité = Têtes * Cylindres * Secteurs * 512

- Codes d'erreur, cf. Fonction 00h.
- Le contenu des registres BX, CX, DX, SI, DI, BP et des registres de segment n'est pas modifié par cette fonction. Le contenu de tous les autres registres peut avoir été modifié.

Interruption 13h, Fonction 09h BIOS
Disque dur : Adaptation de disques durs étrangers

Cette fonction permet d'adapter des lecteurs étrangers de façon à ce que les fonctions du BIOS puissent y accéder.

Entrée : AH = 09h
DL = Numéro de disque dur (80h ou 81h)

Sortie : Flag Carry =0 : Opération exécutée
Flag Carry =1 : Erreur, dans ce cas AH=Code d'erreur

Remarques :

- Le premier lecteur de disque dur porte le numéro 80h, le second le numéro 81h.
- Le BIOS tire les informations nécessaires sur le disque dur à adapter (nombre de têtes de lecture/écriture, etc...) d'une table. L'adresse de cette table figure dans le vecteur d'interruption 41h pour le disque dur numéro 80h et dans le vecteur 46h pour le disque dur numéro 81h. C'est sous ces vecteurs que vous trouverez de plus amples renseignements sur cette table.
- Codes d'erreur, cf. Fonction 00h.
- Le contenu des registres BX, CX, DX, SI, DI, BP et des registres de segment n'est pas modifié par cette fonction. Le contenu de tous les autres registres peut avoir été modifié.

Interruption 13h, Fonction 0Ah BIOS
Disque dur : Lecture étendue

Cette fonction permet de lire un ou plusieurs secteurs sur le disque dur pour les transférer dans un buffer. Outre les 512 octets que comporte le secteur, seront également lus les 4 octets de contrôle (ECC) qui sont stockés à la fin de chaque secteur.

Entrée :
- AH = 0Ah
- DL = Numéro de disque dur (80h ou 81h)
- DH = Numéro de tête de lecture/écriture
- CH = Numéro de cylindre
- CL = Numéro de secteur
- AL = Nombre de secteurs à lire (1 - 127)
- ES = Adresse de segment du buffer
- BX = Adresse d'offset du buffer

Sortie : Flag Carry =0 : Opération exécutée
Flag Carry =1 : Erreur, dans ce cas AH=Code d'erreur

Remarques :
- Le premier lecteur de disque dur porte le numéro 80h, le second le numéro 81h.
- Les 4 octets de contrôle sont normalement calculés par le contrôleur mais ici il les tire directement du buffer.
- Les 8 bits du registre CH ne permettant d'adresser que 256 cylindres, les bits 6 et 7 du numéro de secteur (registre CL) représentent les bits 8 et 9 du numéro de cylindre et permettent ainsi d'adresser jusqu'à 1023 cylindres.
- Si on lit plusieurs secteurs et que le dernier secteur d'un cylindre est atteint, la lecture se poursuit sur le premier secteur du même cylindre sur la prochaine tête de lecture/écriture. Une fois la dernière tête de lecture/écriture atteinte, la lecture se poursuit sur le premier secteur du cylindre suivant sur la première tête de lecture/écriture.
- Codes d'erreur, cf. Fonction 00h.
- Le contenu des registres BX, CX, DX, SI, DI, BP et des registres de segment n'est pas modifié par cette fonction. Le contenu de tous les autres registres peut avoir été modifié.

Interruption 13h, Fonction 0Bh BIOS
Disque dur : Ecriture étendue

Cette fonction permet d'écrire un ou plusieurs secteurs sur le disque dur provenant d'un buffer. Outre les 512 octets que comporte le secteur, seront également écrits les 4 octets de contrôle (ECC), tirés du buffer, qui sont stockés à la fin de chaque secteur.

Entrée :
- AH = 0Bh
- DL = Numéro de disque dur (80h ou 81h)
- DH = Numéro de tête de lecture/écriture
- CH = Numéro de cylindre
- CL = Numéro de secteur
- AL = Nombre de secteurs à écrire (1 - 127)

ES = Adresse de segment du buffer
BX = Adresse d'offset du buffer

Sortie : Flag Carry =0 : Opération exécutée
Flag Carry =1 : Erreur, dans ce cas AH=Code d'erreur

Remarques :
- Le premier lecteur de disque dur porte le numéro 80h, le second le numéro 81h.
- Les 4 octets de contrôle sont normalement calculés par le contrôleur mais ici il les tire directement du buffer.
- Les 8 bits du registre CH ne permettant d'adresser que 256 cylindres, les bits 6 et 7 du numéro de secteur (registre CL) représentent les bits 8 et 9 du numéro de cylindre et permettent ainsi d'adresser jusqu'à 1023 cylindres.
- Si on écrit sur plusieurs secteurs et que le dernier secteur d'un cylindre est atteint, l'écriture se poursuit sur le premier secteur du même cylindre sur la prochaine tête de lecture/écriture. Une fois la dernière tête de lecture/écriture atteinte, l'écriture se poursuit sur le premier secteur du cylindre suivant sur la première tête de lecture/écriture.
- Codes d'erreur, cf. Fonction 00h.
- Le contenu des registres BX, CX, DX, SI, DI, BP et des registres de segment n'est pas modifié par cette fonction. Le contenu de tous les autres registres peut avoir été modifié.

Interruption 13h, Fonction 0Ch BIOS (à partir de XT)
Disque dur : Déplacer tête de lecture/écriture

Cette fonction permet de positionner l'une des têtes de lecture/écriture du disque dur sur un cylindre donné. Cette option s'avère particulièrement utile lors d'un transport pour éviter que la tête ne vienne s'écraser sur un cylindre utilisé.

Entrée : AH = 0Ch
DL = Numéro du lecteur de disque (80h ou 81h)
DH = Numéro de la tête de lecture/écriture
CH = Numéro du cylindre
CL = Numéro du secteur

Sortie : Flag Carry = 0 : Opération exécutée
Flag Carry = 1 : Erreur, dans ce cas AH = Code d'erreur.

Remarques :
- Le premier lecteur de disque porte le numéro 80h, le second le numéro 81h.

- Comme les 8 bits du registre CH ne permettent d'adresser que 256 cylindres, les bits 6 et 7 constituent le numéro de secteur (registre CL), les bits 8 et 9 le numéro de cylindre et permettent ainsi d'adresser jusqu'à 1023 cylindres.
- Lors des opérations de lecture/écriture, la tête de lecture/écriture se positionne automatiquement, si bien qu'il n'est pas nécessaire d'appeler préalablement cette fonction.
- Codes d'erreur, cf. Fonction 00h.
- Le contenu des registres BX, CX, DX, SI, DI, BP et des registres de segment n'est pas modifié par cette fonction. Le contenu de tous les autres registres peut avoir été modifié.

Interruption 13h, Fonction 0Dh BIOS
Disque dur : Réinitialisation

Lorsque cette fonction est appelée, une réinitialisation est opérée sur le contrôleur de disque dur ainsi que sur les disques durs connectés. Il convient d'effectuer une réinitialisation après toute opération sur disque dur à la suite de laquelle une erreur a été signalée.

Entrée : AH = 0Dh
DL = (80h ou 81h)

Sortie : Flag Carry =0 : Opération exécutée
Flag Carry =1 : Erreur, dans ce cas AH=Code d'erreur

Remarques :
- La valeur dans le registre DL ne sert à rien en fait car la réinitialisation est systématiquement opérée sur tous les disques durs connectés. Elle est cependant utilisée, sur le XT et l'AT, pour distinguer si la réinitialisation doit concerner les lecteurs de disquette ou le disque dur.
- Cette fonction est identique à la fonction 0.
- Le premier lecteur de disque dur porte le numéro 80h, le second le numéro 81h.
- Codes d'erreur, cf. Fonction 00h.
- Le contenu des registres BX, CX, DX, SI, DI, BP et des registres de segment n'est pas modifié par cette fonction. Le contenu de tous les autres registres peut avoir été modifié.

Interruption 13h, Fonction 0Eh BIOS (PS/2)
Disque dur : Test de lecture du contrôleur

Cette fonction effectue un test interne de la transmission des données entre le contrôleur et le CPU dans le cas des ordinateurs PS/2 uniquement. Elle n'est pas conçue pour être appelée par un programme d'application.

Interruption 13h, Fonction 0Fh BIOS (PS/2)
Disque dur : Test d'écriture du contrôleur

Cette fonction effectue un test interne de la transmission des données entre le contrôleur et le CPU dans le cas des ordinateurs PS/2 uniquement. Elle n'est pas conçue pour être appelée par un programme d'application.

Interruption 13h, Fonction 10h BIOS
Disque dur : Le lecteur est-il prêt ?

Cette fonction permet de déterminer si le lecteur de disque dur est prêt, c'est-à-dire si la dernière opération est déjà achevée.

Entrée : AH = 10h
DL = (80h ou 81h)

Sortie : Flag Carry =0 : Opération exécutée
Flag Carry =1 : Erreur, dans ce cas AH=Code d'erreur

Remarques :
- Le premier lecteur de disque dur porte le numéro 80h, le second le numéro 81h.
- Codes d'erreur, cf. Fonction 00h.
- Le contenu des registres BX, CX, DX, SI, DI, BP et des registres de segment n'est pas modifié par cette fonction. Le contenu de tous les autres registres peut avoir été modifié.

Interruption 13h, Fonction 11h BIOS
Disque dur : Recalibrage du lecteur

Après apparition d'une erreur (surtout s'il s'agit d'une erreur de lecture ou écriture), il est conseillé d'opérer non seulement une réinitialisation mais aussi un recalibrage du disque dur.

Entrée : AH = 11h
DL = (80h ou 81h)

Sortie : Flag Carry =0 : Opération exécutée
Flag Carry =1 : Erreur, dans ce cas AH=Code d'erreur

Remarques :
- Le premier lecteur de disque dur porte le numéro 80h, le second le numéro 81h.
- Codes d'erreur, cf. Fonction 00h.
- Le contenu des registres BX, CX, DX, SI, DI, BP et des registres de segment n'est pas modifié par cette fonction. Le contenu de tous les autres registres peut avoir été modifié.

Interruption 13h, Fonction 12h BIOS (PS/2)
Disque dur : Test RAM du contrôleur

Cette fonction n'effectue un test interne du contrôleur RAM que pour les ordinateurs PS/2. Elle n'est donc pas conçue pour être appelée par un programme d'application.

Interruption 13h, Fonction 13h BIOS (PS/2)
Disque dur : Test de lecteur

Cette fonction n'effectue un test interne de la mécanique du lecteur que pour les ordinateurs PS/2. Elle n'est donc pas conçue pour être appelée par un programme d'application.

Interruption 13h, Fonction 15h BIOS (A partir de AT)
Disque dur : Déterminer le type de lecteur

L'AT contient un contrôleur qui peut gérer aussi bien des disques durs que des lecteurs de disquette. Il peut gérer soit 2 lecteurs de disquette et un disque dur, soit 2 disques durs et un lecteur de disquette. Cette fonction vous permet donc de déterminer si les périphériques portant les numéros 80h et 81h sont des lecteurs de disquette ou de disque dur.

Entrée : AH = 15h
DL = Numéro de lecteur (80h ou 81h)

Sortie : Flag Carry =0 : Opération exécutée, dans ce cas AH = 03h
CX = Mot de poids fort du nombre de secteurs

DX = Mot de poids faible du nombre de secteurs
Flag Carry =1 : Erreur, pas de lecteur de disque dur

Remarques :
- Le premier lecteur de disque dur porte le numéro 80h, le second le numéro 81h.
- La capacité du disque dur en octets se calcule d'après la formule ((CX*65536) + DX) * 512
- Le contenu des registres BX, CX, DX, SI, DI, BP et des registres de segment n'est pas modifié par cette fonction. Le contenu de tous les autres registres peut avoir été modifié.

Interruption 13h, Fonction 19h BIOS
Disque dur : Parquer les têtes

Cette fonction permet de parquer les têtes du disque dur avant un transport.

Entrée : AH = 19h
DL = (80h ou 81h)

Sortie : Flag Carry =0 : Opération exécutée
Flag Carry =1 : Erreur, dans ce cas AH=Code d'erreur

Remarques :
- Le premier lecteur de disque dur porte le numéro 80h, le second le numéro 81h.
- Codes d'erreur, cf. Fonction 00h.
- Le contenu des registres BX, CX, DX, SI, DI, BP et des registres de segment n'est pas modifié par cette fonction. Le contenu de tous les autres registres peut avoir été modifié.

Interruption 14h, Fonction 00h BIOS
Interface série : Initialisation

Cette fonction permet d'initialiser et de configurer une interface série connectée sur le PC en définissant la parité de la transmission ainsi que le nombre de bits Stop et la vitesse de transmission en bauds.

Entrée : AH = 00h
DX = Numéro d'interface série (la première interface série porte le numéro 0)
AL = Paramètres de configuration
Bits 0-1 : largeur de données

10(b) = 7 bits
11(b) = 8 bits

Bit 2 : Nombre de bits Stop
0(b) = 1 bit Stop
1(b) = 2 bits Stop

Bit 3-4 : Contrôle de parité
00(b) = aucun
01(b) = impaire
11(b) = paire

Bits 5-7 : Vitesse de transmission
000(b) = 110 bauds
001(b) = 150 bauds
010(b) = 300 bauds
011(b) = 600 bauds
100(b) = 1200 bauds
101(b) = 2400 bauds
110(b) = 4800 bauds
111(b) = 9600 bauds

Sortie : AH = Etat de l'interface série
Bit 0 : Données prêtes
Bit 1 : Données effacées
Bit 2 : Erreur de parité
Bit 3 : Protocole n'a pas été respecté
Bit 4 : Interruption détectée
Bit 5 : Transmission Hold Register vide
Bit 6 : Transmission Shift Register vide
Bit 7 : Time Out (Périphérique ne répond pas)

AL = Etat du modem
Bit 0 : (Delta) Modem prêt à émettre
Bit 1 : (Delta) Modem activé
Bit 2 : (Delta) Téléphone sonne
Bit 3 : (Delta) Liaison avec modem récepteur établie
Bit 4 : Modem prêt à émettre
Bit 5 : Modem activé
Bit 6 : Téléphone sonne
Bit 7 : Liaison avec modem récepteur établie

Remarques :

- L'interface série appelée COM1 par le DOS porte le numéro 0, COM2 le numéro 1.
- Les bits Delta montrent la modification de l'état en cours par rapport au dernier appel de la fonction. Si un bit Delta est posé, cela signifie que l'état correspondant a changé entre-temps.
- Le contenu des registres BX, CX, DX, SI, DI, BP et des registres de segment n'est pas modifié par cette fonction. Le contenu de tous les autres registres peut avoir été modifié.

Interruption 14h, Fonction 01h BIOS
Interface série : Emission de caractères

Cette fonction est appelée pour envoyer des caractères à travers une interface série.

Entrée : AH = 01h
DX = Numéro d'interface série (la première interface série porte le numéro 0)
AL = Code du caractère à sortir

Sortie : AH : Bit 7 = 0 : Caractère a été transmis
Bit 7 = 1 : Erreur, dans ce cas :
Bits 0 à 6 = Etat de l'interface série
Bit 0 : Données prêtes
Bit 1 : Données effacées
Bit 2 : Erreur de parité
Bit 3 : Protocole n'a pas été respecté
Bit 4 : Interruption détectée
Bit 5 : Transmission Hold Register vide
Bit 6 : Transmission Shift Register vide

Remarques :

- L'interface série appelée COM1 par le DOS porte le numéro 0, COM2 le numéro 1.
- Si le bit 7 du registre AH est réglé après l'appel de la fonction, on est en présence d'une erreur Time Out. Les bits restants spécifient alors avec précision la cause de l'erreur.
- Le contenu des registres AL, BX, CX, DX, SI, DI, BP et des registres de segment n'est pas modifié par cette fonction. Le contenu de tous les autres registres peut avoir été modifié.

Interruption 14h, Fonction 02h BIOS
Interface série : Réception de caractères

Reçoit un caractère de l'interface série.

Entrée : AH = 02h
DX = Numéro d'interface série (la première interface série porte le numéro 0)

Sortie : AH : Bit 7 = 0 : Caractère a été reçu, dans ce cas :
AL = Caractère reçu
Bit 7 = 1 : Erreur, dans ce cas :
Bits 0-6 : Etat de l'interface série

Bit 0 : Données prêtes
Bit 1 : Données effacées
Bit 2 : Erreur de parité
Bit 3 : Protocole n'a pas été respecté
Bit 4 : Interruption détectée
Bit 5 : Transmission Hold Register vide
Bit 6 : Transmission Shift Register vide

Remarques :

- L'interface série appelée COM1 par le DOS porte le numéro 0, COM2 le numéro 1.
- Il convient de n'appeler cette fonction qu'après que la fonction 3 ait permis de constater qu'un caractère était prêt à être reçu.
- Le contenu des registres BX, CX, DX, SI, DI, BP et des registres de segment n'est pas modifié par cette fonction. Le contenu de tous les autres registres peut avoir été modifié.

Interruption 14h, Fonction 03h BIOS
Interface série : Tester état

Teste l'état de l'interface série et du modem éventuellement connecté.

Entrée : AH = 03h
DX = Numéro d'interface série (la première interface série porte le numéro 0)

Sortie : AH = Etat de l'interface série
Bit 0 : Données prêtes
Bit 1 : Données effacées
Bit 2 : Erreur de parité
Bit 3 : Protocole n'a pas été respecté
Bit 4 : Interruption détectée
Bit 5 : Transmission Hold Register vide
Bit 6 : Transmission Shift Register vide
Bit 7 : Time Out (Périphérique ne répond pas)

AL = Etat du modem
Bit 0 : (Delta) Modem prêt à émettre
Bit 1 : (Delta) Modem activé
Bit 2 : (Delta) Téléphone sonne
Bit 3 : (Delta) Liaison avec modem récepteur établie
Bit 4 : Modem prêt à émettre
Bit 5 : Modem activé
Bit 6 : Téléphone sonne
Bit 7 : Liaison avec modem récepteur établie

Remarques :

- L'interface série appelée COM1 par le DOS porte le numéro 0, COM2 le numéro 1.
- Cette fonction doit être appelée avant d'appeler la fonction 2 (Réception de caractères) pour déterminer si un caractère est prêt à être reçu. Dans ce cas, le bit 0 du registre AH vaut 1.
- Le contenu des registres BX, CX, DX, SI, DI, BP et des registres de segment n'est pas modifié par cette fonction. Le contenu de tous les autres registres peut avoir été modifié.

Interruption 15h, Fonction 83h — BIOS (A partir de AT)
Fixer flag après délai

L'appel de cette fonction a pour effet de fixer sur 1 le bit 7 d'un flag spécifié par le programme d'appel, au bout d'un délai indiqué en microsecondes.

Entrée :
AH = 83h
ES = Adresse de segment du flag
BX = Adresse d'offset du flag
CX = Mot fort du délai exprimé en microsecondes
DX = Mot faible du délai exprimé en microsecondes

Sortie : Aucune

Remarques :

- Une microseconde correspond à un millionième de seconde.
- Le contenu des registres BX, CX, DX, SI, DI, BP et des registres de segment n'est pas modifié par cette fonction. Le contenu de tous les autres registres peut avoir été modifié.

Interruption 15h, Fonction 84h, Sous-fonction 0 — BIOS (A partir de AT)
Test de l'état des boutons de feu des joysticks

Si un adaptateur de jeux avec les joysticks correspondants est connecté sur le PC, l'état des boutons Feu peut être testé à l'aide de cette fonction.

Entrée :
AH = 84h
DX = 0

Sortie :
Flag Carry =1 : Pas d'adaptateur de jeux connecté
Flag Carry =0 : Adaptateur de jeux présent, dans ce cas
AL = Etat des boutons Feu
Bit 7 = 1 : Premier bouton Feu du premier joystick enfoncé
Bit 6 = 1 : Second bouton Feu du premier joystick enfoncé

Bit 5 = 1 : Premier bouton Feu du second joystick enfoncé
Bit 4 = 1 : Second bouton Feu du second joystick enfoncé

Remarques :
- La position des joysticks peut être testée à l'aide de la sous-fonction 1.
- Le contenu des registres BX, CX, DX, SI, DI, BP et des registres de segment n'est pas modifié par cette fonction. Le contenu de tous les autres registres peut avoir été modifié.

Interruption 15h, Fonction 84h, Sous-fonction 1 BIOS (A partir de AT)
Test de la position des joysticks

Si un adaptateur de jeux avec les joysticks correspondants est connecté sur le PC, la position des joysticks peut être testée à l'aide de cette fonction.

Entrée : AH = 84h
DX = 1

Sortie : Flag Carry =1 : Pas d'adaptateur de jeux connecté
Flag Carry =0 : Adaptateur de jeux présent, dans ce cas
AX = Position X du premier joystick
BX = Position Y du premier joystick
CX = Position X du second joystick
DX = Position Y du second joystick

Remarques :
- L'état des boutons Feu peut être testé à l'aide de la sous-fonction 0.
- Le contenu des registres SI, DI, BP et des registres de segment n'est pas modifié par cette fonction. Le contenu de tous les autres registres peut avoir été modifié.

Interruption 15h, Fonction 85h BIOS (A partir de AT)
Touche System Request actionnée

Si la touche SysReq est enfoncée ou relâchée, cette fonction sera appelée par la routine du clavier.

Entrée : AH = 85h
AL = 0 : Touche a été enfoncée
AL = 1 : Touche a été relâchée

Sortie : aucune

Remarques :
- Cette fonction n'est pas conçue pour être appelée directement par un programme d'application mais elle peut être redirigée par un programme d'application sur une routine de ce programme, de façon à ce que le fait d'actionner la touche SysReq soit enregistré et entraîne les actions appropriées.

Interruption 15h, Fonction 86h — BIOS (A partir de AT)
Attendre

Après appel de cette fonction, le contrôle n'est rendu au programme d'appel qu'une fois écoulé un délai déterminé.

Entrée :
AH = 86h
CX = Mot de poids fort de la pause en microsecondes
DX = Mot de poids faible de la pause en microsecondes

Sortie : aucune

Remarques :
- Une microseconde correspond à un millionième de seconde.
- Le contenu des registres BX, SI, DI, BP et des registres de segment n'est pas modifié par cette fonction. Le contenu de tous les autres registres peut avoir été modifié.

Interruption 15h, Fonction 87h — BIOS (A partir de AT)
Transfert de zones de mémoire

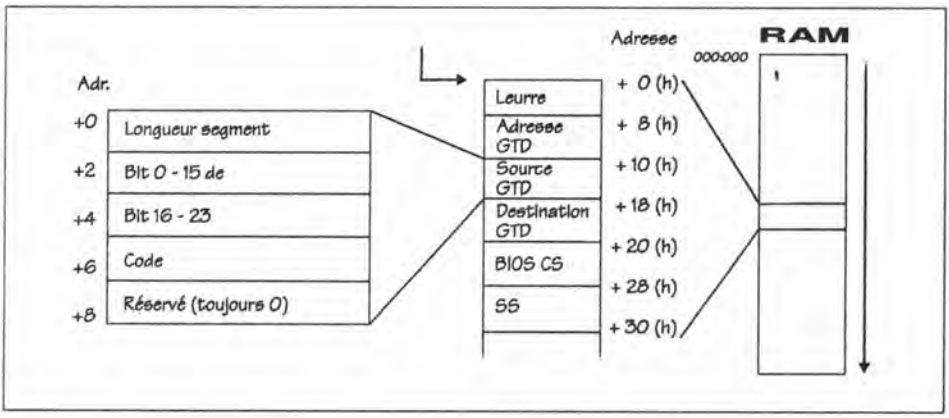
Cette fonction permet de transférer un nombre déterminé de cellules de mémoire entre la RAM sous la limite de 1 Mo et la RAM située au-dessus de cette limite.

Entrée :
AH = 87h
CX = Nombre de mots à transférer
ES = Adresse de segment de la Global Descriptor Table
SI = Adresse d'offset de la Global Descriptor Table

Sortie :
Flag Carry =0 : aucune erreur
Flag Carry =1 : Erreur, dans ce cas
AH = 1 : Erreur de parité de la RAM
AH = 2 : GDT incorrecte lors d'appel de fonction
AH = 3 : Mode protégé n'a pu être initialisé

Remarques :
- Seuls des mots, et non des octets isolés, peuvent être transférés.

- 64 Ko au maximum peuvent être transférés. La valeur dans le registre CX ne doit donc pas être supérieure à 8000h.
- Toutes les interruptions sont désactivées pendant le transfert du bloc de mémoire.
- Le contenu des registres BX, CX, DX, SI, DI, BP et des registres de segment n'est pas modifié par cette fonction. Le contenu de tous les autres registres peut avoir été modifié.



Structure et disposition des descripteurs de segment tels que les attend la fonction 87h

Interruption 15h, Fonction 88h BIOS (A partir de AT)
Déterminer taille de mémoire au-delà de 1 Mo

Cette fonction permet de déterminer la taille de la mémoire installée au-delà de la limite de 1 Mo.

Entrée : AH = 88h

Sortie : AX = Taille de la mémoire

Remarques :
- Le paramètre dans le registre AX est exprimé en Ko.
- La taille de la mémoire sous la limite de 1 Mo peut être déterminée à l'aide de l'interruption 12h.
- Le contenu des registres BX, CX, DX, SI, DI, BP et des registres de segment n'est pas modifié par cette fonction. Le contenu de tous les autres registres peut avoir été modifié.

Interruption 15h, Fonction 89h BIOS (A partir de AT)
Commutation en mode protégé

Après appel de cette fonction, le processeur 80286 est en mode protégé.

Entrée : AH = 89h

Sortie : aucune

Remarques :
- Cette fonction ne doit être appelée que si vous connaissez bien le fonctionnement du processeur en mode protégé car tout emploi inconsidéré de cette fonction peut aisément entraîner un plantage du système.

Interruption 16h, Fonction 00h BIOS
Clavier : Lire un caractère

Cette fonction lit un caractère dans le buffer clavier. Si aucun caractère n'y est stocké, la fonction attend qu'un caractère ait été entré. Le caractère lu est retiré du buffer clavier.

Entrée : AH = 00h

Sortie : AL = 0 : Code clavier étendu, dans ce cas :
AH = Code clavier étendu différent de 0
Touche normale actionnée, dans ce cas
AL = Code ASCII de la touche
AH = Code clavier de la touche

Remarques :
- Le code ASCII d'un caractère est défini indépendamment de tel ou tel modèle de clavier, alors que le code clavier ne s'applique que sur le type de clavier connecté sur le PC.
- L'annexe J vous fournit une description du jeu de caractères ASCII.
- Le chapitre 7.11 vous fournit une liste des codes clavier étendus.
- Le contenu des registres CX, DX, SI, DI, BP et des registres de segment n'est pas modifié par cette fonction. Le contenu de tous les autres registres peut avoir été modifié.

Interruption 16h, Fonction 01h BIOS
Clavier : Caractère présent ?

Cette fonction détermine si un caractère figure dans le buffer du clavier. Si c'est le cas, elle renvoie cette fonction à la fonction d'appel. Le caractère n'est toutefois pas retiré du buffer clavier, de sorte qu'il peut être à nouveau lu lors d'un appel consécutif de la fonction 1 ou 0. Dans tous les cas, la fonction revient au programme d'appel immédiatement après avoir été appelée.

Entrée : AH = 01h

Sortie : Flag Zéro = 1 : Aucun caractère dans le buffer clavier
Flag Zéro = 0 : Caractère présent, dans ce cas :
AL = 0 : Code clavier étendu, dans ce cas
AH = Code clavier étendu différent de 0
Touche normale actionnée, dans ce cas
AL = Code ASCII de la touche
AH = Code clavier de la touche

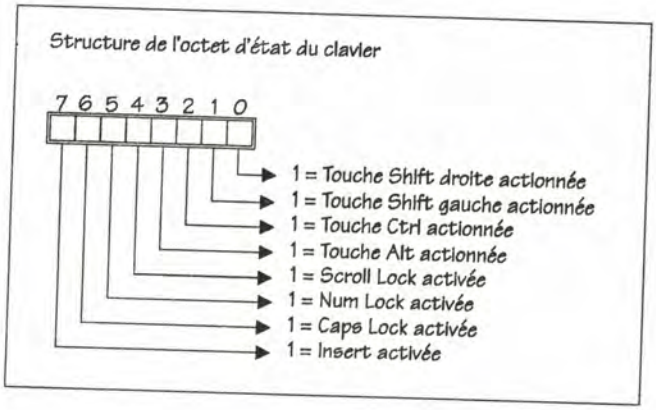
Remarques :
- Le code ASCII d'un caractère est défini indépendamment de tel ou tel modèle de clavier, alors que le code clavier ne s'applique que sur le type de clavier connecté sur le PC.
- L'annexe I vous fournit une description du jeu de caractères ASCII.
- Le chapitre 7.11 vous fournit une liste des codes clavier étendus.
- Le contenu des registres CX, DX, SI, DI, BP et des registres de segment n'est pas modifié par cette fonction. Le contenu de tous les autres registres peut avoir été modifié.

Interruption 16h, Fonction 02h BIOS
Clavier : Tester l'état du clavier

Cette fonction permet de consulter la position de certaines touches de commande ainsi que l'état de différents modes du clavier.

Entrée : AH = 02h

Sortie : AL = Etat du clavier



Remarques :

- Le contenu des registres BX, CX, DX, SI, DI, BP et des registres de segment n'est pas modifié par cette fonction. Le contenu de tous les autres registres peut avoir été modifié.

Interruption 16h, Fonction 03h — BIOS (à partir de AT)
Clavier : Définir le taux de répétition

Cette fonction permet d'agir sur la répétition des touches du clavier pour qu'elles soient répétées automatiquement au bout d'un certain délai.

Entrée :

AH = 03h
AL = 05h
BH = Ralentissement jusqu'à la mise en place de la répétition
BL = Taux de répétition

Sortie : aucune

Remarques :

- Attention ! Cette fonction n'est pas reconnue par tous les BIOS.
- Les valeurs suivantes peuvent être indiquées pour spécifier la durée de ralentissement dans le registre BL :

  00h : 1/4 seconde
  01h : 1/2 seconde
  10h : 1/4 seconde
  11h : 1 seconde

- Les valeurs suivantes peuvent être utilisées pour le taux de répétition dans le registre BL :

Code	RPS*	Code	RPS*	Code	RPS*
1Fh	2,0	17h	4,0	0Fh	8,0
1Eh	2,1	16h	4,3	0Eh	8,6
1Dh	2,3	15h	4,6	0Dh	9,2
1Ch	2,5	14h	5,0	0Ch	10,0
1Bh	2,7	13h	5,5	0Bh	10,9
1Ah	3,0	12h	6,0	0Ah	12,0
19h	3,3	11h	6,7	09h	13,3
18h	3,7	10h	7,5	08h	15,0
* Répétition/Seconde					

- Le contenu des registres BX, CX, DX, SI, DI, BP et des registres de segment n'est pas modifié par cette fonction. Le contenu de tous les autres registres peut avoir été modifié.

Interruption 16h, Fonction 05h BIOS (à partir de AT)
Clavier : Simuler l'appui sur une touche

Cette fonction permet à un programme de simuler l'appui sur une touche en ajoutant un code de touche particulier à la fin actuelle du buffer de clavier.

Entrée : AH = 05h
CH = Scan code de la touche
CL = Code ASCII de la touche

Sortie : AL = 00h:ok
AL = 01h : buffer de clavier saturé

Remarques :
- Attention ! Tous les BIOS ne reconnaissent pas cette fonction.
- Le contenu des registres BX, CX, DX, SI, DI, BP et des registres de segment n'est pas modifié par cette fonction. Le contenu de tous les autres registres peut avoir été modifié.

Interruption 16h, Fonction 10h BIOS (à partir de AT)
Clavier : Test de claviers étendus

Cette fonction est conçue spécialement pour tester la présence de claviers étendus (F11 + F12) avec 101 ou 102 touches. Elle fonctionne comme la fonction 00h, mais ne construit pas les scan codes des touches selon la disposition d'un clavier normal à 84 touches.

Entrée : AH = 10h
AH = scan code de la touche
AL = code ASCII de la touche

Remarques :
- Attention ! Tous les BIOS ne reconnaissent pas cette fonction.
- Le contenu des registres BX, CX, DX, SI, DI, BP et des registres de segment n'est pas modifié par cette fonction. Le contenu de tous les autres registres peut avoir été modifié.

Interruption 16h, Fonction 11h BIOS (à partir de AT)
Clavier : Test de claviers étendus

Cette fonction est conçue spécialement pour tester la présence de claviers étendus (F11 + F12) avec 101 ou 102 touches. Elle fonctionne comme la fonction 00h, mais ne construit pas les scan codes des touches selon la disposition d'un clavier normal à 84 touches.

Entrée : AH = 11h

Sortie : Flag zéro = 1 : pas de caractères dans le buffer de clavier
Flag zéro = 0 : présence de caractères, dans ce cas
AL = 0 : code clavier étendu, dans ce cas
AH = code clavier étendu
AL = différent de 0 : touche normale, dans ce cas
AL = code ASCII de la touche
AH = scan code de la touche

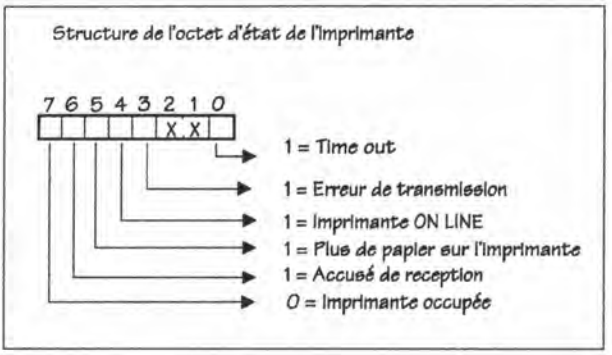
Remarques :
- Attention ! Tous les BIOS ne reconnaissent pas cette fonction.
- Le contenu des registres BX, CX, DX, SI, DI, BP et des registres de segment n'est pas modifié par cette fonction. Le contenu de tous les autres registres peut avoir été modifié.

Interruption 17h, Fonction 00h BIOS
Imprimante (parallèle) : Sortie de caractères

Cette fonction envoie un caractère sur l'une des imprimantes connectées au PC.

Entrée : AH = 00h
AL = Code du caractère à sortir
DX = Numéro d'imprimante

Sortie : AH = Etat de l'imprimante



Remarques :
- La première imprimante connectée sur le PC porte le numéro 0.
- Le contenu des registres BX, CX, DX, SI, DI, BP et des registres de segment n'est pas modifié par cette fonction. Le contenu de tous les autres registres peut avoir été modifié.

Interruption 17h, Fonction 01h BIOS
Imprimante (parallèle) : Initialisation imprimante

Cette fonction initialise une imprimante connectée sur le PC. Il convient d'y recourir avant toute première transmission d'un caractère sur une imprimante donnée.

Entrée : AH = 01h
DX = Numéro d'imprimante

Sortie : AH = Etat de l'imprimante, voir fonction 00h

Remarques :
- La première imprimante connectée sur le PC porte le numéro 0.

- Le contenu des registres BX, CX, DX, SI, DI, BP et des registres de segment n'est pas modifié par cette fonction. Le contenu de tous les autres registres peut avoir été modifié.

Interruption 17h, Fonction 02h BIOS
Imprimante (parallèle) : Tester l'état de l'imprimante

La seule tâche de cette fonction consiste à renvoyer l'état d'une imprimante connectée sur le PC.

Entrée : AH = 02h
DX = Numéro d'imprimante

Sortie : AH = Etat de l'imprimante, voir fonction 00h

Remarques :

- La première imprimante connectée sur le PC porte le numéro 0.
- Le contenu des registres BX, CX, DX, SI, DI, BP et des registres de segment n'est pas modifié par cette fonction. Le contenu de tous les autres registres peut avoir été modifié.

Interruption 18h BASIC en ROM
Appel du BASIC en ROM

Si le PC utilisé dispose d'un BASIC placé en ROM, celui-ci peut être lancé en appelant cette interruption.

Entrée : aucune

Sortie : aucune

Remarques :

- De nombreux modèles récents de PC ne disposent plus d'un BASIC intégré en ROM. Dans ce cas, l'interruption 18h revient immédiatement au programme d'appel ou provoque un plantage du système. Si par contre le PC comporte bien un BASIC intégré en ROM, l'interruption 18h ne rendra pas la main au programme d'appel. Il ne sera alors plus possible de revenir au DOS à moins de déclencher un démarrage à chaud du système (en actionnant ALT-Control-Delete) ou d'éteindre puis de rallumer l'ordinateur.

Interruption 19h BIOS
Lancement du système

L'appel de cette interruption déclenche un lancement du système.

Entrée : aucune

Sortie : aucune

Remarques :

- Il arrive que cette interruption ne relance pas l'ordinateur, mais au contraire provoque un plantage. Cela dépend du fait que le contenu de la mémoire RAM est effacé alors que la table des vecteurs d'interruption reste intacte dans la zone des interruptions 00h à 1Ch. Si un programme TSR atteint l'une de ces interruptions, le prochain appel de cette interruption provoque irrémédiablement un plantage. Cela s'applique surtout à l'interruption 08h qui est utilisée par la plupart des programmes TSR.
- Par conséquent, évitez d'utiliser cette interruption et adoptez plutôt la technique suivante pour relancer le système : pour effectuer un démarrage à chaud, sauvegardez d'abord la valeur 1234h dans la cellule 0040:0072 de la mémoire et exécutez ensuite un FAR-jump vers la cellule FFFF:0000.
- Cette méthode permet en outre de réaliser un démarrage à froid à condition de charger à cet effet la valeur 0000h dans la cellule 0040:0072.

Interruption 1Ah, Fonction 00h BIOS
Date et heure : Lire le compteur horaire

Cette fonction permet de lire le contenu du compteur horaire. Il est incrémenté 18,2 fois par seconde et permet de calculer le temps écoulé depuis la mise en marche de l'ordinateur ou depuis minuit.

Entrée : AH = 00h

Sortie : CX = Mot fort du compteur horaire
DX = Mot faible du compteur horaire
AL = 0 : Moins de 24 heures se sont écoulées depuis la dernière fois que l'heure a été lue.
Différent de 0 : Plus de 24 heures se sont écoulées depuis la dernière fois que l'heure a été lue.

Remarques :
- L'AT, qui dispose d'une horloge sur piles, fixe le compteur horaire sur l'heure réelle lors du lancement de l'ordinateur. Les PC qui ne disposent pas d'horloge en temps réel fixent ce compteur sur 0 lors du lancement.
- Le contenu des registres BX, SI, DI, BP et des registres de segment n'est pas modifié par cette fonction. Le contenu de tous les autres registres peut avoir été modifié.

Interruption 1Ah, Fonction 01h BIOS
Date et heure : Fixer le compteur horaire

Cette fonction permet de fixer le contenu du compteur horaire. Ce compteur étant augmenté 18,2 fois par seconde, il permet de connaître le temps écoulé depuis la mise en marche de l'ordinateur ou depuis minuit, à moins que cette fonction ne soit utilisée pour le régler.

Entrée : AH = 01h
CX = Mot fort du compteur horaire
DX = Mot faible du compteur horaire

Sortie : aucune

Remarques :
- L'AT, qui dispose d'une horloge sur piles, fixe le compteur horaire sur l'heure réelle lors du lancement de l'ordinateur. Les PC qui ne disposent pas d'horloge en temps réel fixent ce compteur sur 0 lors du lancement. C'est pourquoi, sur ces modèles, il convient d'utiliser cette fonction si l'on veut que le compteur horaire soit réglé sur l'heure actuelle.
- Le contenu des registres AX, BX, CX, DX, SI, DI, BP et des registres de segment n'est pas modifié par cette fonction. Le contenu de tous les autres registres peut avoir été modifié.

Interruption 1Ah, Fonction 02h BIOS (A partir de AT)
Date et heure : Lecture de l'horloge en temps réel

Cette fonction permet de lire l'heure sur l'horloge en temps réel sur piles. Comme ce type d'horloge n'existe que sur les AT, seul ce modèle de PC soutient cette fonction.

Entrée : AH = 02h

Sortie : Flag Carry =0 : Tout va bien, dans ce cas
CH = Heures

CL = Minutes
DH = Secondes
Flag Carry =1 : Les piles de l'horloge sont vides

Remarques :
- Toutes les indications sont fournies en format BCD.
- Le contenu des registres BX, SI, DI, BP et des registres de segment n'est pas modifié par cette fonction. Le contenu de tous les autres registres peut avoir été modifié.

Interruption 1Ah, Fonction 03h BIOS (A partir de AT)
Date et heure : Régler l'horloge en temps réel

Cette fonction permet de fixer l'heure sur l'horloge en temps réel sur piles. Comme ce type d'horloge n'existe que sur les AT, seul ce modèle de PC soutient cette fonction.

Entrée :
AH = 03h
CH = Heures
CL = Minutes
DH = Secondes
DL = 1 : c'est l'heure d'été
DL = 0 : ce n'est pas l'heure d'été

Sortie : aucune

Remarques :
- Les indications d'heures, minutes et secondes doivent être fournies en format BCD.
- Le contenu des registres BX, SI, DI, BP et des registres de segment n'est pas modifié par cette fonction. Le contenu de tous les autres registres peut avoir été modifié.

Interruption 1Ah, Fonction 04h BIOS (A partir de AT)
Date et heure : Lecture de la date sur l'horloge en temps réel

Cette fonction permet de lire la date qui est stockée dans la mémoire RAM de l'horloge en temps réel sur piles. Cette fonction n'est soutenue que par l'AT.

Entrée : AH = 04h

Sortie : Flag Carry =0 : Tout va bien, dans ce cas
CH = Siècle (19 ou 20)
CL = Année
DH = Mois

DL = Jour
Flag Carry =1 : Les piles de l'horloge sont vides

Remarques :
- Toutes les indications renvoyées par la fonction sont fournies en format BCD.
- Le contenu des registres BX, SI, DI, BP et des registres de segment n'est pas modifié par cette fonction. Le contenu de tous les autres registres peut avoir été modifié.

Interruption 1Ah, Fonction 05h BIOS (A partir de AT)
Date et heure : Fixer la date sur l'horloge en temps réel

Cette fonction permet de fixer la date qui est stockée dans la mémoire RAM de l'horloge en temps réel sur piles. Cette fonction n'est soutenue que par l'AT qui dispose seul d'une horloge en temps réel.

Entrée :
AH = 05h
CH = Siècle (19 ou 20)
CL = Année
DH = Mois
DL = Jour

Sortie : aucune

Remarques :
- Toutes les indications fournies à la fonction doivent être codées en format BCD.
- Le contenu des registres BX, CX, SI, DI, BP et des registres de segment n'est pas modifié par cette fonction. Le contenu de tous les autres registres peut avoir été modifié.

Interruption 1Ah, Fonction 06h BIOS (A partir de AT)
Date et heure : Fixer l'heure d'alarme

Cette fonction permet de fixer une heure d'alarme se référant au jour actuel. Lorsque cette heure d'alarme est atteinte, l'interruption 4Ah est déclenchée. L'horloge en temps réel sur piles n'existe que sur les AT.

Entrée :
AH = 06h
CH = Heures
CL = Minutes
DH = Secondes

Sortie : Flag Carry =0 : Tout va bien
Flag Carry =1 : soit les piles de l'horloge sont vides, soit une autre heure d'alarme a déjà été programmée.

Remarques :
- Les indications fournies à la fonction doivent être codées en format BCD.
- L'interruption 4Ah est dirigée, lors du lancement du système, sur une instruction langage machine IRET. Si cette interruption n'est donc pas redirigée sur une routine spéciale, il ne se passera rien lorsque l'heure d'alarme sera atteinte.
- Il n'est pas possible d'activer plus d'une heure d'alarme à la fois. Si une heure d'alarme est déjà programmée, il faut d'abord l'annuler à l'aide de la fonction 7.
- Le contenu des registres BX, CX, SI, DI, BP et des registres de segment n'est pas modifié par cette fonction. Le contenu de tous les autres registres peut avoir été modifié.

Interruption 1Ah, Fonction 07h BIOS (A partir de AT)
Date et heure : Annuler l'heure d'alarme

Cette fonction permet d'annuler une heure d'alarme déjà programmée. Aucune alarme ne sera donc plus déclenchée lorsque cette heure sera atteinte (seulement AT).

Entrée : AH = 07h

Sortie : aucune

Remarques :
- Cette fonction doit notamment être appelée chaque fois que l'on veut modifier l'heure d'alarme. Dans ce cas en effet, il ne sera possible de programmer une nouvelle heure d'alarme, en appelant la fonction 6, qu'après avoir appelé cette fonction.
- Le contenu des registres BX, CX, SI, DI, BP et des registres de segment n'est pas modifié par cette fonction. Le contenu de tous les autres registres peut avoir été modifié.

B. Description des fonctions du BIOS EGA/VGA

Interruption 10h, Fonction 00h BIOS EGA/VGA
Ecran : Fixer le mode vidéo

Cette fonction permet de sélectionner et initialiser un mode vidéo.

Entrée : AH = 00h
AL = mode vidéo EGA/VGA
- 00h: 40*25 caractères texte, 16 couleurs, mais pas d'affichage en couleur (EGA/VGA sur un moniteur couleur)
- 01h: 40*25 caractères texte, 16 couleurs, mais pas d'affichage en couleur (EGA/VGA sur un moniteur couleur)
- 02h: 80*25 caractères texte, 16 couleurs, mais pas d'affichage en couleur (EGA/VGA sur un moniteur couleur)
- 03h: 80*25 caractères texte, 16 couleurs (EGA/VGA sur un moniteur couleur)
- 04h: 320*200 points graphiques, 4 couleurs, mais pas d'affichage en couleur (EGA/VGA sur un moniteur couleur)
- 05h: 320*200 points graphiques, 4 couleurs (EGA/VGA sur un moniteur couleur)
- 06h: 640*200 points graphiques, 2 couleurs, mais pas d'affichage en couleur (EGA/VGA sur un moniteur couleur)
- 07h: 80*25 caractères texte, monochrome (EGA/VGA sur un moniteur monochrome)
- 0Dh: 320*200 points graphiques, 16 couleurs (EGA/VGA sur un moniteur couleur)
- 0Eh: 640*200 points graphiques, 16 couleurs (EGA/VGA sur un moniteur couleur)
- 0Fh: 640*350 points graphiques, monochrome (EGA/VGA sur un moniteur monochrome)
- 10h: 640*350 points graphiques, 4 couleurs (EGA/VGA avec 64 Ko sur un moniteur haute résolution)
  640*350 points graphiques, 16 couleurs (EGA/VGA avec au moins 128 Ko sur un moniteur haute résolution)
- 11h: 640*480 points graphiques, 2 couleurs (VGA uniquement)
- 12h: 640*480 points graphiques, 16 couleurs (VGA uniquement)
- 13h: 320*200 points graphiques, 256 couleurs (VGA uniquement)

Sortie : pas de sortie

Remarques :

- Les modes 0 et 1, 2 et 3 ainsi que 4 et 5 se distinguent normalement par le fait que sous le premier mode de chaque paire indiquée, aucun signal de couleur n'est envoyé au moniteur. Comme ce n'est cependant pas possible avec la carte EGA, les modes 0 et 1, 2 et 3 ainsi que 4 et 5 sont donc identiques.
- Si le bit 7 du registre AL est mis lors de l'appel de cette fonction, le contenu de la RAM vidéo n'est pas effacé lors du passage au nouveau mode vidéo.
- Cette fonction sert à programmer le contrôleur vidéo de la carte EGA/VGA et à définir une palette de couleurs. Les autres tâches de cette fonction peuvent en outre être définies à l'aide de la fonction 12h.
- Le contenu des registres BX, CX, DX, SI, DI et BP ainsi que celui de tous les registres de segment n'est pas modifié par cette fonction.

Interruption 10h, Fonction 01h BIOS EGA/VGA
Ecran : Définition de l'apparence du curseur

Cette fonction permet de fixer les lignes de départ et de fin du curseur clignotant de l'écran. Elle est indépendante de la page écran affichée sur l'écran.

Entrée : AH = 01h
CH = ligne de départ du curseur
CL = ligne de fin du curseur

Sortie : pas de sortie

Remarques :

- Comme les réglages possibles avec la carte EGA/VGA dépendent de la taille de la matrice de caractère sous le mode vidéo actuel, les spécifications dans les registres CH et CL se réfèrent ici toujours à une matrice de caractère composée de 8 lignes. Les valeurs spécifiées doivent donc être comprises entre 0 et 7. Le BIOS EGA/VGA adapte ces spécifications à la taille actuelle de la matrice de caractère en les convertissant donc d'une matrice de 8 lignes vers la hauteur actuelle de la matrice.
- Le contenu des registres BX, CX, DX, SI, DI et BP ainsi que celui de tous les registres de segment n'est pas modifié par cette fonction.

Interruption 10h, Fonction 02h BIOS EGA/VGA
Ecran : Positionnement du curseur

Cette fonction permet de fixer le curseur, qui fixe la position de l'écran que les fonctions BIOS pour la sortie de caractères doivent employer.

Entrée : AH = 02h
BH = numéro de page écran
DH = ligne de l'écran
DL = colonne de l'écran

Sortie : pas de sortie

Remarques :
- Le curseur clignotant de l'écran n'est déplacé par cette fonction que si la page écran appelée est la page écran actuellement affichée.
- Les valeurs pour la ligne de l'écran et la colonne de l'écran dépendent de la résolution sous le mode vidéo actuel.
- Une méthode pour faire disparaître le curseur qui clignote est de le mettre à une position écran non existante (par exemple colonne 0, ligne 25).
- Le numéro de la page dépendante du nombre de pages écran de la carte vidéo à votre disposition.
- Le contenu des registres BX, CX, DX, SI, DI et BP ainsi que celui de tous les registres de segment n'est pas modifié par cette fonction.

Interruption 10h, Fonction 03h BIOS EGA/VGA
Ecran : Lecture de la position du curseur

Cette fonction renvoie la position du curseur de texte dans une page écran et les lignes de départ et de fin du curseur clignotant de l'écran..

Entrée : AH = 03h
BH = numéro de page écran

Sortie : DH = ligne de l'écran dans laquelle figure le curseur
DL = colonne de l'écran dans laquelle figure le curseur
CH = ligne de départ du curseur clignotant de l'écran
CL = ligne de fin du curseur clignotant de l'écran

Remarques :
- les ligne et colonne de l'écran se réfèrent toujours au système de coordonnées de texte, même lorsque c'est un mode graphique qui est activé.

- Les lignes de départ et de fin du curseur ne sont fournies de façon exacte que dans le cadre des modes de texte. Dans les modes graphiques, ces indications sont dépourvues de signification.
- Le contenu des registres BX, SI, DI et BP ainsi que celui de tous les registres de segment n'est pas modifié par cette fonction.

Interruption 10h, Fonction 04h EGA
crayon optique : Lire la position

La position d'un crayon optique sur l'écran est lue si cela est possible. Cela ne concerne principalement que les cartes EGA parce que les cartes VGA n'utilisent pas un crayon optique.

Entrée : AH = 04h

Sortie : AH = 0 : La position du crayon optique ne peut pas être obtenue dans l'immédiat
AH = 1 : Position du crayon optique demandée, dans ce cas,
DH = Ligne de l'écran du crayon optique (mode texte)
DL = Colonne de l'écran du crayon optique (mode texte)
CH = Ligne de l'écran du crayon optique (mode graphique)
BX = Colonne de l'écran du crayon optique (mode graphique)

Remarques :
- L'appel de cette fonction doit se répéter tant que la valeur 1 n'est pas retournée dans le registre AH. La position du crayon optique n'est transmise qu'à ce moment-là.
- Il se peut que les coordonnées du crayon optique ne soient pas rendues correctement surtout dans le mode graphique. La coordonnée Y (ligne) est toujours un multiple de 2 et il est difficile de distinguer si le crayon optique se trouve dans la ligne 8 ou 9.

  En mode graphique 320*200 points, la coordonnée X (colonne) est toujours un multiple de 4 et en mode 640*200 points, un multiple de 8.
- Le contenu du registre CL et le contenu des registres de segment SS, CS et DS ne sont pas modifiés par cette fonction. Le contenu de tous les autres registres, en particulier SI et DI, peuvent être modifiés.

Interruption 10h, Fonction 05h BIOS EGA/VGA
Ecran : Sélection de la page écran actuelle

Sélection de la page écran actuelle, c'est-à-dire de la page écran devant être affichée sur l'écran (seulement en mode de texte)

Entrée : AH = 05h
AL = numéro de page écran

Sortie : pas de sortie

Remarques :
- Le nombre de pages écran disponibles dépend de la taille de la mémoire RAM installée sur la carte EGA/VGA.
- Lors du passage à une autre page écran, le curseur clignotant de l'écran est fixé sur la position du curseur de texte dans cette page.
- La commutation entre différentes pages écran ne modifie pas leur contenu.
- Le contenu des registres BX, CX, DX, SI, DI et BP ainsi que celui de tous les registres de segment n'est pas modifié par cette fonction.

Interruption 10h, Fonction 06h BIOS EGA/VGA
Ecran : Faire défiler des lignes de texte vers le haut (scrolling)

Une partie de la page écran actuelle est décalée vers le haut d'une ou plusieurs lignes ou bien vidée.

Entrée : AH = 06h
AL = nombre de lignes sur lesquelles la fenêtre doit être décalée vers le haut (0 signifie vider la fenêtre)
CH = ligne de l'écran du coin supérieur gauche de la fenêtre
CL = colonne de l'écran du coin supérieur gauche de la fenêtre
DH = ligne de l'écran du coin inférieur droit de la fenêtre
DL = colonne de l'écran du coin inférieur droit de la fenêtre
BH = couleur (attribut) pour la ou les lignes vides

Sortie : pas de sortie

Remarques :
- C'est normalement le contenu de la page écran actuelle qui est décalé mais lorsque cette fonction est appelée en mode graphique 320*200 points avec 4 couleurs, c'est toujours la page vidéo 0 qui est affectée.

- Le fait de vider cette zone de l'écran (nombre de lignes = 0) revient en fait à la remplir d'espaces (code ASCII 32).
- Le contenu des lignes expulsées de la fenêtre est irrémédiablement perdu.
- Pour vider l'écran en totalité, il est préférable d'avoir recours à la fonction 0 de cette interruption.
- L'interpretation de l'octet d'attribut dépend du mode vidéo actuel. En mode de texte, il est interprété comme n'importe quel autre octet d'attribut dans la RAM vidéo. En mode graphique 640*200 points avec 2 couleurs, cet octet représente par contre chaque fois les codes couleur pour 8 points consécutifs. En mode 320*200 points avec 4 couleurs, il représente les codes couleur pour 4 points consécutifs chaque fois. Sous tous les autres modes graphiques, il indique la couleur pour un point graphique qui devra être attribuée à tous les points graphiques dans la zone de l'écran effacée.
- Le contenu des registres BX, CX, DX, SI, DI et BP ainsi que celui de tous les registres de segment n'est pas modifié par cette fonction.

Interruption 10h, Fonction 07h BIOS EGA/VGA
Ecran : Faire défiler des lignes de texte vers le bas (scrolling)

Une partie de la page écran actuelle est décalée vers le bas d'une ou plusieurs lignes ou bien vidée.

Entrée :
- AH = 07h
- AL = nombre de lignes sur lesquelles la fenêtre doit être décalée vers le bas (0 signifie vider la fenêtre)
- CH = ligne de l'écran du coin supérieur gauche de la fenêtre
- CL = colonne de l'écran du coin supérieur gauche de la fenêtre
- DH = ligne de l'écran du coin inférieur droit de la fenêtre
- DL = colonne de l'écran du coin inférieur droit de la fenêtre
- BH = couleur (attribut) pour la ou les lignes vides

Sortie : pas de sortie

Remarques :
- C'est normalement le contenu de la page écran actuelle qui est décalé mais lorsque cette fonction est appelée en mode graphique 320*200 points avec 4 couleurs, c'est toujours la page vidéo 0 qui est affectée.
- Le fait de vider cette zone de l'écran (nombre de lignes = 0) revient en fait à la remplir d'espaces (code ASCII 32).

- Le contenu des lignes expulsées de la fenêtre est irrémédiablement perdu.
- Pour vider l'écran en totalité, il est préférable d'avoir recours à la fonction 0 de cette interruption.
- L'interpretation de l'octet d'attribut dépend du mode vidéo actuel. En mode de texte, il est interprété comme n'importe quel autre octet d'attribut dans la RAM vidéo. En mode graphique 640*200 points avec 2 couleurs, cet octet représente par contre chaque fois les codes couleur pour 8 points consécutifs. En mode 320*200 points avec 4 couleurs, il représente les codes couleur pour 4 points consécutifs chaque fois. Sous tous les autres modes graphiques, il indique la couleur pour un point graphique qui devra être attribuée à tous les points graphiques dans la zone de l'écran effacée.
- Le contenu des registres BX, CX, DX, SI, DI et BP ainsi que celui de tous les registres de segment n'est pas modifié par cette fonction.

Interruption 10h, Fonction 08h BIOS EGA/VGA
Ecran : Lecture d'un caractère et de sa couleur

Cette fonction renvoie le code ASCII du caractère dans la position actuelle du curseur ainsi que sa couleur (son attribut).

Entrée : AH = 08h
BH = numéro de page écran

Sortie : AL = code ASCII du caractère
AH = couleur (attribut)

Remarques :
- Cette fonction peut également être appelée en mode graphique, auquel cas le motif de bits du caractère sur l'écran sera comparé avec les motifs de bits des caractères. Si le caractère ne peut être identifié de cette manière, le registre AL contiendra la valeur 0 après appel de la fonction.
- En mode graphique 320*200 points avec 4 couleurs, cette fonction ne travaille correctement que lorsqu'on accède à la page écran 0.
- Le contenu des registres BX, CX, DX, SI, DI et BP ainsi que celui de tous les registres de segment n'est pas modifié par cette fonction.

Interruption 10h, Fonction 09h BIOS EGA/VGA
Ecran : Ecriture d'un caractère et d'une couleur

Un caractère est écrit avec une couleur déterminée dans la position actuelle du curseur (dans la page écran spécifiée).

Entrée : AH = 09h
BH = numéro de page écran
CX = coefficient de répétition
AL = code ASCII du caractère
BL = attribut

Sortie : pas de sortie

Remarques :

- Si le caractère spécifié doit être sorti plusieurs fois (c'est-à-dire que le registre CX est supérieur à 1), il faut, en mode graphique, que tous les caractères puissent tenir dans la ligne d'écran actuelle.
- En mode graphique 320*200 points avec 4 couleurs, cette fonction ne travaille correctement que lorsqu'on accède à la page écran 0.
- Dans le cadre d'un mode graphique, l'attribut dans le registre BL indique la couleur de premier plan du caractère, la couleur de fond étant toujours 0. Si le bit 7 est mis, le caractère est combiné par un XOR avec le motif de points écran actuel dans la position de sortie.
- Les codes de contrôle Bell, Carriage Return etc. ne sont pas identifiés comme tels mais sortis comme des caractères ASCII ordinaires.
- Cette fonction permet aussi de sortir des caractères en mode graphique, les motifs des caractères étant tirés de l'une des tables de caractères EGA/VGA.
- Cette fonction ne déplace pas le curseur vers la prochaine position de l'écran.
- Le contenu des registres BX, CX, DX, SI, DI et BP ainsi que celui de tous les registres de segment n'est pas modifié par cette fonction.

Interruption 10h, Fonction 0Ah BIOS EGA/VGA
Ecran : Ecriture d'un caractère

Un caractère est écrit dans la position actuelle du curseur dans la page écran spécifiée, la couleur du caractère qui figurait auparavant dans cette position de l'écran restant inchangée.

Entrée : AH = 0Ah
BH = numéro de page écran
BL = couleur de premier plan du caractère pour les modes graphiques
CX = coefficient de répétition
AL = code ASCII du caractère

Sortie : pas de sortie

Remarques :
- Si le caractère spécifié doit être sorti plusieurs fois (c'est-à-dire que le registre CX est supérieur à 1), il faut, en mode graphique, que tous les caractères puissent tenir dans la ligne d'écran actuelle.
- Les codes de contrôle Bell, Carriage Return etc. ne sont pas identifiés comme tels mais sortis comme des caractères ASCII ordinaires.
- Cette fonction permet aussi de sortir des caractères en mode graphique, les motifs des caractères étant tirés de l'une des tables de caractères EGA.
- Dans le cadre d'un mode graphique, l'attribut dans le registre BL indique la couleur de premier plan du caractère, la couleur de fond étant toujours 0. Si le bit 7 est mis, le caractère est combiné par un XOR avec le motif de points écran actuel dans la position de sortie.
- Cette fonction ne déplace pas le curseur vers la prochaine position de l'écran.
- Le contenu des registres BX, CX, DX, SI, DI et BP ainsi que celui de tous les registres de segment n'est pas modifié par cette fonction.

Interruption 10h, Fonction 0Bh, Sous-fonction 0 BIOS EGA/VGA
Ecran : Sélection de la couleur du cadre et du fond

Cette fonction permet de sélectionner la couleur du cadre et du fond pour les modes graphique ou texte.

Entrée : AH = 0Bh
BH = 0
BL = couleur de cadre et de fond

Sortie : pas de sortie

Remarques :
- Cette fonction ne doit être appelée que si la carte EGA/VGA est en mode graphique 320*200 points ou 640*200 points. Dans tous les autres modes, il est préférable d'utiliser plutôt la fonction 10h.

- Les bits 0 à 3 du registre BL définissent la couleur du fond et du cadre. On peut en outre obtenir des couleurs plus intenses en fixant le bit 4.
- Le contenu des registres BX, CX, DX, SI, DI et BP ainsi que celui de tous les registres de segment n'est pas modifié par cette fonction.

Interruption 10h, Fonction 0Bh, Sous-fonction 1 BIOS EGA/VGA
Ecran : sélection de la palette de couleurs

Sélection de l'une des palettes de couleurs pour le mode graphique 320*200 points.

Entrée : AH = 0Bh
BH = 1
BL = numéro de la palette de couleur

Sortie : pas de sortie

Remarques :
- Cette fonction ne doit être appelée que si la carte EGA/VGA est en mode graphique 320*200 points ou 640*200 points. Dans tous les autres modes, il est préférable d'utiliser plutôt la fonction 10h.
- Le BIOS EGA/VGA émule les deux palettes de couleurs CGA avec les numéros 0 et 1. Elles comportent les couleurs suivantes :
  Palette 0 : vert, rouge, jaune
  Palette 1 : cyan, magenta, blanc
- Le contenu des registres BX, CX, DX, SI, DI et BP ainsi que celui de tous les registres de segment n'est pas modifié par cette fonction.

Interruption 10h, Fonction 0Ch BIOS EGA/VGA
Ecran : Ecrire un point graphique

Fixe le code couleur d'un point écran en mode graphique.

Entrée : AH = 0Ch
BH = page écran
DX = ligne de l'écran
CX = colonne de l'écran
AL = code couleur

Sortie : pas de sortie

Remarques :
- Le code couleur dépend du nombre de couleur affichées sous le mode graphique actuel.

- Si le bit 7 du registre AL est mis, le code couleur sera combiné par un XOR avec le code couleur antérieur des points graphiques.
- L'indication de la page écran est ignorée en mode graphique 320*200 points avec 4 couleurs.
- Le contenu des registres BX, CX, DX, SI, DI et BP ainsi que celui de tous les registres de segment n'est pas modifié par cette fonction.

Interruption 10h, Fonction 0Dh BIOS EGA/VGA
Ecran : Lire un point graphique

Lecture du code couleur d'un point écran en mode graphique.

Entrée : AH = 0Dh
BH = page écran
DX = ligne de l'écran
CX = colonne de l'écran

Sortie : AL = code couleur

Remarques :
- Le code couleur dépend du nombre de couleur affichées sous le mode graphique actuel.
- L'indication de la page écran est ignorée en mode graphique 320*200 points avec 4 couleurs.
- Le contenu des registres BX, CX, DX, SI, DI et BP ainsi que celui de tous les registres de segment n'est pas modifié par cette fonction.

Interruption 10h, Fonction 0Eh BIOS EGA/VGA
Ecran : Ecriture d'un caractère

Un caractère est écrit dans la position actuelle du curseur dans la page écran actuelle, la couleur du caractère qui figurait auparavant dans cette position de l'écran étant conservée.

Entrée : AH = 0Eh
AL = code ASCII du caractère
BL = couleur de premier plan du caractère pour le mode graphique

Sortie : pas de sortie

Remarques :
- Cette fonction n'interprète pas les différents codes de contrôle tels que Bell et Carriage Return comme des codes ASCII ordinaires mais

bien comme des codes de contrôle. Un bip sera donc émis par exemple au lieu d'afficher le caractère Bell.

- Après qu'un caractère ait été sorti avec cette fonction, la position du curseur est incrémentée de façon à ce que le caractère suivant soit sorti sur la prochaine position de l'écran. Si la dernière position de l'écran est atteinte, l'écran est décalé d'une ligne vers le haut et la sortie se poursuit dans la première colonne de la dernière ligne de l'écran.
- Si le bit 7 du registre AL est mis, le code couleur sera combiné par un XOR avec le code couleur antérieur des points graphiques. La couleur de fond des caractères est toujours 0.
- Cette fonction permet aussi de sortir des caractères en mode graphique, les motifs des caractères étant tirés de l'une des tables de caractères EGA.
- Le contenu des registres BX, CX, DX, SI, DI et BP ainsi que celui de tous les registres de segment n'est pas modifié par cette fonction.

Interruption 10h, Fonction 0Fh BIOS EGA/VGA
Ecran : lecture du mode vidéo

Lecture du mode vidéo actuel, du nombre de caractères par ligne et du numéro de la page écran actuelle.

Entrée : AH = 0Fh

Sortie : AL = Mode vidéo en cours (cf. Fonction 00h)
AH = Nombre de caractères par ligne
BH = Numéro de la page écran en cours

Remarques :
- Le contenu des registres BX, CX, DX, SI, DI et BP ainsi que celui de tous les registres de segment n'est pas modifié par cette fonction.

Interruption 10h, Fonction 10h, Sous-fonction 00h BIOS EGA/VGA
Ecran : Fixer le registre de palette

Cette fonction permet de fixer le contenu de l'un des registres de palette du contrôleur d'attribut de la carte EGA/VGA.

Entrée : AH = 10h
AL = 00h

BL = code couleur
BH = registre à adresser

Sortie : aucune

Remarques :
- Le numéro de registre n'étant pas contrôlé par le BIOS, cette fonction permet de programmer aussi les autres registres du contrôleur d'attribut. Il s'agit notamment des registres Mode Control, Overscan etc.
- Le contenu des registres BX, CX, DX, SI, DI et BP ainsi que celui de tous les registres de segment n'est pas modifié par cette fonction.

Interruption 10h, Fonction 10h, Sous-fonction 01h BIOS EGA/VGA
Ecran : Fixer la couleur du cadre de l'écran

La valeur spécifiée est copiée par cette fonction dans le registre Overscan du contrôleur d'attribut EGA/VGA.

Entrée : AH = 10h
AL = 01h
BH = couleur du cadre

Sortie : aucune

Remarques :
- Le contenu des registres BX, CX, DX, SI, DI et BP ainsi que celui de tous les registres de segment n'est pas modifié par cette fonction.

Interruption 10h, Fonction 10h, Sous-fonction 02h BIOS EGA/VGA
Ecran : Fixer tous les registres de palette

Ces fonctions permettent de fixer simultanément les 16 registres de palette et le registre Overscan.

Entrée : AH = 10h
AL = 02h
ES = adresse de segment de la table de couleurs
DX = adresse d'offset de la table de couleurs

Sortie : aucune

Remarques :
- La paire de registres ES:BX désigne une table qui doit avoir une longueur de 17 octets. Les 16 premiers octets seront transférés

dans les 16 registres de palette du contrôleur d'attribut alors que le 17ème registre sera copié dans le registre Overscan.

- Le contenu des registres BX, CX, DX, SI, DI et BP ainsi que celui de tous les registres de segment n'est pas modifié par cette fonction.

Interruption 10h, Fonction 10h, Sous-fonction 03h BIOS EGA/VGA
Ecran : Fixer attribut de clignotement

L'appel de cette fonction permet de savoir si le bit 7 de l'octet d'attribut d'un caractère entraîne, en mode de texte, un clignotement de ce caractère, ou un affichage plus intense du fond du caractère.

Entrée : AH = 10h
AL = 03h
BL = attribut de clignotement
0 = couleur de fond intense
1 = clignotement

Sortie : aucune

Remarques :
- Le contenu des registres BX, CX, DX, SI, DI et BP ainsi que celui de tous les registres de segment n'est pas modifié par cette fonction.

Interruption 10h, Fonction 10h, Sous-fonction 07h VGA
Lire le registre de palette

Cette fonction offre à un programme la possibilité de connaître le contenu de l'un des registres de palette du contrôleur d'attribut.

Entrée : AH = 10h
AL = 07h
BL = Numéro du registre de palette

Sortie : BH = Contenu du registre de palette adressé

Remarques :
- Le numéro du registre de palette appelé n'étant pas contrôlé par le BIOS, cette fonction permet en fait de lire tous les registres du contrôleur d'attribut.
- Le contenu des registres BL, CX, DX, SI, DI et BP ainsi que le contenu de tous les registres de segment ne sont pas modifiés par cette fonction.

Interruption 10h, Fonction 10h, Sous-fonction 08h VGA
Lire le contenu du registre Overscan

Cette fonction renvoie au programme d'appel le contenu du registre Overscan, qui définit la couleur du cadre de l'écran.

Entrée : AH = 10h
AL = 08h

Sortie : BH = Contenu du registre Overscan

Remarques :
- Le contenu des registres BL, CX, DX, SI, DI et BP ainsi que le contenu de tous les registres de segment ne sont pas modifiés par cette fonction.

Interruption 10h, Fonction 10h, Sous-fonction 09h VGA
Lire le contenu de tous les registres de palette et le registre Overscan

Lorsque cette fonction est appelée, le contenu des 16 registres de palette ainsi que le contenu du registre Overscan sont copiés dans un buffer du programme d'appel.

Entrée : AH = 10h
AL = 09h
ES = Adresse de segment du buffer
DX = Adresse d'offset du buffer

Sortie : Aucune

Remarques :
- Le buffer doit disposer d'une place d'au moins 17 octets pour pouvoir recevoir le contenu de tous les registres de palette (octets 0 à 15) et le contenu du registre Overscan (octet 16).
- Le contenu des registres BX, CX, DX, SI, DI et BP ainsi que le contenu de tous les registres de segment ne sont pas modifiés par cette fonction.

Interruption 10h, Fonction 10h, Sous-fonction 10h VGA
Charger un registre de couleur DAC

Cette fonction permet de définir le contenu de l'un des 256 registres de couleurs DAC.

Entrée : AH = 10h
AL = 10h

BX = Numéro du registre de couleur DAC (0 à 255)
CH = Vert
CL = Bleu
DH = Rouge

Sortie : Aucune

Remarques :
- Seuls les bits 0 à 5 des registres CH, CL et DH servent à composer le mélange de la couleur, les autres bits sont ignorés.
- Le contenu des registres BX, CX, DX, SI, DI et BP ainsi que le contenu de tous les registres de segment ne sont pas modifiés par cette fonction.

Interruption 10h, Fonction 10h, Sous-fonction 12h VGA
Charger plusieurs registres de couleur DAC

Cette fonction représente une extension de la fonction 10h et permet de charger des valeurs dans plusieurs registres de couleur DAC simultanément.

Entrée : AH = 10h
AL = 12h
BX = Numéro du premier registre de couleur DAC appelé (0 à 255)
CX = Nombre de registres à définir
ES = Adresse de segment du buffer
DX = Adresse d'offset du buffer

Sortie : Aucune

Remarques :
- Le buffer spécifié doit comporter pour chacun des registres de couleur à définir un groupe de 3 octets consécutifs, dont le premier définit la part de rouge, le deuxième la part de vert et le troisième la part de bleu. Les trois premiers octets correspondent au premier registre de couleur DAC appelé, les trois octets suivants au registre de couleur DAC suivant, etc.
- Pour le mélange de la couleur, seuls les bits 0 à 5 sont significatifs, les autres bits sont ignorés.
- Si la somme de BX et CX est supérieure à 255, c'est le premier registre de couleur DAC qui sera traité après que le dernier registre de couleur DAC ait été traité. Il y a donc une exécution en boucle (Wrap Around).
- Le contenu des registres BX, CX, DX, SI, DI et BP ainsi que le contenu de tous les registres de segment ne sont pas modifiés par cette fonction.

Interruption 10h, Fonction 10h, Sous-fonction 13h VGA
Fixer la méthode de sélection des couleurs ou sélectionner un groupe de registres DAC

Cette fonction manipule le bit 7 du registre de contrôle de mode.

Entrée : AH = 10h
AL = 13h
BL = 00h ou 01h (voir plus bas)
BH = voir plus bas

Sortie : Aucune

Remarques :

- Cette sous-fonction dispose elle-même à son tour de deux sous-fonctions sélectionnées à travers la valeur dans le registre BL. La sous-fonction 00h permet de fixer la sélection des couleurs, alors que la sous-fonction 01h sert à sélectionner le groupe de registres DAC activé.
- La sous-fonction 00h copie le bit 0 du registre BH dans le bit 7 du registre de contrôle de mode, fixant ainsi la méthode de sélection des couleurs. Si le bit 0 de BH contient la valeur 0, les 256 registres de couleur DAC seront divisés en quatre groupes de 64 registres. En ce qui concerne la sélection de couleur, les bits 0 à 5 de chaque registre de palette forment avec les bits 2 à 3 du registre de sélection de couleur les 8 bits utilisés comme index dans la table de couleurs DAC. Si le bit 0 du registre BH contient par contre la valeur 1, les registres de couleur DAC seront groupés en 16 groupes de 16 registres. Les 4 bits inférieurs du registre de palette composeront alors avec les 4 bits inférieurs du registre de sélection de couleur l'index 8 bits sur la table de couleurs DAC.
- La sous-fonction 01h sert à définir le registre de sélection de couleur, dont le contenu sélectionne le groupe de registres de couleur DAC activé. Le contenu du registre BH est copié à cet effet dans le registre de sélection de couleur.
- Le contenu des registres BX, CX, DX, SI, DI et BP ainsi que le contenu de tous les registres de segment ne sont pas modifiés par cette fonction.

Interruption 10h, Fonction 10h, Sous-fonction 15h VGA
Lit un des registres de couleur DAC

Fournit au programme d'appel le contenu de l'un des 256 registres de couleur DAC.

Entrée : AH = 10h
AL = 15h
BX = Numéro du registre de couleur DAC

Sortie : CH = Vert
CL = Bleu
DH = Rouge

Remarques :
- Seuls les bits 0 à 5 des registres CH, CL et DH participent au mélange de la couleur.
- Le contenu des registres BX, DL, SI, DI et BP ainsi que le contenu de tous les registres de segment ne sont pas modifiés par cette fonction.

Interrupt 10h, Fonction 10h, Sous-fonction 17h VGA
Fixer le contenu de plusieurs registres de couleur DAC

Cette fonction, qui représente une extension de la fonction 10h, permet de définir en une fois le contenu du plusieurs registres de couleur DAC.

Entrée : AH = 10h
AL = 17h
BX = Numéro du premier registre de couleur DAC appelé (0 à 255)
CX = Nombre de registres à définir
ES = Adresse de segment du buffer
DX = Adresse d'offset du buffer

Sortie : Aucune

Remarques :
- Le buffer spécifié doit comporter pour chacun des registres de couleur à définir un groupe de 3 octets consécutifs, dont le premier définit la part de rouge, le deuxième la part de vert et le troisième la part de bleu. Les trois premiers octets correspondent au premier registre de couleur DAC appelé, les trois octets suivants au registre de couleur DAC suivant, etc.
- Pour le mélange de la couleur, seuls les bits 0 à 5 sont significatifs, les autres bits sont ignorés.
- Si la somme de BX et CX est supérieure à 255, c'est le premier registre de couleur DAC qui sera traité après que le dernier registre de couleur DAC ait été traité. Il y a donc une exécution en boucle (Wrap Around).

- Le contenu des registres BX, CX, DX, SI, DI et BP ainsi que le contenu de tous les registres de segment ne sont pas modifiés par cette fonction.

Interrupt 10h, Fonction 10h, Sous-fonction 18h VGA
Définit le registre de masque DAC

Charge la valeur transmise dans le registre de masque DAC.

Entrée : AH = 10h
AL = 18h
BL = Valeur pour le registre de masque DAC

Sortie : Aucune

Remarques :
- Le contenu du registre de masque DAC intervient dans la sélection de couleur car il est combiné par un ET logique avec l'index servant à l'accès à la table de couleurs DAC.
- Le contenu des registres BH, CX, DX, SI, DI et BP ainsi que le contenu de tous les registres de segment ne sont pas modifiés par cette fonction.

Interruption 10h, Fonction 10h, Sous-fonction 19h VGA
Lire le contenu du registre de masque DAC

Lit le contenu actuel du registre de masque DAC.

Entrée : AH = 10h
AL = 19h

Sortie : BL = Contenu du registre de masque DAC

Remarques :
- Le contenu du registre de masque DAC intervient dans la sélection de couleur car il est combiné par un ET logique avec l'index servant à l'accès à la table de couleurs DAC.
- Le contenu des registres BH, CX, DX, SI, DI et BP ainsi que le contenu de tous les registres de segment ne sont pas modifiés par cette fonction.

Interrupt 10h, Fonction 10h, Sous-fonction 1Ah VGA
Déterminer la méthode de sélec. de couleur et le contenu du registre de sélec. de couleur

Cette fonction fournit d'une part le contenu du bit 7 du registre de contrôle de mode, et donc la méthode de sélection de couleur, ainsi que le contenu du registre de sélection de couleur, qui détermine le groupe de registres de couleur DAC activé.

Entrée : AH = 10h
AL = 1Ah

Sortie : BL = Bit 7 du registre de contrôle de mode
BH = Contenu du registre de sélection de couleur

Remarques :
- Le contenu des registres BX, CX, DX, SI, DI et BP ainsi que le contenu de tous les registres de segment ne sont pas modifiés par cette fonction.

Interruption 10h, Fonction 10h, Sous-fonction 1Bh VGA
Convertir le contenu des registres de couleur DAC en nuances de gris

Cette fonction convertit une section déterminée de la table de couleurs DAC en nuances de gris correspondantes.

Entrée : AH = 10h
AL = 1Bh
BX = Numéro du premier registre de couleur DAC à convertir
CX = Nombre de registres de couleur DAC à traiter

Sortie : Aucune

Remarques :
- La conversion en une nuance de gris s'effectue par une pondération différenciée des parts de rouge, de vert et de bleu, qui reflète les différences d'intensité entre ces couleurs fondamentales sur l'écran. Le coefficient appliqué au rouge est 0,3, celui appliqué au vert 0,59, et celui appliqué au bleu 0,11.
- Le contenu des registres BX, CX, DX, SI, DI et BP ainsi que le contenu de tous les registres de segment ne sont pas modifiés par cette fonction.

Interruption 10h, Fonction 11h, Sous-fonction 00h BIOS EGA/VGA
Ecran : Charger le jeu de caractères défini par l'utilisateur

Cette fonction charge un jeu de caractères défini par l'utilisateur de la mémoire RAM dans l'une des deux tables de caractères EGA/VGA.

Entrée :
- AH = 11h
- AL = 00h
- BH = lignes par caractère (et donc octets par caractère)
- BL = table de caractères appelée (0 ou 1)
- CX = nombre de caractères dans la table
- DX = code ASCII du premier caractère dans la table
- ES = adresse de segment de la table de caractères dans la RAM
- BP = adresse d'offset de la table de caractères dans la RAM

Sortie : aucune

Remarques :
- 512 caractères peuvent être chargés au maximum par table de caractères.
- Le jeu de caractères chargé n'est pas activé et les registres du CRTC ne sont pas non plus programmés de façon à adapter l'affichage des caractères sur l'écran à la taille des caractères. Les modifications ne sont donc perceptibles sur l'écran que si la table de caractères dans laquelle sont chargées les définitions de caractères est la table actuellement activée.
- Le contenu des registres BX, CX, DX, SI, DI et BP ainsi que celui de tous les registres de segment n'est pas modifié par cette fonction.

Interruption 10h, Fonction 11h, Sous-fonction 01h BIOS EGA/VGA
Ecran : Charger le jeu de caractères 8*14

Cette fonction charge le jeu de caractères 8*14 points tout entier du BIOS en ROM dans l'une des deux tables de caractères.

Entrée :
- AH = 11h
- AL = 01h
- BL = table de caractères appelée (0 ou 1)

Sortie : aucune

Remarques :
- Le jeu de caractères chargé n'est pas activé et les registres du CRTC ne sont pas non plus programmés de façon à adapter l'affichage des caractères sur l'écran à la taille des caractères. Les modifications

ne sont donc perceptibles sur l'écran que si la table de caractères dans laquelle sont chargées les définitions de caractères est la table actuellement activée. La carte EGA affiche dans ce cas 25 lignes sur l'écran, la carte VGA 28.

- Le contenu des registres BX, CX, DX, SI, DI et BP ainsi que celui de tous les registres de segment n'est pas modifié par cette fonction.

Interruption 10h, Fonction 11h, Sous-fonction 12h BIOS EGA/VGA
Ecran : Charger le jeu de caractères 8*8

Cette fonction charge le jeu de caractères 8*8 points tout entier du BIOS en ROM dans l'une des deux tables de caractères.

Entrée : AH = 11h
AL = 12
BL = table de caractères appelée (0 ou 1)

Sortie : aucune

Remarques :

- Le jeu de caractères chargé n'est pas activé et les registres du CRTC ne sont pas non plus programmés de façon à adapter l'affichage des caractères sur l'écran à la taille des caractères. Les modifications ne sont donc perceptibles sur l'écran que si la table de caractères dans laquelle sont chargées les définitions de caractères est la table actuellement activée. La carte EGA affiche dans ce cas 43 lignes sur l'écran, la carte VGA 50.
- Le contenu des registres BX, CX, DX, SI, DI et BP ainsi que celui de tous les registres de segment n'est pas modifié par cette fonction.

Interruption 10h, Fonction 11h, Sous-fonction 03h BIOS EGA/VGA
Ecran : Activer un jeu de caractères

Active un (ou deux) des quatre jeux de 256 caractères.

Entrée : AH = 11h
AL = 03h
BL = numéro du jeu de caractères à activer

Sortie : aucune

Remarques :

- Les bits 0 et 1 du registre BL spécifient le numéro du jeu de caractères auquel il faut accéder lorsque le bit 3 de l'octet d'attribut vaut 0.
- Les bits 2 et 3 du registre BL spécifient le numéro du jeu de caractères auquel il faut accéder lorsque le bit 3 de l'octet d'attribut vaut 1.
- Si le contenu des bits 0 et 1 ainsi que celui des bits 2 et 3 du registre BL est identique, le bit 3 de l'octet d'attribut d'un caractère est sans effet sur le code ASCII du caractère affiché. Seuls 256 caractères différents peuvent donc être affichés sur l'écran.
- Le contenu des registres BX, CX, DX, SI, DI et BP ainsi que celui de tous les registres de segment n'est pas modifié par cette fonction.

Interruption 10h, Fonction 11h, Sous-fonction 04h VGA
Charger le jeu de caractères 8*16

Cette fonction charge le jeu de caractères 8*16 points complet du BIOS en ROM de la carte VGA dans l'une des deux tables de caractères.

Entrée : AH = 11h
AL = 04h
BL = table de caractères appelée (0 ou 1)

Sortie : Aucune

Remarques :

- Le jeu de caractères chargé n'est pas activé et les registres du CRTC ne sont pas non plus programmés de façon à adapter l'affichage des caractères sur l'écran à la taille des caractères. Les modifications ne sont donc perceptibles sur l'écran que si la table de caractères dans laquelle sont chargées les définitions de caractères est la table actuellement activée. La carte VGA affiche dans ce cas 25 lignes sur l'écran.
- Le contenu des registres BX, CX, DX, SI, DI et BP ainsi que celui de tous les registres de segment n'est pas modifié par cette fonction.

Interruption 10h, Fonction 11h, Sous-fonction 10h BIOS EGA/VGA
Ecran : Charger et activer un jeu de caractères défini par l'utilisateur

Cette fonction charge un jeu de caractères défini par l'utilisateur dans l'une des deux tables de caractères EGA/VGA et l'active en programmant les registres du CRTC.

Entrée : AH = 11h
AL = 10h
BH = lignes par caractère (et donc octets par caractère)
BL = table de caractères appelée (0 ou 1)
CX = nombre de caractères dans la table
DX = code ASCII du premier caractère dans la table
ES = adresse de segment de la table de caractères dans la RAM
BP = adresse d'offset de la table de caractères dans la RAM

Sortie : aucune

Remarques :
- 512 caractères peuvent être chargés au maximum par table de caractères.
- Le nombre de lignes de texte affichées sur l'écran dépend de la hauteur des caractères. Il est égal au nombre de lignes de l'écran (350 ou 480) divisé par la hauteur de caractère.
- Les lignes de début et de fin du curseur clignotant de l'écran sont automatiquement adaptées à la hauteur de la nouvelle matrice de caractère.
- Le contenu des registres BX, CX, DX, SI, DI et BP ainsi que celui de tous les registres de segment n'est pas modifié par cette fonction.

Interruption 10h, Fonction 11h, Sous-fonction 11h BIOS EGA/VGA
Ecran : Charger et activer le jeu de caractères 8*14

Cette fonction charge le jeu de caractères 8*14 points tout entier du BIOS en ROM de la carte EGA/VGA dans l'une des deux tables de caractères et l'active en programmant les registres du CRTC.

Entrée : AH = 11h
AL = 11h
BL = table de caractères appelée (0 ou 1)

Sortie : aucune

Remarques :
- Le fait d'appeler cette fonction fait automatiquement passer l'écran en affichage de 25 lignes pour la carte EGA et de 28 lignes pour la carte VGA.
- Les lignes de début et de fin du curseur clignotant de l'écran sont automatiquement adaptées à la hauteur de la nouvelle matrice de caractère.
- Le contenu des registres BX, CX, DX, SI, DI et BP ainsi que celui de tous les registres de segment n'est pas modifié par cette fonction.

Interruption 10h, Fonction 11h, Sous-fonction 12h BIOS EGA/VGA
Ecran : Charger le jeu de caractères 8*8

Cette fonction charge le jeu de caractères 8*8 points tout entier du BIOS en ROM de la carte EGA/VGA dans l'une des deux tables de caractères et l'active en programmant les registres du CRTC.

Entrée : AH = 11h
AL = 12h
BL = table de caractères appelée (0 ou 1)

Sortie : Aucune

Remarques :
- Après appel de cette fonction, la carte EGA affiche 25 lignes sur l'écran, et la carte VGA 28 lignes.
- Les lignes de départ et de fin du curseur clignotant de l'écran sont automatiquement adaptées à la hauteur de la nouvelle matrice de caractère.
- Le contenu des registres BX, CX, DX, SI, DI et BP ainsi que celui de tous les registres de segment n'est pas modifié par cette fonction.

Interruption 10h, Fonction 11h, Sous-fonction 14h VGA
Charger jeu de caractères 8*16

Cette fonction charge le jeu de caractères 8*16 points complet du BIOS en ROM de la carte VGA dans l'une des deux tables de caractères et l'active en programmant les registres CRTC.

Entrée : AH = 11h
AL = 14h
BL = table de caractères appelée (0 ou 1)

Sortie : Aucune

Remarques :
- Après appel de cette fonction, la carte VGA affiche 25 lignes de texte sur l'écran.
- Les lignes de départ et de fin du curseur clignotant de l'écran sont automatiquement adaptées à la hauteur de la nouvelle matrice de caractère.
- Le contenu des registres BX, CX, DX, SI, DI et BP ainsi que celui de tous les registres de segment n'est pas modifié par cette fonction.

Interruption 10h, Fonction 11h, Sous-fonction 30h BIOS EGA/VGA
Ecran : Rechercher des informations sur le générateur de caractères

Cette fonction permet d'obtenir différentes informations sur l'état actuel du générateur de caractères.

Entrée : AH = 11h
AL = 30h
BH = Type d'informations recherchées
0 = Contenu du vecteur d'interruption 1Fh
1 = Contenu du vecteur d'interruption 43h
2 = Adresse de la table de caractères 8*14 en ROM
3 = Adresse de la table de caractères 8*8 en ROM
4 = Adresse de la seconde moitié de la table de caractères 8*8
5 = Adresse de la table de caractères 9*14 alternative en ROM
6 = Adresse de la table de caractères 8*16 alternative en ROM
7 = Adresse de la table de caractères 9*16 alternative en ROM

Sortie : CX = Hauteur de la matrice de caractère actuelle
DL = Nombre de colonnes par ligne - 1
ES = Adresse de segment du pointeur testé
BP = Adresse de segment du pointeur testé

Remarques :
- Le contenu des registres BX, DH, SI, DI ainsi que celui des registres de segment CS, DS et SS n'est pas modifié par cette fonction.

Interruption 10h, Fonction 12h, Sous-fonction 10h BIOS EGA/VGA
Ecran : Déterminer configuration EGA/VGA

Cette fonction sert à déterminer la configuration de la carte EGA/VGA.

Entrée : AH = 12h
BL = 10h

Sortie : BH = moniteur connecté
0 = moniteur couleur ou haute résolution
1 = moniteur monochrome
BL = taille de la RAM EGA
0 = 64 Ko
1 = 128 Ko
2 = 192 Ko
3 = 256 Ko
CL = moniteur connecté

Remarques :
- Pour le type de moniteur, les codes suivants sont employés :
  - 0Bh: moniteur monochrome
  - 08h: moniteur couleur
  - 09h: moniteur haute résolution (EGA ou Multisync)
- Le contenu des registres BX, CX, DX, SI, DI et BP ainsi que celui de tous les registres de segment n'est pas modifié par cette fonction.

Interruption 10h, Fonction 12h, Sous-fonction 20h BIOS EGA
Ecran : Activer routine de copie d'écran de remplacement

Comme la routine de copie d'écran normale du BIOS en ROM sort systématiquement une copie d'écran sur 25 lignes, elle ne convient pas pour une copie d'écran sous les modes EGA/VGA affichant plus de 25 lignes sur l'écran. C'est pourquoi en appelant cette fonction on peut faire installer une routine de copie d'écran de remplacement qui sortira sur l'imprimante autant de lignes qu'il y en a effectivement d'affichées sur l'écran.

Entrée : AH = 12h
BL = 20h

Sortie : aucune

Remarques :
- Le contenu des registres BX, CX, DX, SI, DI et BP ainsi que celui de tous les registres de segment n'est pas modifié par cette fonction.

Interruption 10h, Fonction 12h, Sous-fonction 30h EGA/VGA
Fixer le nombre de lignes de l'écran (lignes de balayage)

Sélectionne le nombre de lignes apparaissant sur l'écran (à ne pas confondre avec les lignes de texte).

Entrée : AH = 12h
BL = 30h
AL = 0 : 200 lignes de balayage (EGA et VGA)
1 : 350 lignes de balayage (EGA et VGA)
2 : 400 lignes de balayage (possible uniquement avec VGA)

Sortie : Aucune

Remarques :
- Le nombre de lignes sélectionné ne peut être affiché sur l'écran que si un moniteur suffisamment puissant est relié à la carte vidéo. C'est ainsi qu'un moniteur CGA ne permettra jamais d'afficher plus de

200 lignes sur l'écran, même si l'électronique vidéo autorise une résolution plus élevée.

- Le contenu des registres BX, CX, DX, SI, DI et BP ainsi que le contenu de tous les registres de segment ne sont pas modifiés par cette fonction.

Interruption 10h, Fonction 12h, Sous-fonction 31h VGA
Activer ou désactiver le chargement des registres de palette

Cette fonction permet à un programme d'indiquer au BIOS VGA si les valeurs défaut doivent automatiquement être chargées dans les registres de palette lorsqu'un changement de mode vidéo est opéré à l'aide de la fonction 00h.

Entrée : AH = 12h
BL = 31h
AL = Chargement automatique des registres de palette
0 : Oui
1 : Non

Sortie : Aucune

Remarques :
- Le contenu des registres BX, CX, DX, SI, DI et BP ainsi que le contenu de tous les registres de segment ne sont pas modifiés par cette fonction.

Interruption 10h, Fonction 12h, Sous-fonction 32h VGA
Autoriser/verrouiller l'accès de l'unité centrale à la RAM vidéo

Cette fonction permet d'autoriser ou de verrouiller l'accès de l'unité centrale à la RAM vidéo et aux différents ports d'entrée/sortie d'une carte VGA.

Entrée : AH = 12h
BL = 32h
AL = 0 : Accès autorisé
1 : Accès interdit

Sortie : Aucune

Remarques :
- Le BIOS EGA ne disposant pas de cette fonction, l'accès direct de l'unité centrale à la carte vidéo peut néammoins être interdit en manipulant directement le bit 1 du registre de sortie (adresse de port 3C2h).

- Le contenu des registres BX, CX, DX, SI, DI et BP ainsi que le contenu de tous les registres de segment ne sont pas modifiés par cette fonction.

Interruption 10h, Fonction 12h, Sous-fonction 33h VGA
Activer/désactiver la conversion en gris des registres de couleur DAC

Alors que la sous-fonction 1Bh de la fonction 10h entraîne une conversion sélective du contenu des registres de couleur DAC en nuances de gris, cette fonction provoque une conversion systématique, lors de tous les accès à ces registres à travers le BIOS VGA.

Entrée : AH = 12h
BL = 33h
AL = Conversion en gris des registres de couleur DAC
0 : oui
1 : non

Sortie : Aucune

Remarques :
- Le contenu des registres BX, CX, DX, SI, DI et BP ainsi que le contenu de tous les registres de segment ne sont pas modifiés par cette fonction.

Interruption 10h, Fonction 12h, Sous-fonction 34h VGA
Activer/désactiver l'émulation du curseur

Pour que le programme appelant la fonction 01h (définition des lignes de départ et de fin du curseur de l'écran) n'ait pas à tenir compte des différentes résolutions de la matrice de caractère dans les différents modes, il peut se référer systématiquement à une matrice de caractère 8*8 et laisser au BIOS VGA le soin d'effectuer la conversion vers la taille de la matrice de caractère actuelle. Ce n'est cependant possible qu'en mode dit d'émulation du curseur, que cette fonction permet d'activer ou de désactiver.

Entrée : AH = 12h
BL = 34h
AL = Mode d'émulation du curseur ...
0 : oui
1 : non

Sortie : Aucune

Remarques : ■ Le contenu des registres BX, CX, DX, SI, DI et BP ainsi que le contenu de tous les registres de segment ne sont pas modifiés par cette fonction.

Interruption 10h, Fonction 12h, Sous-fonction 36h VGA
Interdire la construction de l'écran

Cette fonction permet à un programme d'interdire temporairement la construction de l'écran, pour effectuer des manipulations d'envergure à l'intérieur de la RAM vidéo.

Entrée : AH = 12h
BL = 36h
AL = Construction de l'écran ...
0 : oui
1 : non

Sortie : Aucune

Remarques : ■ Le contenu des registres BX, CX, DX, SI, DI et BP ainsi que le contenu de tous les registres de segment ne sont pas modifiés par cette fonction.

Interruption 10h, Fonction 13h BIOS EGA/VGA
Moniteur : Sortie d'une chaîne de caractères

Une chaîne de caractères est sortie sur le moniteur, dans une page déterminée du moniteur, à partir de la position du moniteur spécifiée. Les caractères sont pour cela tirés d'un buffer dont l'adresse doit être communiquée à la fonction.

Entrée : AH = 13h
AL = Mode de sortie (0 à 3)
0 = attribut dans BL, conserver la position du curseur
1 = attribut dans BL, actualiser la position du curseur
2 = attribut dans le buffer, conserver la position du curseur
3 = attribut dans le buffer, actualiser la position du curseur
BL = octet d'attribut des caractères (seulement modes 0 et 1)
CX = nombre de caractères à sortir
DH = ligne du moniteur
DL = colonne du moniteur
BH = page du moniteur
ES = adresse de segment du buffer
BP = adresse d'offset du buffer

Sortie : pas de sortie

Remarques :

- Sous les modes 1 et 3, la position du curseur est fixée, après sortie de la chaîne de caractères, à la suite du dernier caractère de la chaîne, de sorte que lors d'appels ultérieurs de fonctions BIOS de sortie de caractères, les caractères seront sortis à la suite de la chaîne de caractères. La position du curseur n'est pas changée sous les modes 0 et 2.
- Sous les modes 0 et 1, le buffer ne contient que les codes ASCII des caractères à sortir. La couleur de tous les caractères de la chaîne de caractères est dans ce cas définie par le registre BL. Sous les modes 2 et 3 par contre, chaque caractère est suivi dans le buffer de l'octet d'attribut correspondant, de sorte que chaque caractère dispose d'un attribut particulier. Sous ces modes, il n'est pas nécessaire de définir la valeur du registre BL. Bien que la chaîne de caractères ait, sous ces modes, une taille double du nombre de caractères à sortir, ce n'est pas la longueur de la chaîne de caractères mais bien le nombre de caractères ASCII à sortir qui doit être spécifié dans le registre CX.
- Les codes de contrôle spéciaux comme "bell" et "carriage return" sont interprétés effectivement comme des codes de contrôle et non comme des codes ASCII. Si cependant Carriage Return et Line Feed sont sortis en dehors de la page 0 du moniteur, une erreur en résulte car ces codes sont systématiquement sortis dans la page 0, quelle que soit la page de moniteur spécifiée.
- Si la dernière position de l'écran est atteinte, l'écran est décalé d'une ligne vers le haut et la sortie se poursuit dans la première colonne de la dernière ligne de l'écran.
- Pour une sortie en mode graphique, l'attribut dans le registre BL indique la couleur de premier plan du caractère. (La couleur de fond des caractères est toujours 0.). Si le bit 7 du registre AL est mis, le code couleur sera combiné par un XOR avec le code couleur antérieur des points graphiques.
- Cette fonction permet aussi de sortir des caractères en mode graphique, les motifs des caractères étant tirés de l'une des tables de caractères EGA.
- Le contenu des registres BX, CX, DX, SI, DI et BP ainsi que celui de tous les registres de segment n'est pas modifié par cette fonction.

Interruption 10h, Fonction 14h BIOS EGA/VGA
Non documentée

La signification de cette fonction n'est pas connue. Elle ne doit donc pas être appelée par un programme utilisateur.

Interruption 10h, Fonction 15h BIOS EGA/VGA
Non documentée

La signification de cette fonction n'est pas connue. Elle ne doit donc pas être appelée par un programme utilisateur.

Interruption 10h, Fonction 16h BIOS EGA/VGA
Non documentée

La signification de cette fonction n'est pas connue. Elle ne doit donc pas être appelée par un programme utilisateur.

Interruption 10h, Fonction 17h BIOS EGA/VGA
Non documentée

La signification de cette fonction n'est pas connue. Elle ne doit donc pas être appelée par un programme utilisateur.

Interruption 10h, Fonction 18h BIOS EGA/VGA
Non documentée

La signification de cette fonction n'est pas connue. Elle ne doit donc pas être appelée par un programme utilisateur.

Interruption 10h, Fonction 19h BIOS EGA/VGA
Non documentée

La signification de cette fonction n'est pas connue. Elle ne doit donc pas être appelée par un programme utilisateur.

Interruption 10h, Fonction 1Ah, Sous-fonction 00h VGA
Déterminer le type d'adapteur vidéo primaire et secondaire

Entrée : AH = 13h
AL = 0
Sortie : AL = 1Ah
BL = Code de périphérique de la carte vidéo activée
BH = Code de périphérique de la carte vidéo non activée

Remarques :

- Si la valeur 1Ah n'est pas renvoyée dans le registre AL, c'est qu'aucun BIOS VGA et donc aucune carte VGA n'est installé.
- Les codes de périphérique suivants sont renvoyés :

  FFh = Carte vidéo inconnue
  00h = Pas de carte vidéo
  01h = MDA avec écran monochrome
  02h = CGA avec moniteur CGA
  03h = Réservé
  04h = EGA avec moniteur EGA ou Multisync
  05h = EGA avec moniteur monochrome
  06h = Réservé
  07h = VGA avec écran monochrome analogique
  08h = VGA avec écran couleur analogique (VGA, Multisync)
  09h = Réservé
  0Ah = Carte MCGA avec moniteur CGA (PS/2 uniquement)
  0Bh = MCGA avec écran monochrome analogique (PS/2 uniquement)
  0Ch = MCGA avec écran couleur analogique (PS/2 uniquement)
- Il suffit qu'une seule carte vidéo soit présente dans le registre BH pour que la valeur 00h soit retournée.
- Le contenu des registres CX, DX, SI, DI et BP ainsi que le contenu de tous les registres de segment ne sont pas modifiés par cette fonction.

Interruption 10h, Fonction 1Ah, Sous-fonction 01h VGA
Définir les cartes vidéo primaire et secondaire

A l'intérieur d'un système équipé de deux cartes vidéo, cette fonction permet d'activer l'une des deux cartes et de désactiver l'autre.

Entrée : AH = 1Ah
AL = 1

BL = Code de périphérique de la carte vidéo active
BH = Code de périphérique de la carte vidéo inactive

Sortie : AL = 1Ah

Remarques :
- Si la valeur 1Ah n'est pas retournée dans le registre AL, cela signifie que ni un BIOS VGA ni une carte VGA n'est installé.
- La liste des codes de périphérique est fournie dans le cadre de la description de la sous-fonction 00h.
- Le contenu des registres BX, CX, DX, SI, DI et BP ainsi que celui de tous les registres de segment n'est pas modifié par cette fonction.

Interruption 10h, Fonction 1Bh VGA
Lire les informations d'état concernant le mode vidéo et le BIOS VGA

Cette fonction permet de lire toutes les caractéristiques du mode vidéo en cours ainsi que les informations concernant les techniques fondamentales du BIOS VGA.

Entrée : AH = 1Bh
BX = 0
ES:DI = Pointeur sur un buffer

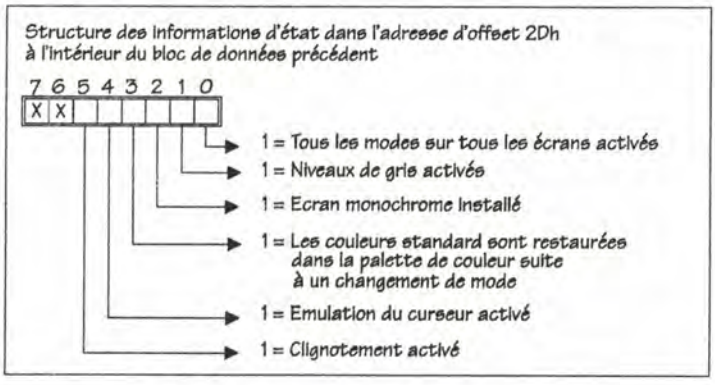
Sortie : AL = 1Bh

Remarques :
- Si la valeur 1Bh n'est pas retournée dans le registre AL, cela signifie que ni un BIOS VGA ni une carte VGA n'est installé.
- Le buffer transmis par l'appel de cette fonction doit contenir au moins 64 Ko parce que le BIOS VGA y place une table comme celle présentée plus bas. Cette table contient toutes les informations décrivant le mode vidéo en cours.

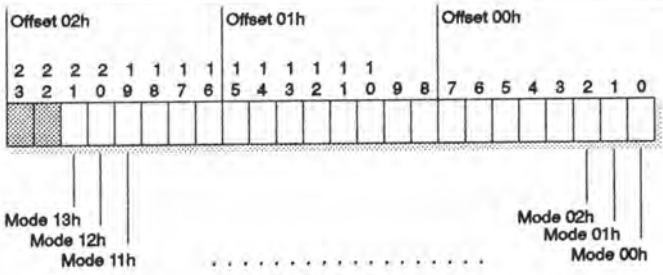
  Un pointeur se trouve au début de cette table. Il conduit vers une autre table située dans la ROM BIOS et pour laquelle le programme d'appel ne doit allouer aucune place mémoire. Elle reproduit les techniques du BIOS VGA qui sont indépendantes de la définition actuelle des différents paramètres (mode vidéo, registre de palettes, etc.).
- Le contenu des registres BX, CX, DX, SI, DI et BP ainsi que celui de tous les registres de segment n'est pas modifié par cette fonction.

Structure du bloc de données avec les informations d'état du BIOS VGA compte tenu du mode		
Offset	Contenu	Type
00h	Adresse de la table contenant les informations statiques	1 PTR
04h	Numéro de code du mode vidéo en cours	1 BYTE
05h	Nombre de colonnes de l'écran ou de points affichées	1 WORD
07h	Longueur d'une page écran à l'intérieur de la RAM vidéo	1 WORD
09h	Adresse de début de la page écran en cours dans la RAM vidéo	1 WORD
0Bh	Positions du curseur dans les huit pages écran maximales où la spécification de la colonne prévaut toujours par rapport à celle de la ligne	16 BYTE
1Bh	Ligne de fin du curseur (ligne de points)	1 BYTE
1Ch	Ligne de début du curseur (ligne de points)	1 BYTE
1Dh	Numéro de la page écran en cours	1 BYTE
1Eh	Adresse du port du registre d'adresse du contrôleur CRT	1 WORD
20h	Contenu actuel du registre CRTC dans l'adresse du port 3B8h (émulation DMA) ou 3D8h (VGA)	
21h	Contenu actuel du registre de sélection de couleurs dans l'adresse du port 3D9h	1 BYTE
22h	Nombre de lignes écran affichées	1 BYTE
23h	Hauteur des caractères en lignes de points	1 WORD
25h	Numéro de code de l'adaptateur vidéo en cours (voir la fonction 1Ah, sous-fonction 00h)	1 BYTE
26h	Numéro de code de l'adaptateur vidéo inactif (voir la fonction 1Ah, sous-fonction 00h)	1 BYTE
27h	Nombre de couleurs à afficher (0 = monochrome)	1 WORD
29h	Nombre de pages écran	1 BYTE
2Ah	Nombre de lignes de points affichées	1 BYTE
2Bh	Numéro de la table de caractères utilisée et dont le bit 3 vaut 0 dans l'octet d'attribut	1 BYTE
2Ch	Numéro de la table de caractères utilisée et dont le bit 3 vaut 1 dans l'octet d'attribut	1 BYTE
2Dh	Informations diverses #1 (voir plus bas)	1 BYTE

Structure du bloc de données avec les informations d'état du BIOS VGA compte tenu du mode		
Offset	Contenu	Type
2Eh	Réservé	3 BYTE
31h	Taille de la RAM vidéo disponible	1 BYTE
32h	Réservé	14 BYTE
64 octets		

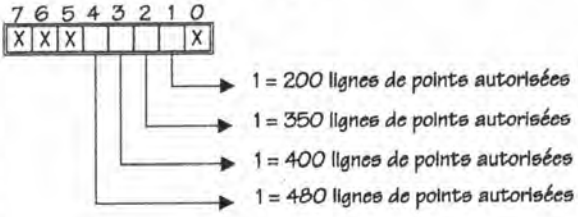


Structure du bloc de données avec les informations d'état du BIOS VGA compte tenu du mode (informations statiques)		
Offset	Contenu	Type
00h	Modes vidéo reconnus (voir plus bas)	3 BYTE
03h	Réservé	4 BYTE
07h	Nombre des lignes de points dans les modes texte (voir plus bas)	1 BYTE
08h	Nombre maximal des jeux de caractères affichables	1 BYTE
09h	Nombre des tables de jeux de caractères à charger dans la RAM vidéo	1 BYTE
0Ah	Informations d'état sur les capacités du BIOS VGA (voir plus bas)	1 BYTE
0Bh	Autres informations d'état sur les capacités du BIOS VGA (voir plus bas)	1 BYTE
0Ch	Réservé	2 BYTE
0Eh	Réservé	2 BYTE
16 octets		

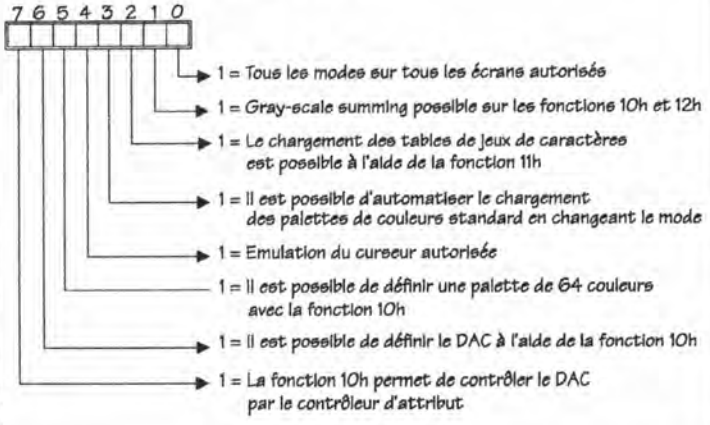


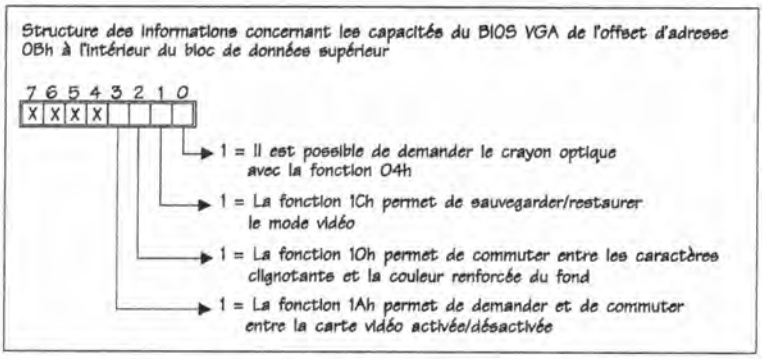
Les bits contiennent la valeur 1b lorsque le mode vidéo correspondant est soutenu

Structure des informations concernant le nombre de lignes de points dans les modes texte de l'adresse d'offset 07h à l'intérieur du bloc de données supérieur



Structure des informations concernant les capacités du BIOS VGA de l'offset d'adresse 0Ah à l'intérieur du bloc de donnée supérieur





Interruption 10h, Fonction 1Ch, Sous-fonction 00h VGA
Demander la taille de la zone de sauvegarde

Les trois sous-fonctions de la fonction 1Ch permettent de sauvegarder l'état en cours de l'équipement vidéo et du BIOS VGA dans un buffer du programme. A cet effet, le programme d'appel doit d'abord demander la taille du buffer nécessaire à l'aide de cette sous-fonction.

Entrée : AH = 1Ch
AL = 0
CX = Composantes à sauvegarder (voir plus bas)

Sortie : AL = 1Ch
BX = Taille de buffer en unités de 64 octets

Remarques :

- Si la valeur 1Ch n'est pas retournée dans le registre AL, cela signifie que ni un BIOS VGA ni une carte VGA n'est installé.
- Lors de l'appel de la fonction, les trois bits inférieurs indiquent les composantes de l'équipement vidéo et le BIOS VGA à sauvegarder dans le registre CX. Le bit approprié est à régler sur 1 si les éléments correspondants doivent être sauvegardés.

  Bit 0 = Equipement vidéo
  Bit 1 = Zone de données du BIOS VGA
  Bit 3 = Contenu du registre DAC
- Notez que "l'équipement vidéo" n'inclut pas également le contenu de la RAM vidéo. La sauvegarde de ce dernier est à la charge du programme utilisateur.
- Le contenu des registres BX, CX, DX, SI, DI et BP ainsi que celui de tous les registres de segment n'est pas modifié par cette fonction.

Interruption 10h, Fonction 1Ch, Sous-fonction 01h VGA
Sauvegarder l'état vidéo

Après avoir demandé le buffer nécessaire à l'aide de la sous-fonction 00h, vous pouvez sauvegarder l'état du matériel vidéo et/ou du BIOS VGA que vous souhaitez avec cette fonction.

Entrée : AH = 1Ch
AL = 1
CX = Composantes à sauvegarder (voir sous-fonction 00h)
ES:BX = Pointeur sur le buffer devant stocker les informations

Sortie : AL = 1Ch

Remarques :
- Si la valeur 1Ch n'est pas retournée dans le registre AL, cela signifie que ni un BIOS VGA ni une carte VGA n'est installé.
- Avant d'appeler cette fonction, il faut d'abord obtenir la taille du buffer nécessaire à l'aide de la sous-fonction 00h. Notez que le contenu du registre CX ne varie pas entre les deux appels sinon le buffer transmis risque d'être trop petit.
- Le contenu des registres BX, CX, DX, SI, DI et BP ainsi que celui de tous les registres de segment n'est pas modifié par cette fonction.

Interruption 10h, Fonction 1Ch, Sous-fonction 02h VGA
Restaurer l'état vidéo

Suite à un appel préalable de la sous-fonction 01h, cette fonction peut restaurer l'état sauvegardé du matériel vidéo et/ou du BIOS VGA.

Entrée : AH = 1Ch
AL = 2
CX = Composantes à restaurer (voir sous-fonction 00h)
ES:BX = Pointeur sur le buffer dans lequel les informations ont été sauvegardées

Sortie : AL = 1Ch

Remarques :
- Si la valeur 1Ch n'est pas retournée dans le registre AL, cela signifie que ni un BIOS VGA ni une carte VGA n'est installé.

- L'appel de cette fonction n'a un sens que s'il s'effectue après un appel de la sous-fonction 01h. Le nombre de composantes restaurées dépend naturellement du nombre qui a été préalablement sauvegardé. Avant d'appeler cette fonction, pensez donc à vérifier le contenu du registre CX.
- Le contenu des registres BX, CX, DX, SI, DI et BP ainsi que celui de tous les registres de segment n'est pas modifié par cette fonction.

C. Le standard VESA

L'interface VESA fournit des fonctions standardisées pour l'accès aux cartes Super VGA en libérant un programme des spécificités physiques des diverses cartes. Les six fonctions du standard VESA sont appelées comme des sous-fonctions de la fonction 4Fh, ces dernières étant intégrées par le driver VESA (ou le BIOS VGA étendu au VESA) dans l'interruption vidéo BIOS 10h.

Les modes graphiques suivants sont actuellement reconnus par les diverses fonctions VESA.

Les modes graphiques du BIOS VESA

Code	Résolution	Couleurs	Mémoire
100h	640*400	256	256 Ko
101h	640*480	256	512 Ko
102h	800*600	16	256 Ko
103h	800*600	256	512 Ko
104h	1024*768	16	512 Ko
105h	1024*768	256	1 Mo
106h	1280*1024	256	1,25 Mo
6Ah	800*600	16	256 Ko

Interruption 10h, Fonction 4Fh, Sous-fonction 00h VESA
Obtenir les spécificités de la carte Super VGA

Cette fonction permet de prendre connaissance des techniques offertes par la carte Super VGA installée et déterminer si les fonctions VESA sont soutenues.

Entrée : AH = 4Fh
AL = 00h
ES:DI : Pointeur FAR sur le buffer Info

Sortie : AL = 4Fh et
AH = 00h : Fonctions VESA soutenues

Remarques :

- Le buffer Info dont l'adresse est à transmettre à la fonction doit disposer de 256 octets. Si la fonction s'exécute correctement, vous obtenez les informations suivantes :

Structure du buffer Info		
Offset	Contenu	Type
00h	Signature VESA ("VESA")	4 BYTE
04h	Version VESA, Numéro de version principal	1 BYTE
05h	Version VESA, Numéro de version secondaire	1 BYTE
06h	Pointeur FAR sur chaîne ASCIIZ contenant le nom du fabricant de cartes	1 DWORD
0Ah	Flag indiquant les performances de la carte. Actuellement inutilisé, donc 0000h.	
0Eh	Pointeur FAR sur liste contenant les numéros de code des modes vidéo reconnus	1 DWORD

La liste des numéros de codes des modes vidéo reconnus, transmise dans le dernier champ du buffer, se compose de divers Words indiquant chacun le code d'un mode vidéo d'après le tableau ci-dessus. Cette liste se termine par un Word de valeur 0FFFFh. Sa longueur varie d'une carte à l'autre.

Interruption 10h, Fonction 4Fh, Sous-fonction 01h VESA
Déterminer les données-clés d'un mode VESA

Cette fonction donne des informations sur un mode VESA sans pour autant l'activer.

Entrée : AH = 4Fh
AL = 01h
CX = Numéro de code du mode VESA souhaité
ES:DI = Pointeur FAR sur buffer Info

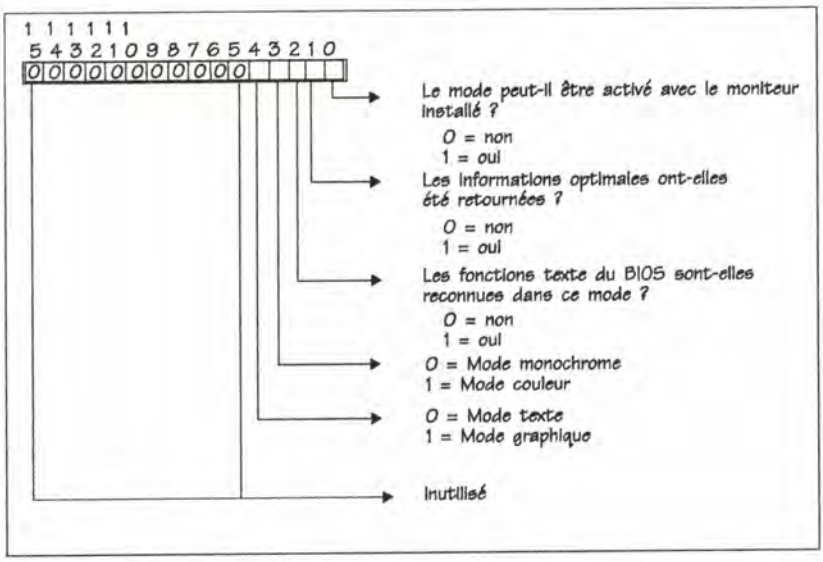
Sortie : AL = 4Fh et
AH = 00h : Fonction non exécutée correctement

Remarques :

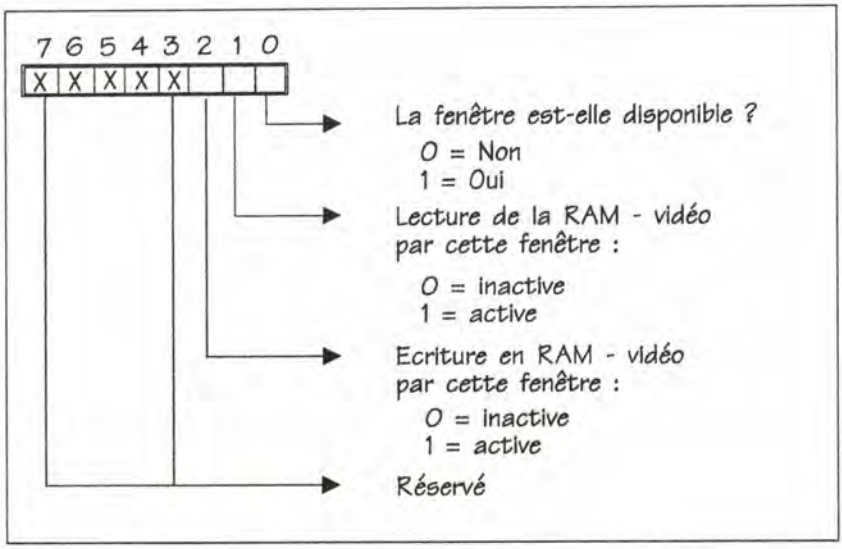
- Cette fonction ne doit être appelée que si la sous-fonction 00h s'est exécutée correctement et ainsi l'existence d'un driver VESA a pu être signalée. En outre, seuls les modes inclus dans la liste de la fonction 00h peuvent être réclamés.
- Le buffer Info dont l'adresse est à transmettre à la fonction doit disposer de 29 octets. Si la fonction s'exécute correctement, vous obtenez les informations suivantes :

Structure du buffer Info		
Offset	Contenu	Type
00h	Flag de mode, voir plus loin	1 WORD
02h	Flags pour la première fenêtre d'accès, voir plus loin	1 BYTE
03h	Flags pour la seconde fenêtre d'accès, voir plus loin	1 BYTE
04h	Granularité en Ko à utiliser pour déplacer les deux fenêtres	1 WORD
06h	Taille des deux fenêtres d'accès en Ko	1 WORD
08h	Adresse de segment de la première fenêtre d'accès	1 WORD
0Ah	Adresse de segment de la seconde fenêtre d'accès	1 WORD
0Ch	Pointeur FAR sur routine de définition de la zone visible dans les deux fenêtres d'accès	1 DWORD
10h	Nombre d'octets occupant chacun une ligne de points dans la RAM vidéo	1 WORD
Informations en option, cf. Flag de mode		
12h	Résolution X en points/caractères	1 WORD
14h	Résolution Y en points/caractères	1 WORD
16h	Largeur de la matrice de caractères en points	1 BYTE
17h	Hauteur de la matrice de caractères en points	1 BTYE
18h	Nombre des plans de bits	1 BYTE
19h	Nombre de bits par point écran	1 BYTE
1Ah	Nombre de blocs mémoire	1 BYTE
1Bh	Modèle de mémoire	1 BYTE
1Ch	Taille des blocs de mémoire en Ko	1 BYTE

Le flag de mode (offset 00h) indique si les champs optionnels du buffer Info ont été complétés par la fonction VESA et retourne également des informations importantes concernant le mode souhaité.



Les deux fenêtres d'accès (Offset 02h et 03h) sont décrites à travers les champs de bits suivants :



Le modèle de mémoire (Offset 1Bh) reproduit la structure de la RAM vidéo dans le mode vidéo souhaité. Les codes suivants sont reconnus à cet effet :

Codes de description du modèle de mémoire	
Nº	Fonction
00h	Mode texte
01h	Format CGA, soit 2 ou 4 blocs de mémoire
02h	Format Hercules avec 4 blocs de mémoire
03h	Format normal EGA/VGA pour mode graphique 16 couleurs
04h	Format compacté avec deux points à 4 bits par octet
05h	Format normal EGA/VGA pour mode graphique 256 couleurs
06h-0Fh	Réservé
10h-FFh	Code spécifique au constructeur, actuellement inutilisé

Interruption 10h, Fonction 4Fh, Sous-fonction 02h VESA
Activer le mode VESA

Cette fonction sert à activer un mode VESA.

Entrée : AH = 4Fh
AL = 02h
BX = Numéro de code du mode souhaité

Sortie : AL = 4Fh et
AH = 00h : Fonction exécutée correctement

Remarques :
- Cette fonction ne doit être appelée que si la sous-fonction 00h s'est exécutée correctement et ainsi l'existence d'un driver VESA a pu être signalée. En outre, seuls les modes inclus dans la liste de la fonction 00h peuvent être initialisés.
- Le bit 15 peut être mis dans le registre BX contenant le numéro de code du mode vidéo si la RAM vidéo ne doit pas être effacée pendant l'initialisation du mode vidéo.

Interruption 10h, Fonction 4Fh, Sous-fonction 03h VESA
Obtenir le mode en cours

Cette fonction retourne le numéro de code du mode vidéo en cours et tient compte des modes non VESA.

Entrée : AH = 4Fh
AL = 03h

Sortie : AL = 4Fh et
AH = 00h : Fonction exécutée correctement, dans ce cas
BX = Numéro de code du mode en cours

Remarques :
- Cette fonction ne doit être appelée que si la sous-fonction 00h s'est exécutée correctement et ainsi l'existence d'un driver VESA a pu être signalée.

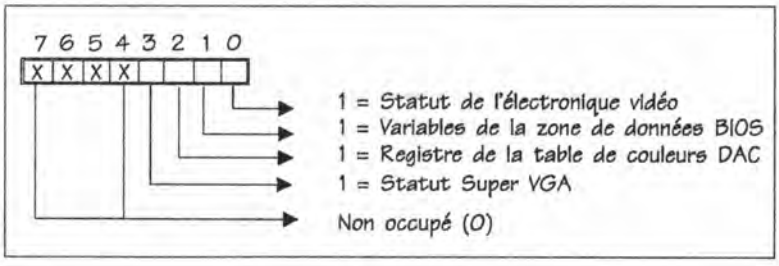
Interruption 10h, Fonction 4Fh, Sous-fonction 04h/00h VESA
Déterminer la taille du buffer de stockage

La taille du buffer de stockage nécessaire peut être obtenue à l'aide de cette fonction en vue d'un appel ultérieur de la sous-fonction 04h/01h.

Entrée : AH = 4Fh
AL = 04h
DL = 00h
CX = Eléments de l'état vidéo à sauvegarder

Sortie : AL = 4Fh et
AH = 00h : Fonction exécutée correctement, dans ce cas
BX = Nombre des 64 blocs d'octets associés nécessaires en guise de buffer de stockage

Remarques :
- Cette fonction doit être appelée avant la sous-fonction 04h/01h. La taille du buffer nécessaire à la sous-fonction 04h/01h peut ainsi être obtenue.
- Lors de l'appel de la fonction, les différents bits du registre CX indiquent les éléments de l'état vidéo à sauvegarder. Dans ce cas, seul l'octet faible du registre CX est occupé par les bits suivants :



Interruption 10h, Fonction 4Fh, Sous-fonction 04h/01h VESA
Sauver l'état vidéo de la carte Super VGA

Après un appel préalable de la sous-fonction 04h/00h, les informations souhaitées à propos des divers éléments de l'état vidéo sont stockées dans un buffer à l'aide de cette fonction.

Entrée : AH = 4Fh
AL = 04h
DL = 01h
CX = Eléments de l'état vidéo à sauvegarder
ES:BX = Pointeur FAR sur le buffer de stockage

Sortie : AL = 4Fh et
AH = 00h : Fonction exécutée correctement

Remarques :
- Le buffer transmis doit indiquer la taille obtenue lors de l'appel préalable de la sous-fonction 04h/00h à travers le registre BX.
- Lors de l'appel de la fonction, les différents bits du registre CX indiquent les éléments de l'état vidéo à stocker. Reportez-vous à ce propos à la sous-fonction 04h/00h.

Interruption 10h, Fonction 4Fh, Sous-fonction 04h/02h VESA
Restaurer l'état vidéo de la carte Super VGA

Après avoir sauvegardé l'état vidéo avec la sous-fonction 04h/01h, vous pouvez le restaurer en appelant cette fonction.

Entrée : AH = 4Fh
AL = 04h
DL = 01h
CX = Eléments de l'état vidéo à restaurer
ES:BX = Pointeur FAR sur l'adresse de segment du buffer de stockage

Sortie : AL = 4Fh et
AH = 00h : Fonction exécutée correctement

Remarques :
- Le buffer transmis doit auparavant être chargé avec les informations sur l'état vidéo à travers un appel de la sous-fonction 04h/01h.
- Lors de l'appel de la fonction, les différents bits du registre CX montrent les éléments de l'état vidéo à restaurer. Reportez-vous à ce propos à la sous-fonction 04h/00h.

Interruption 10h, Fonction 4Fh, Sous-fonction 05h/00h VESA
Placer la fenêtre d'accès sur la RAM vidéo

Cette fonction sert à insérer une portion précise de la RAM vidéo dans l'une des deux fenêtres d'accès VESA pour qu'un programme puisse l'adresser.

Entrée : AH = 4Fh
AL = 05h
BH = 00h
BL = Fenêtre d'accès (0 ou 1)
DX = Adresse de départ

Sortie : AL = 4Fh et
AH = 00h : Fonction exécutée correctement

Remarques :
- Dans le registre DX, l'adresse de départ doit être considérée par rapport à la granularité de la fenêtre obtenue par l'appel de la sous-fonction 01h.
- La seconde fenêtre d'accès ne doit être réclamée que si un appel préalable de la sous-fonction 00h confirme l'existence de deux fenêtres d'accès.

Interruption 10h, Fonction 4Fh, Sous-fonction 05h/01h VESA
Déterminer la fenêtre d'accès sur la RAM vidéo

Cette fonction permet de déterminer l'état de la fenêtre d'accès sur la RAM vidéo de la carte Super VGA.

Entrée : AH = 4Fh
AL = 05h
BH = 01h
BL = Fenêtre d'accès (0 ou 1)

Sortie : AL = 4Fh et
AH = 00h : Fonction exécutée correctement, dans ce cas
DX = Adresse de départ

Remarques :
- L'adresse de départ située dans le registre DX doit être considérée par rapport à la granularité de la fenêtre obtenue par l'appel de la sous-fonction 01h.

D. Description des interruptions et fonctions du DOS

Interface de programmation d'application DOS (DOS API)

L'interruption 21h permet d'appeler plus de 100 fonctions que DOS met à la disposition d'un programme et c'est pourquoi on les appelle Interface de programmation d'application (DOS API).

Ces fonctions sont décrites dans ce chapitre ainsi qu'une série d'autres fonctions dont la signification n'a jamais été officialisée par Microsoft. Elles sont donc désignées comme des fonctions non documentées. Les fonctions non documentées citées ici sont toutefois utilisées dans les milliers d'applications commercialisées actuellement. Ainsi, Microsoft ne peut pas se permettre de les éliminer de l'API du jour au lendemain dans le cadre d'une nouvelle version du DOS. Par conséquent, n'hésitez surtout pas à vous en servir toutes les fois que vous ne trouvez pas une fonction officielle répondant à vos besoins.

Interruption 21h, Fonction 00h DOS (à partir de 1.0)
Terminer programme

En appelant cette fonction, on indique au système d'exploitation que l'exécution du programme actuellement traité doit s'achever et que le contrôle doit être rendu au programme d'appel. Avant que cela ne se produise, toutefois, les trois vecteurs d'interruption dont le contenu avait été sauvegardé dans le PSP avant appel du programme, sont restaurés. Si le programme a redirigé ces vecteurs sur des routines à lui, ces routines ne pourront être effacées par un autre programme puisque la mémoire RAM occupée par le programme qu'il s'agit de terminer est à nouveau libérée pour d'autres programmes. Cette mémoire est donc libérée et tous les buffers de fichiers sont vidés avant que le contrôle ne soit rendu au programme d'appel.

Entrée : AH = 00h
CS = Adresse de segment du PSP

Sortie : aucune

Remarques :

- Pour un programme COM, l'adresse de segment du PSP est de toute façon stockée dans le registre CS. Dans un programme EXE, le code et le PSP ne sont pas logés dans le même segment. Cette fonction ne peut donc être appelée par un programme EXE.
- Il est préférable d'utiliser, plutôt que cette fonction, les fonctions 31h ou 4Ch de l'interruption 21h du DOS pour terminer un programme.

Interruption 21h, Fonction 01h DOS (à partir de 1.0)
Entrée de caractères avec sortie

Un caractère est lu sur le périphérique d'entrée standard et sorti sur le périphérique de sortie standard. Si aucun caractère n'est encore prêt au moment de l'appel de fonction, la fonction attend jusqu'à ce qu'un caractère soit disponible. Comme les entrée et sortie standard peuvent être redirigées, cette fonction ne lit pas nécessairement un caractère au clavier et ne le sort pas nécessairement sur l'écran. Les caractères entrés peuvent donc parfaitement provenir d'un autre périphérique ou d'un fichier. Dans ce dernier cas cependant, l'entrée n'est pas redirigée sur le clavier lorsqu'est atteinte la fin du fichier. La fonction continuera donc à essayer de lire des caractères dans le fichier, bien que ce ne soit plus possible.

Entrée : AH = 01h

Sortie : AL = Le caractère lu

Remarques :
- Lorsque des codes étendus sont lus, le code 0 est tout d'abord renvoyé dans le registre AL. La fonction doit être appelée à nouveau pour lire le code étendu lui-même.
- Lorsque le caractère entré est le caractère Control C (Code ASCII 3), l'interruption 23h est appelée.
- Le contenu des registres AH, BX, CX, DX, SI, DI, BP, CS, DS, SS, ES et du registre de flags n'est pas modifié par cette fonction.

Interruption 21h, Fonction 02h DOS (à partir de 1.0)
Sortie d'un caractère

Cette fonction permet de sortir un caractère sur le périphérique de sortie standard. Comme ce périphérique peut être redirigé, le caractère ne doit pas nécessairement être sorti sur l'écran mais peut aussi être envoyé sur un autre périphérique ou dans un fichier. Si toutefois le support (disquette, disque dur) sur lequel figure le fichier est déjà plein, la fonction ne détecte pas cette situation et elle continue à essayer d'écrire des caractères dans ce fichier.

Entrée : AH = 02h
DL = Code du caractère à sortir

Sortie : aucune

Remarques :
- Si les caractères sont envoyés sur l'écran, les codes de commande tels que Backspace, Carriage Return et Line Feed sont traités

comme tels. Si par contre la sortie est redirigée vers un fichier, les caractères y sont sauvegardés comme des codes ASCII normaux.

- Après sortie du caractère, le DOS teste si un caractère Control-C (code ASCII 3) a été reçu ou entré entre-temps. Si c'est le cas, l'interruption 23h est appelée.
- Cette fonction ne modifie le contenu d'aucun registre du processeur, y compris le registre de flags.

Interruption 21h, Fonction 03h DOS (à partir de 1.0)
Réception d'un caractère de l'interface série

Cette fonction permet de lire un caractère sur l'interface série. L'accès concerne le périphérique portant la désignation AUX ou COM1.

Entrée : AH = 03h

Sortie : AL = Le caractère reçu

Remarques :

- Comme l'interface série ne dispose pas de buffer interne, il peut arriver que l'interface série reçoive des caractères plus vite que l'appel de cette fonction ne permet de les lire. Ces caractères sont alors perdus.
- Les paramètres de communication (vitesse de transmission, nombre de bits Stop, etc...) doivent avoir été fixés préalablement à l'aide de l'instruction MODE. Le DOS définit sinon les valeurs défaut suivantes : vitesse de transmission de 2400 bauds avec un bit Stop, sans contrôle de parité et une largeur de données de 8 bits.
- Plutôt qu'à cette fonction, il est préférable de recourir aux fonctions du BIOS pour l'accès à l'interface série. Elles peuvent être appelées à travers l'interruption 14h et elles offrent une plus grande souplesse que les fonctions du DOS car elles permettent notamment de tester l'état de l'interface série.
- Si un caractère Control-C (code ASCII 3) est reçu, l'interruption 23h est appelée.
- Le contenu des registres AH, BX, CX, DX, SI, DI, BP, CS, DS, SS, ES et du registre de flags n'est pas modifié par cette fonction.

Interruption 21h, Fonction 04h DOS (à partir de 1.0)
Sortie d'un caractère sur l'interface série

Cette fonction permet de sortir un caractère sur l'interface série. L'accès concerne le périphérique portant la désignation AUX ou COM1.

Entrée : AH = 04h
DL = Le caractère à sortir

Sortie : aucune

Remarques :

- Le caractère n'est sorti qu'après que le périphérique qui doit recevoir le caractère ait signalé qu'il est prêt à recevoir. Ce n'est qu'après cela que le contrôle est rendu au programme d'appel.
- Les paramètres de communication (vitesse de transmission, nombre de bits Stop, etc...) doivent avoir été fixés préalablement à l'aide de l'instruction MODE. Le DOS définit sinon les valeurs défaut suivantes : vitesse de transmission de 2400 bauds avec un bit Stop, sans contrôle de parité et une largeur de données de 8 bits.
- Plutôt qu'à cette fonction, il est préférable de recourir aux fonctions du BIOS pour l'accès à l'interface série. Elles peuvent être appelées à travers l'interruption 14h et elles offrent une plus grande souplesse que les fonctions du DOS car elles permettent notamment de tester l'état de l'interface série.
- Si un caractère Control-C (code ASCII 3) est sorti, l'interruption 23h est appelée.
- Cette fonction ne modifie le contenu d'aucun registre du processeur, y compris le registre de flags.

Interruption 21h, Fonction 05h DOS (à partir de 1.0)
Sortie d'un caractère sur l'imprimante

Cette fonction sort un caractère sur l'imprimante. A moins que cette sortie n'ait été redirigée avec l'instruction MODE, l'accès se fait sur le périphérique portant la désignation LPT1 (identique à PRN).

Entrée : AH = 05h
DL = Code du caractère à sortir.

Sortie : aucune

Remarques :

- Le caractère n'est sorti que si l'imprimante signale qu'elle est prête à recevoir. Ce n'est qu'après cela que le contrôle est rendu au programme d'appel.
- Si un caractère Control-C (code ASCII 3) est détecté, l'interruption 23h est appelée.
- Pour la sortie de caractères sur l'imprimante, les fonctions du BIOS pour la communication avec l'imprimante, qui peuvent être appelées à travers l'interruption 17h, offrent une plus grande souplesse.
- Cette fonction ne modifie le contenu d'aucun registre du processeur, y compris le registre de flags.

Interruption 21h, Fonction 06h DOS (à partir de 1.0)
Entrée/sortie directe de caractère

Cette fonction permet de sortir des caractères sur le périphérique de sortie standard ou de les lire sur le périphérique d'entrée standard. Le caractère reçu ou écrit chaque fois n'est pas examiné par le système d'exploitation. Rien de particulier ne se produit donc lorsqu'un caractère Control-C apparaît. Comme l'entrée et la sortie standard peuvent être redirigées sur d'autres périphériques ou vers un fichier, les caractères sortis ne doivent pas nécessairement apparaître sur l'écran ni les caractères lus provenir obligatoirement du clavier. Toutefois, lorsque l'accès se fait sur un fichier, il est impossible pour le programme d'appel de détecter si tous les caractères de ce fichier ont déjà été lus ou bien si le support (disquette, disque dur) sur lequel figure ce fichier est déjà plein.

Pour l'entrée d'un caractère, la fonction n'attend pas qu'un caractère soit prêt mais revient immédiatement au programme d'appel dans tous les cas.

Entrée : AH = 06h
DL = 0 - 254 : Sortir ce caractère
DL = 255 : Lire un caractère

Sortie : Pour la sortie de caractères : aucune
Pour l'entrée de caractères :
Flag Zéro =1 : aucun caractère n'est prêt
Flag Zéro =0 : Le caractère entré figure dans le registre AL

Remarques :

- Lorsque des codes étendus sont lus, le code 0 est tout d'abord renvoyé dans le registre AL. La fonction doit être appelée à nouveau pour lire le code étendu lui-même.
- Le caractère de code ASCII 255 ne peut être sorti à l'aide de cette fonction puisqu'il est interprété comme d'entrée d'un caractère.
- Le contenu des registres AH, BX, CX, DX, SI, DI, BP, CS, DS, SS, ES et du registre de flags n'est pas modifié par cette fonction.

Interruption 21h, Fonction 07h DOS (à partir de 1.0)
Entrée de caractères directe sans sortie

Un caractère est lu sur le périphérique d'entrée standard mais sans être sorti sur le périphérique de sortie standard. Si aucun caractère n'est disponible au moment de l'appel de fonction, la fonction attend jusqu'à ce qu'un caractère soit disponible.

Le caractère reçu n'est pas examiné par le système d'exploitation, de sorte que rien ne se passe lorsqu'apparaît un caractère Control-C.

Comme l'entrée standard peut être redirigée sur un autre périphérique ou vers un fichier, le caractère lu ne doit pas nécessairement provenir du clavier. Si les caractères transmis proviennent d'un fichier, le programme d'appel n'a aucune possibilité de détecter si tous les caractères de ce fichier ont déjà été lus, autrement dit si la fin du fichier a ou non déjà été atteinte.

Entrée : AH = 07h

Sortie : AL = Le caractère lu

Remarques :
- Lorsque des codes étendus sont lus, le code 0 est tout d'abord renvoyé dans le registre AL. La fonction doit être appelée à nouveau pour lire le code étendu lui-même.
- Le contenu des registres AH, BX, CX, DX, SI, DI, BP, CS, DS, SS, ES et du registre de flags n'est pas modifié par cette fonction.

Interruption 21h, Fonction 08h DOS (à partir de 1.0)
Entrée de caractères sans sortie

Un caractère est lu sur le périphérique d'entrée standard mais sans être sorti sur le périphérique de sortie standard. Si aucun caractère n'est disponible au moment de l'appel de fonction, la fonction attend jusqu'à ce qu'un caractère soit disponible.

Comme l'entrée standard peut être redirigée sur un autre périphérique ou vers un fichier, le caractère lu ne doit pas nécessairement provenir du clavier. Si les caractères transmis proviennent d'un fichier, le programme d'appel n'a aucune possibilité de détecter si tous les caractères de ce fichier ont déjà été lus, autrement dit si la fin du fichier a ou non déjà été atteinte.

Entrée : AH = 08h

Sortie : AL = Le caractère lu

Remarques :

- Lorsque des codes étendus sont lus, le code 0 est tout d'abord renvoyé dans le registre AL. La fonction doit être appelée à nouveau pour lire le code étendu lui-même.
- Lorsque le caractère Control-C est détecté au cours de l'exécution de cette fonction, l'interruption 23h est appelée.
- Le contenu des registres AH, BX, CX, DX, SI, DI, BP, CS, DS, SS, ES et du registre de flags n'est pas modifié par cette fonction.

Interruption 21h, Fonction 09h DOS (à partir de 1.0)
Sortie d'une chaîne de caractères

Cette fonction permet de sortir une chaîne de caractères sur le périphérique de sortie standard. Comme ce périphérique standard peut être redirigé sur un autre périphérique ou vers un fichier, il n'y a aucune garantie que la chaîne de caractères apparaisse sur l'écran. Si la sortie est redirigée sur un fichier, le programme d'appel n'a aucune possibilité de détecter si le support (disquette, disque dur) sur lequel figure le fichier est déjà plein, autrement dit s'il est encore possible d'écrire la chaîne de caractères dans le fichier.

Entrée : AH = 09h
DS = Adresse de segment de la chaîne de caractères
DX = Adresse d'offset de la chaîne de caractères

Sortie : aucune

Remarques :

- La chaîne de caractères doit être stockée dans la mémoire sous forme d'une séquence d'octets correspondant aux codes ASCII des caractères composant la chaîne. La fin de la chaîne de caractères doit être signalée au DOS à l'aide d'un caractère "$" (code ASCII 36).
- Si la chaîne de caractères contient des codes de commande comme Backspace, Carriage Return ou Line Feed, ceux-ci seront traités comme tels.
- Seul le contenu du registre AL est modifié par l'appel de cette fonction.

Interruption 21h, Fonction 0Ah DOS (à partir de 1.0)
Entrée d'une chaîne de caractères

Cette fonction permet de lire un nombre déterminé de caractères sur le périphérique d'entrée standard et de les transférer dans un buffer. L'entrée se termine lorsque la

touche Return est actionnée. Le code de cette touche (13) est alors également placé dans le buffer, comme dernier caractère de la chaîne.

Comme l'entrée standard peut être redirigée sur un autre périphérique ou vers un fichier, les caractères entrés ne proviennent pas nécessairement du clavier. Si les caractères transmis proviennent d'un fichier, le programme d'appel n'a aucune possibilité de détecter si tous les caractères de ce fichier ont déjà été lus, autrement dit si la fin de fichier a déjà été atteinte.

Entrée : AH = 0Ah
DS = Adresse de segment du buffer
DX = Adresse d'offset du buffer

Sortie : aucune

Remarques :

- Le premier octet du buffer définit le nombre maximal de caractères (y compris le Carriage Return final) qui pourront être lus et placés dans le buffer à partir de la cellule de mémoire numéro deux. Ce paramètre doit être inscrit dans le buffer avant d'appeler la fonction, pour que celle-ci sache combien de caractères elle doit lire au maximum.
- Le DOS inscrit dans la cellule de mémoire 1, une fois le travail terminé, le nombre de caractères lus à l'exception du Carriage Return.
- Le buffer doit donc être d'une longueur égale au nombre de caractères à entrer plus 2 octets.
- Dès que l'avant-dernière cellule de mémoire du buffer est atteinte, l'entrée d'autres caractères entraîne l'émission d'un bip et seule la touche Return est encore acceptée pour conclure l'entrée.
- Les codes clavier étendus occupent deux octets dans le buffer. Le premier octet contient le code 0 et le second contient le code de la touche étendue.
- Lorsque le caractère Control-C est détecté au cours de l'entrée, l'interruption 23h est appelée.
- L'entrée peut être éditée à l'aide de la touche Backspace et des touches curseur, sans que ces touches soient sauvegardées dans le buffer.
- Cette fonction ne modifie le contenu d'aucun registre du processeur, y compris le registre de flags.

Interruption 21h, Fonction 0Bh DOS (à partir de 1.0)
Lire état d'entrée

Cette fonction permet de déterminer si des caractères sont prêts à être lus sur le périphérique d'entrée standard.

Entrée : AH = 0Bh

Sortie : AL = 0 : Aucun caractère disponible
AL = 255 : Un ou plusieurs caractères prêts à être lus

Remarques :
- Lorsque le caractère Control-C est détecté au cours de l'exécution de cette fonction, l'interruption 23h est appelée.
- Le contenu des registres AH, BX, CX, DX, SI, DI, BP, CS, DS, SS, ES et du registre de flags n'est pas modifié par cette fonction.

Interruption 21h, Fonction 0Ch DOS (à partir de 1.0)
Vider buffer d'entrée et appeler fonction d'entrée

Cette fonction vide tout d'abord le buffer d'entrée et appelle ensuite une des fonctions d'entrée de caractères. Comme les fonctions d'entrée de caractères tirent tous leurs caractères du périphérique d'entrée standard et que ce dernier n'est pas nécessairement le clavier, le fait de vider le buffer d'entrée ne présente d'intérêt que si le périphérique d'entrée standard est le clavier. Dans ce cas en effet, il se peut que des caractères aient été entrés avant appel de la fonction mais n'aient pas encore été lus par celle-ci. En vidant le buffer, on efface donc ces caractères pour avoir l'assurance que la fonction appelée ensuite ne recevra que des caractères entrés au cours de l'exécution de cette fonction.

Entrée : AH = 0Ch
AL = Numéro de la fonction d'appel
Pour l'appel de la fonction 10 :
DS = Adresse de segment du buffer d'entrée
DX = Adresse d'offset du buffer d'entrée

Sortie : Pour les fonctions 01h, 06h, 07h et 08h :
AL = le caractère entré
Pour la fonction 0Ah : aucune

Remarques :
- Les numéros de fonction 01h, 06h, 07h, 08 et 0Ah peuvent seuls être transmis à cette fonction comme fonctions à appeler.
- Seul le contenu du registre AL est modifié par l'appel de cette fonction.

Interruption 21h, Fonction 0Dh DOS (à partir de 1.0)
Réinitialisation des drivers de bloc

Cette fonction permet de transmettre au périphérique approprié (disquette, disque dur) toutes les données censées être transmises à un driver de bloc mais qui étaient encore stockées dans un buffer interne du DOS. Les fichiers ouverts (Handles ou FCB) ne sont pas refermés.

Entrée : AH = 0Dh

Sortie : aucune

Remarques :
- Même si cette fonction a été appelée, il faut encore refermer comme il convient tous les fichiers ouverts. Il peut arriver sinon que l'entrée de répertoire correspondant à un fichier donné n'ait pas été actualisée, auquel cas il ne sera pas possible d'accéder aux données nouvellement écrites dans le fichier.
- Cette fonction ne modifie le contenu d'aucun registre du processeur, y compris le registre de flags.

Interruption 21h, Fonction 0Eh DOS (à partir de 1.0)
Sélection du lecteur actuel

Cette fonction permet de définir le lecteur actuel dont la désignation apparaîtra désormais sur l'écran comme message de dialogue chaque fois que l'interpréteur de commandes attendra des entrées de l'utilisateur. Tous les accès de fichier pour lesquels aucune désignation de lecteur ne sera spécifiée se feront à l'avenir sur le lecteur indiqué ici.

Entrée : AH = 0Eh
DL = Code du lecteur actuel

Sortie : AL = Nombre de lecteurs (ou Volumes) installés

Remarques :
- Le lecteur A porte le code 0, B le code 1, etc...
- Même si un PC dispose seulement d'un lecteur de disquette et d'un disque dur, le nombre de volumes dans le registre AL peut être supérieur à 2 car le disque dur peut être subdivisé en plusieurs volumes et un ou plusieurs disques RAM peuvent également avoir été installés.
- Sur un PC disposant d'un seul lecteur de disquette, le nombre de volumes est de 2 car ce lecteur fait office aussi bien de lecteur A que de lecteur B.

- La version 2 du DOS autorisait 63 codes de périphériques différents alors que la version 3 du DOS a limité le nombre de périphériques à 26 (les lettres A à Z). Il n'est donc pas possible d'accéder à plus de 26 périphériques à la fois.
- Le nombre de lecteurs de disquette physiques n'est pas nécessairement identique au nombre de volumes. Il peut donc être déterminé de façon plus précise en appelant l'interruption 11h du BIOS.
- Le contenu des registres AH, BX, CX, DX, SI, DI, BP, CS, DS, SS, ES et du registre de flags n'est pas modifié par cette fonction.

Interruption 21h, Fonction 0Fh DOS (à partir de 1.0)
Ouvrir fichier (FCB)

Cette fonction permet d'ouvrir un fichier (s'il existe). Après exécution réussie de cette fonction, le fichier peut alors être lu ou écrit.

Entrée : AH = 0Fh
DS = Adresse de segment du FCB du fichier
DX = Adresse d'offset du FCB du fichier

Sortie : AL = 0 : Fichier trouvé et ouvert
AL = 255 : Fichier non trouvé

Remarques :
- Peuvent être utilisés aussi bien des FCB normaux que des FCB étendus.
- Si le fichier a été trouvé, le DOS inscrit dans le FCB la taille du fichier, la date et l'heure de sa création ou de sa dernière modification.
- Le DOS fixe la longueur d'enregistrement sur 128 octets. Cette longueur d'enregistrement peut être modifiée dans le FCB mais cela ne doit être fait qu'une fois le fichier ouvert. Pour travailler avec une longueur d'enregistrement supérieure, il est en outre nécessaire de déplacer la DTA car la DTA prédéfinie a une taille de 128 octets seulement.
- Pour effectuer un accès sélectif au fichier, il convient de fixer le champ approprié du FCB après ouverture réussie du fichier.
- Le pointeur de fichier est dirigé sur le premier octet du fichier après ouverture de celui-ci.
- Le contenu des registres AH, BX, CX, DX, SI, DI, BP, CS, DS, SS, ES et du registre de flags n'est pas modifié par cette fonction.

Interruption 21h, Fonction 10h DOS (à partir de 1.0)
Fermer fichier (FCB)

Après appel de cette fonction, toutes les données qui figurent encore dans un des buffers internes du DOS sont écrites dans le fichier qui est ensuite refermé. L'entrée du répertoire correspondant à ce fichier est alors actualisée pour tenir compte de la nouvelle taille du fichier ainsi que des date et heure de modification.

Entrée : AH = 10h
DS = Adresse de segment du FCB du fichier
DX = Adresse d'offset du FCB du fichier

Sortie : AL = 0 : Fichier refermé et entrée du répertoire actualisée
AL = 255 : Fichier non trouvé dans le répertoire

Remarques :
- Seuls peuvent naturellement être refermés des fichiers qui ont été précédemment ouverts.
- Pour les fichiers sur disquette, il faut absolument s'assurer que la disquette placée dans le lecteur au moment de l'appel de cette fonction est bien la disquette qui contient le fichier. S'il n'en est pas ainsi, une FAT et un répertoire erronés seront écrits sur la disquette, ce qui rendra inutilisables les données qu'elle contenait.
- Le contenu des registres AH, BX, CX, DX, SI, DI, BP, CS, DS, SS, ES et du registre de flags n'est pas modifié par cette fonction.

Interruption 21h, Fonction 11h DOS (à partir de 1.0)
Rechercher première entrée du répertoire (FCB)

Cette fonction permet de rechercher la première apparition d'un nom de fichier déterminé dans le répertoire actuel du périphérique désigné dans le FCB.

Entrée : AH = 11h
DS = Adresse de segment du FCB
DX = Adresse d'offset du FCB

Sortie : AL = 0 : Fichier trouvé
AL = 255 : Fichier non trouvé

Remarques :
- Le FCB transmis à la fonction contient la désignation du périphérique sur lequel le fichier doit être recherché ainsi que le nom de ce fichier.
- Le nom de fichier peut contenir le joker "?" si vousvoulez faire rechercher un groupe de fichiers.

- La recherche s'effectue exclusivement sur le répertoire actuel du périphérique spécifié.
- Si c'est un fichier normal qu'il s'agit de rechercher, un FCB normal doit être transmis à la fonction. S'il s'agit au contraire de rechercher des fichiers dotés d'attributs particuliers (noms de volumes, sous-répertoires, fichiers cachés, etc...), il faut travailler avec des FCB étendus.
- Si un fichier a été trouvé, la DTA contient un FCB de même type que le FCB transmis, qui contient le nom du fichier trouvé. C'est pourquoi la DTA doit toujours être d'une taille lui permettant de recevoir au moins un FCB normal ou étendu.
- La DTA peut être déplacée vers un buffer utilisateur à l'aide de la fonction 1Ah pour s'assurer qu'elle sera d'une taille suffisante pour recevoir le FCB correspondant.
- Le contenu des registres AH, BX, CX, DX, SI, DI, BP, CS, DS, SS, ES et du registre de flags n'est pas modifié par cette fonction.

Interruption 21h, Fonction 12h DOS (à partir de 1.0)
Rechercher prochaine entrée du répertoire (FCB)

Après que le premier nom de fichier ait été identifié en appelant la fonction 17, cette fonction vous permet de faire rechercher tous les autres noms de fichiers (s'il y en a).

Entrée : AH = 12h
DS = Adresse de segment du FCB
DX = Adresse d'offset du FCB

Sortie : AL = 0 : Fichier trouvé
AL = 255 : Fichier non trouvé (pas d'autres fichiers présents)

Remarques :
- Cette fonction ne doit être appelée qu'après que la fonction 11h ait été tout d'abord appelée.
- Si un autre nom de fichier a été trouvé, son nom figure à nouveau dans le FCB placé au début de la DTA.
- La recherche s'effectue exclusivement dans le répertoire actuel du périphérique spécifié.
- La DTA peut être déplacée à l'aide de la fonction 1Ah vers un buffer utilisateur pour garantir que ce buffer soit suffisamment grand pour recevoir le FCB approprié.
- Le contenu des registres AH, BX, CX, DX, SI, DI, BP, CS, DS, SS, ES et du registre de flags n'est pas modifié par cette fonction.

Interruption 21h, Fonction 13h DOS (à partir de 1.0)
Supprimer fichier(s) (FCB)

Cette fonction sert à supprimer un ou plusieurs fichiers dans le répertoire actuel du périphérique spécifié.

Entrée : AH = 13h
DS = Adresse de segment du FCB
DX = Adresse d'offset du FCB

Sortie : AL = 0 : Fichier(s) supprimé(s)
AL = 255 : Aucun fichier trouvé ou bien les fichiers trouvés ne pouvaient être supprimés (parce qu'ils étaient dotés de l'attribut "Lecture seule")

Remarques :

- Le FCB transmis à la fonction contient la désignation du périphérique sur lequel figurent les fichiers à supprimer ainsi que le nom de ces fichiers.
- Le nom de fichier peut contenir le joker "?" si vous voulez supprimer un groupe de fichiers.
- Seuls des fichiers du répertoire actuel du périphérique spécifié peuvent être supprimés.
- Si c'est un fichier normal qu'il s'agit de supprimer, un FCB normal doit être transmis à la fonction. S'il s'agit au contraire de supprimer des fichiers dotés d'attributs particuliers (noms de volumes, fichiers cachés, etc...), il faut travailler avec des FCB étendus.
- Des noms de volumes peuvent être supprimés à l'aide de cette fonction mais pas des sous-répertoires.
- Le contenu des registres AH, BX, CX, DX, SI, DI, BP, CS, DS, SS, ES et du registre de flags n'est pas modifié par cette fonction.

Interruption 21h, Fonction 14h DOS (à partir de 1.0)
Lecture séquentielle (FCB)

Lire le prochain enregistrement d'un fichier.

Entrée : AH = 14h
DS = Adresse de segment du FCB
DX = Adresse d'offset du FCB

Sortie : AL = 0 : Enregistrement a été lu
AL = 1 : Fin du fichier atteinte

AL = 2 : Débordement de segment
AL = 3 : Lu enregistrement partiel

Remarques :

- Cette fonction ne doit être appelée qu'après que le fichier ait été ouvert à travers le FCB indiqué.
- L'enregistrement est transféré dans la DTA. Si celle-ci n'est pas assez grande, il convient de la déplacer auparavant vers un buffer utilisateur à l'aide de la fonction 1Ah.
- La taille de l'enregistrement, c'est-à-dire le nombre d'octets lus, est inscrit dans le FCB.
- L'erreur 2 se produit si la DTA est située à la fin d'un segment et que l'enregistrement lu est conduit, de ce fait, à dépasser la limite du segment.
- L'erreur 3 se produit lorsque la fin d'un fichier ne comporte pas un enregistrement complet mais seulement un enregistrement partiel. Dans ce cas, l'enregistrement partiel est lu mais comblé de zéros jusqu'à la longueur prescrite.
- Après que l'enregistrement ait été lu, le pointeur de fichier est dirigé sur le début du prochain enregistrement, de sorte que c'est automatiquement le prochain enregistrement qui sera lu lors du prochain appel de cette fonction.
- Le contenu des registres AH, BX, CX, DX, SI, DI, BP, CS, DS, SS, ES et du registre de flags n'est pas modifié par cette fonction.

Interruption 21h, Fonction 15h — DOS (à partir de 1.0)
Ecriture séquentielle (FCB)

Ecrire le prochain enregistrement dans un fichier.

Entrée : AH = 15h
DS = Adresse de segment du FCB
DX = Adresse d'offset du FCB

Sortie : AL = 0 : Enregistrement a été écrit
AL = 1 : Support (disquette/disque dur) plein
AL = 2 : Débordement de segment

Remarques :

- Cette fonction ne doit être appelée qu'après que le fichier ait été ouvert à travers le FCB indiqué.
- L'enregistrement est transféré de la DTA dans le fichier. Si la DTA n'est pas assez grande, il convient de la déplacer auparavant vers un buffer utilisateur à l'aide de la fonction 1Ah.

- La taille de l'enregistrement, c'est-à-dire le nombre d'octets écrits, est inscrit dans le FCB.
- L'erreur 2 se produit si la DTA est située à la fin d'un segment et que l'enregistrement à écrire est conduit, de ce fait, à dépasser la limite du segment.
- Après que l'enregistrement ait été écrit, le pointeur de fichier est dirigé sur le début du prochain enregistrement, de sorte que c'est automatiquement le prochain enregistrement qui sera écrit lors du prochain appel de cette fonction.
- Le contenu des registres AH, BX, CX, DX, SI, DI, BP, CS, DS, SS, ES et du registre de flags n'est pas modifié par cette fonction.

Interruption 21h, Fonction 16h DOS (à partir de 1.0)
Créer ou vider un fichier (FCB)

Après appel de cette fonction, un fichier n'existant pas encore sera créé. Un fichier déjà existant sera vidé par l'appel de cette fonction, la longueur de ce fichier passant donc à 0. Le fichier est en même temps ouvert par cette fonction, de sorte que ce fichier pourra être écrit et ensuite lu en appelant d'autres fonctions.

Entrée : AH = 16h
DS = Adresse de segment du FCB
DX = Adresse d'offset du FCB

Sortie : AL = 0 : Fichier a été créé ou vidé
AL = 255 : Fichier n'a pu être créé (par exemple parce que le répertoire était déjà plein)

Remarques :
- Le contenu d'un fichier déjà existant qui aura été vidé à l'aide de cette fonction sera irrémédiablement perdu.
- Après appel de cette fonction, il n'est plus nécessaire d'ouvrir le fichier à l'aide de la fonction 0Fh.
- Si le fichier est ouvert à l'aide d'un FCB étendu, il peut aussi posséder certains attributs (nom de volume, fichier caché). Il n'est toutefois pas possible de créer un sous-répertoire à l'aide de cette fonction.
- Après ouverture du fichier, le pointeur de fichier est dirigé sur le premier octet du fichier.
- Le contenu des registres AH, BX, CX, DX, SI, DI, BP, CS, DS, SS, ES et du registre de flags n'est pas modifié par cette fonction.

Interruption 21h, Fonction 17h DOS (à partir de 1.0)
Renommer fichier(s) (FCB)

Renommer un fichier ou un groupe de fichiers dans le répertoire actuel sur le périphérique spécifié.

Entrée : AH = 17h
DS = Adresse de segment du FCB
DX = Adresse d'offset du FCB

Sortie : AL = 0 : Fichier(s) renommé(s)
AL = 255 :

Remarques :

- Le FCB transmis doit être un FCB spécial. Il est basé sur un FCB normal dont les 12 premiers octets contiennent comme d'habitude la désignation de périphérique et le nom du fichier à renommer. Ce FCB a cependant ceci de particulier qu'il comporte à partir de la cellule de mémoire 16 la désignation de périphérique et le nouveau nom du fichier. La désignation de périphérique doit naturellement être identique à celle spécifiée pour l'ancien nom du fichier.
- Le nom du fichier à renommer peut contenir le joker "?" si vous voulez renommer plusieurs fichiers à la fois.
- Si le nouveau nom du fichier contient également le joker "?", les emplacements du nom de fichier ou de l'extension de fichier pour lesquels figure un "?" dans le nouveau nom de fichier ne seront pas modifiés.
- A partir de la version 2.0, un FCB augmenté peut être spécifié pour renommer un répertoire simultanément avec cette fonction.
- Le contenu des registres AH, BX, CX, DX, SI, DI, BP, CS, DS, SS, ES et du registre de flags n'est pas modifié par cette fonction.

Interruption 21h, Fonction 18h DOS (à partir de 1.0)

Réservée à un usage interne

Interruption 21h, Fonction 19h DOS (à partir de 1.0)
Obtenir la désignation de périphérique du lecteur actuel

L'appel de cette fonction renvoie la désignation de périphérique du lecteur actuel.

Entrée : AH = 19h

Sortie : AL = Désignation de périphérique

Remarques :

- Alors que pour d'autres fonctions DOS la désignation de périphérique 0 correspond au périphérique actuel et que c'est le code 1 qui désigne le lecteur A, c'est ici le code 0 qui désigne le lecteur A, le code 1 désignant le lecteur B, etc...
- Le contenu des registres AH, BX, CX, DX, SI, DI, BP, CS, DS, SS, ES et du registre de flags n'est pas modifié par cette fonction.

Interruption 21h, Fonction 1Ah DOS (à partir de 1.0)
Fixer l'adresse de la DTA

En appelant cette fonction, vous pouvez déplacer vers une autre zone de mémoire la zone de transfert disque (Disk Transfer Area = DTA), qui est utilisée pour le stockage intermédiaire des données lors de tous les accès de fichier soutenus par FCB.

Entrée : AH = 1Ah
DS = Adresse de segment de la nouvelle DTA
DX = Adresse d'offset de la nouvelle DTA

Sortie : aucune

Remarques :

- Cette fonction doit être appelée chaque fois que la DTA prédéfinie n'est pas assez grande pour recevoir les données à transférer.
- Lors du lancement d'un programme, le DOS place la DTA à l'adresse 128 du PSP du programme. Elle a donc une taille de 128 octets puisque le programme lui-même commence immédiatement après l'adresse 255 du PSP.
- Le DOS ne connaît pas la longueur de la DTA et il considère toujours a priori qu'elle est suffisamment grande pour absorber les données à transférer. Or s'il n'en est pas ainsi en réalité, les données excédentaires effaceront les données suivantes de la fonction DOS utilisée.
- Avec différentes fonctions, le DOS détecte toutefois une erreur lorsque la DTA se situe à la fin d'un segment et que les données à transférer sont de ce fait conduites à dépasser la limite du segment.
- Cette fonction ne modifie le contenu d'aucun registre du processeur, y compris le registre de flags.

Interruption 21h, Fonction 1Bh DOS (à partir de 1.0)
Lire informations sur le lecteur actuel

Cette fonction permet d'obtenir un certain nombre d'informations sur le format du lecteur actuel.

Entrée : AH = 1Bh

Sortie :
- AL = Nombre de secteurs par cluster
- DS = Adresse de segment du descripteur de support
- BX = Adresse d'offset du descripteur de support
- DX = Nombre de clusters

Remarques :

- Pour le descripteur de support peuvent être renvoyés les codes suivants :
  - F0h : Lecteur de disquette : deux faces, 80 pistes, 18 secteurs par piste (3 pouces 1/2)
  - F8h : Disque dur
  - F9h : Lecteur de disquette : deux faces, 15 secteurs par piste (5 pouces 1/4, AT seulement) ou deux faces, 80 pistes, 9 secteurs par piste (3 pouces 1/2)
  - FAh : Lecteur de disquette : une face, 80 pistes, 8 secteurs par piste (5 ou 3 pouces 1/2)
  - FBh : Lecteur de disquette : deux faces, 80 pistes, 8 secteurs par piste (5 ou 3 pouces 1/2)
  - FCh : Lecteur de disquette : une face, 40 pistes, 9 secteurs par piste (5 pouces 1/4)
  - FDh : Lecteur de disquette : deux faces, 40 pistes, 9 secteurs par piste (5 pouces 1/4)
  - FEh : Lecteur de disquette : une face, 40 pistes, 8 secteurs par piste (5 pouces 1/4)
  - FFh : Lecteur de disquette : deux faces, 40 pistes, 8 secteurs par piste (5 pouces 1/4)
- La fonction 1Ch vous permet d'obtenir des informations sur n'importe quel lecteur.
- Le contenu des registres AH, CX, SI, DI, BP, CS, DS, SS, ES et du registre de flags n'est pas modifié par cette fonction.

Interruption 21h, Fonction 1Ch DOS (à partir de 1.0)
Lire informations sur un lecteur quelconque

Cette fonction permet d'obtenir un certain nombre d'informations sur le format du lecteur spécifié.

Entrée : AH = 1Ch
DL = Désignation de périphérique

Sortie : AL = Nombre de secteurs par cluster
DS = Adresse de segment du descripteur de support
BX = Adresse d'offset du descripteur de support
DX = Nombre de clusters

Remarques :

- Pour la désignation de périphérique, 0 représente le lecteur actuel, 1 le lecteur A, 2 le lecteur B, etc...
- Pour le descripteur de support peuvent être renvoyés les codes suivants :

  F0h : Lecteur de disquette : deux faces, 80 pistes, 18 secteurs par piste (3 pouces 1/2)
  F8h : Disque dur
  F9h : Lecteur de disquette : deux faces, 15 secteurs par piste (5 pouces, AT seulement) ou deux faces, 80 pistes, 9 secteurs par piste (3 pouces 1/2)
  FAh : Lecteur de disquette : une face, 80 pistes, 8 secteurs par piste (5 ou 3 pouces 1/2)
  FBh : Lecteur de disquette : deux faces, 80 pistes, 8 secteurs par piste (5 ou 3 pouces 1/2)
  FCh : Lecteur de disquette : une face, 40 pistes, 9 secteurs par piste (5 pouces 1/4)
  FDh : Lecteur de disquette : deux faces, 40 pistes, 9 secteurs par piste (5 pouces 1/4)
  FEh : Lecteur de disquette : une face, 40 pistes, 8 secteurs par piste (5 pouces 1/4)
  FFh : Lecteur de disquette : deux faces, 40 pistes, 8 secteurs par piste (5 pouces 1/4)

- Le contenu des registres AH, CX, SI, DI, BP, CS, DS, SS, ES et du registre de flags n'est pas modifié par cette fonction.

Interruption 21h, Fonction 1Dh — DOS (à partir de 1.0)

Réservée à un usage interne

Interruption 21h, Fonction 1Eh — DOS (à partir de 1.0)

Réservée à un usage interne

Interruption 21h, Fonction 1Fh DOS (à partir de 1.0)
Lire le pointeur BPD pour le lecteur en cours

A travers l'appel de cette fonction, DOS place un pointeur sur le bloc de paramètres DOS (BPD) du lecteur en cours. Le BPD décrit la structure logique et physique du lecteur.

Entrée : AH = 1Fh

Sortie : AL = 0 : Pas d'erreur, dans ce cas
DS:BX = Pointeur FAR sur BPD

Remarques :
- S'il s'agit d'un changement de disquette effectué sur le périphérique en cours, DOS s'adresse d'abord au lecteur pour compléter le BPD avec les données concernant le lecteur et le format de la disquette. En revanche, pour le disque dur, ces informations se trouvent déjà dans la mémoire si bien qu'il n'est pas nécessaire d'effectuer un accès au périphérique.
- La structure du bloc de paramètres DOS varie en fonction des versions DOS. Reportez-vous au chapitre XXX.
- Le contenu des registres AH, CX, DX, SI, DI, BP, CS, DS, SS, ES et du registre de flags n'est pas modifié par cette fonction.

Interruption 21h, Fonction 21h DOS (à partir de 1.0)
Lecture sélective (FCB)

Lire un enregistrement déterminé d'un fichier en le transférant dans la DTA.

Entrée : AH = 21h
DS = Adresse de segment du FCB
DX = Adresse d'offset du FCB

Sortie : AL = 0 : Enregistrement a été lu
AL = 1 : Fin du fichier atteinte
AL = 2 : Débordement de segment
AL = 3 : Lu enregistrement partiel

Remarques :
- Cette fonction ne doit être appelée qu'après que le fichier ait été ouvert à travers le FCB indiqué.
- L'enregistrement est transféré dans la DTA. Si celle-ci n'est pas assez grande, il convient de la déplacer auparavant vers un buffer utilisateur à l'aide de la fonction 1Ah.

- La taille de l'enregistrement, c'est-à-dire le nombre d'octets lus, est inscrit dans le FCB.
- Après appel de cette fonction, le pointeur de fichier est dirigé automatiquement sur le début de l'enregistrement à lire, de sorte qu'un appel consécutif de la fonction de lecture séquentielle d'un enregistrement lirait à nouveau le même enregistrement.
- Le numéro d'enregistrement n'est pas incrémenté après appel de la fonction, de sorte qu'un nouvel appel de cette fonction lirait le même enregistrement.
- L'erreur 2 se produit si la DTA est située à la fin d'un segment et que l'enregistrement lu est conduit, de ce fait, à dépasser la limite du segment.
- L'erreur 3 se produit lorsque la fin d'un fichier ne comporte pas un enregistrement complet mais seulement un enregistrement partiel. Dans ce cas, l'enregistrement partiel est lu mais comblé de zéros jusqu'à la longueur prescrite.
- Le contenu des registres AH, BX, CX, DX, SI, DI, BP, CS, DS, SS, ES et du registre de flags n'est pas modifié par cette fonction.

Interruption 21h, Fonction 22h DOS (à partir de 1.0)
Ecriture sélective (FCB)

Ecrire un enregistrement déterminé dans le fichier.

Entrée : AH = 22h
DS = Adresse de segment du FCB
DX = Adresse d'offset du FCB

Sortie : AL = 0 : Enregistrement a été écrit
AL = 1 : Support (disquette/disque dur) plein
AL = 2 : Débordement de segment

Remarques :
- Cette fonction ne doit être appelée qu'après que le fichier ait été ouvert à travers le FCB indiqué.
- L'adresse de l'enregistrement à écrire doit être spécifiée à partir de la cellule de mémoire 21h du FCB.
- La taille de l'enregistrement, c'est-à-dire le nombre d'octets écrits, est inscrit dans le FCB.
- L'enregistrement est transféré de la DTA dans le fichier. Si celle-ci n'est pas assez grande, il convient de la déplacer auparavant vers un buffer utilisateur à l'aide de la fonction 1Ah.

- Après appel de cette fonction, le pointeur de fichier est dirigé automatiquement sur le début de l'enregistrement à écrire, de sorte qu'un appel consécutif de la fonction d'écriture séquentielle d'un enregistrement écrirait à nouveau le même enregistrement.
- Le numéro d'enregistrement n'est pas incrémenté après appel de la fonction, de sorte qu'un nouvel appel de cette fonction écrirait le même enregistrement.
- L'erreur 2 se produit si la DTA est située à la fin d'un segment et que l'enregistrement à écrire est conduit, de ce fait, à dépasser la limite du segment.
- Le contenu des registres AH, BX, CX, DX, SI, DI, BP, CS, DS, SS, ES et du registre de flags n'est pas modifié par cette fonction.

Interruption 21h, Fonction 23h DOS (à partir de 1.0)
Lire taille du fichier (FCB)

Cette fonction permet de connaître le nombre d'enregistrements et donc, de ce fait, la taille d'un fichier.

Entrée : AH = 23h
DS = Adresse de segment du FCB
DX = Adresse d'offset du FCB

Sortie : AL = 0 : Fichier trouvé, dans ce cas le champ d'enregistrement à partir de l'adresse 21h du FCB contient le nombre d'enregistrements dans le fichier.
AL = 255 : Fichier non trouvé

Remarques :
- Le FCB transmis contient la désignation de périphérique ainsi que le nom et l'extension du fichier dont il s'agit de déterminer la taille.
- Contrairement à tous les autres accès de fichier soutenus par FCB, la taille d'enregistrement doit être inscrite dans le FCB avant appel de cette fonction.
- Si la taille d'enregistrement spécifiée est 1, cette fonction permet d'obtenir directement et précisément la taille du fichier en octets.
- Le contenu des registres AH, BX, CX, DX, SI, DI, BP, CS, DS, SS, ES et du registre de flags n'est pas modifié par cette fonction.

Interruption 21h, Fonction 24h DOS (à partir de 1.0)
Fixer numéro d'enregistrement

Cette fonction permet de fixer sur la position actuelle du pointeur de fichier le numéro d'enregistrement dans le FCB qui définit l'accès séquentiel à un fichier. Après une série d'accès de fichier séquentiels, on peut ainsi poursuivre avec l'accès sélectif à l'endroit précis où les accès séquentiels avaient été interrompus.

Entrée : AH = 24h
DS = Adresse de segment du FCB
DX = Adresse d'offset du FCB

Sortie : aucune

Remarques :
- Cette fonction ne doit être appelée qu'après que le fichier ait été ouvert à travers le FCB spécifié.
- Cette fonction ne modifie le contenu d'aucun registre du processeur, y compris le registre de flags.

Interruption 21h, Fonction 25h DOS (à partir de 1.0)
Fixer vecteur d'interruption

Cette fonction permet de rediriger n'importe quel vecteur d'interruption sur une autre routine.

Entrée : AH = 25h
AL = Numéro d'interruption
DS = Nouvelle adresse de segment de la routine d'interruption
DX = Nouvelle adresse d'offset de la routine d'interruption

Sortie : aucune

Remarques :
- Avant d'appeler cette fonction, il convient tout d'abord de lire à l'aide de la fonction 35h l'ancien contenu du vecteur d'interruption à modifier et de le sauvegarder. Une fois le programme terminé, il sera préférable de rétablir cet ancien contenu à l'aide de cette même fonction.
- Cette fonction ne modifie le contenu d'aucun registre du processeur, y compris le registre de flags.

Interruption 21h, Fonction 26h DOS (à partir de 1.0)
Créer un nouveau PSP

Le PSP du programme exécuté est copié à l'adresse spécifiée.

Entrée : AH = 26h
DX = Adresse de segment du nouveau PSP

Sortie : aucune

Remarques :
- L'adresse d'offset du nouveau PSP est 0.
- Cette fonction était utilisée sous la version 1 du DOS pour exécuter d'autres programmes. A cet effet, un PSP était mis en place, après quoi le programme voulu pouvait être chargé à la suite de ce PSP puis être exécuté.
- Sous les versions 2 et 3 du DOS, il est préférable de ne plus recourir à cette fonction car les autres programmes peuvent être chargés et exécutés à l'aide de la fonction EXEC 4Bh.
- Cette fonction ne modifie le contenu d'aucun registre du processeur, y compris le registre de flags.

Interruption 21h, Fonction 27h DOS (à partir de 1.0)
Lecture sélective de plusieurs enregistrements (FCB)

Lecture de plusieurs enregistrements consécutifs du fichier, à partir de l'enregistrement spécifié.

Entrée : AH = 27h
CX = Nombre d'enregistrements à lire
DS = Adresse de segment du FCB
DX = Adresse d'offset du FCB

Sortie : AL = 0 : Enregistrement(s) a (ont) été lu(s)
AL = 1 : Fin du fichier atteinte
AL = 2 : Débordement de segment
AL = 3 : Lu enregistrement partiel
CX = Nombre d'enregistrements lus

Remarques :
- Cette fonction ne doit être appelée qu'après que le fichier ait été ouvert à travers le FCB indiqué.
- Le premier enregistrement lu est celui dont l'adresse est spécifiée à partir de la cellule de mémoire 21h du FCB.

- Les enregistrements sont transférés dans la DTA. Si celle-ci n'est pas assez grande, il convient de la déplacer auparavant vers un buffer utilisateur à l'aide de la fonction 1Ah.
- La taille de chaque enregistrement est inscrite dans le FCB.
- Après appel de cette fonction, le pointeur de fichier est dirigé automatiquement à la suite du dernier enregistrement lu. Le numéro d'enregistrement est en outre incrémenté à raison du nombre d'enregistrements lus, de sorte qu'il désigne l'enregistrement suivant le dernier enregistrement lu.
- L'erreur 2 se produit si la DTA est située à la fin d'un segment et que l'enregistrement lu est conduit, de ce fait, à dépasser la limite du segment.
- L'erreur 3 se produit lorsque la fin d'un fichier ne comporte pas un enregistrement complet mais seulement un enregistrement partiel. Dans ce cas, l'enregistrement partiel est lu mais comblé de zéros jusqu'à la longueur prescrite.
- Le contenu des registres AH, BX, DX, SI, DI, BP, CS, DS, SS, ES et du registre de flags n'est pas modifié par cette fonction.

Interruption 21h, Fonction 28h DOS (à partir de 1.0)
Ecriture sélective de plusieurs enregistrements (FCB)

Ecriture de plusieurs enregistrements consécutifs dans un fichier, à partir de l'enregistrement spécifié.

Entrée :
- AH = 28h
- CX = Nombre d'enregistrements à écrire
- DS = Adresse de segment du FCB
- DX = Adresse d'offset du FCB

Sortie :
- AL = 0 : Enregistrement(s) a (ont) été écrit(s)
- AL = 1 : Support (disquette/disque dur) plein
- AL = 2 : Débordement de segment
- CX = Nombre d'enregistrements écrits

Remarques :
- Cette fonction ne doit être appelée qu'après que le fichier ait été ouvert à travers le FCB indiqué.
- L'adresse du premier enregistrement à écrire doit être spécifiée à partir de la cellule de mémoire 21h du FCB.
- La taille de chaque enregistrement est inscrite dans le FCB.

- Les enregistrements sont transférés de la DTA dans le fichier. Si celle-ci n'est pas assez grande, il convient de la déplacer auparavant vers un buffer utilisateur à l'aide de la fonction 1Ah.
- Après appel de cette fonction, le pointeur de fichier est dirigé automatiquement à la suite du dernier enregistrement écrit. Le numéro d'enregistrement est en outre incrémenté à raison du nombre d'enregistrements écrits, de sorte qu'il désigne l'enregistrement suivant le dernier enregistrement écrit.
- L'erreur 2 se produit si la DTA est située à la fin d'un segment et que l'enregistrement à écrire est conduit, de ce fait, à dépasser la limite du segment.
- Le contenu des registres AH, BX, DX, SI, DI, BP, CS, DS, SS, ES et du registre de flags n'est pas modifié par cette fonction.

Interruption 21h, Fonction 29h DOS (à partir de 1.0)
Inscrire un nom de fichier dans FCB

Cette fonction inscrit dans les champs appropriés du FCB un nom de fichier fourni sous forme de chaîne ASCII et qui peut contenir une désignation de périphérique, un nom de fichier et une extension de fichier.

Entrée :
- AH = 29h
- DS = Adresse de segment du nom de fichier dans la mémoire
- SI = Adresse d'offset du nom de fichier dans la mémoire
- ES = Adresse de segment du FCB
- DI = Adresse d'offset du FCB
- AL = Paramètres de transmission
  - Bit 1 = 1 : La désignation de périphérique dans le FCB ne sera modifiée que si le nom de fichier transmis contient une désignation de périphérique.
  - 0: La désignation de périphérique sera modifiée en tout état de cause. Si le nom de fichier transmis ne comporte pas de désignation de périphérique, la valeur 0 (lecteur actuel) sera inscrite dans le FCB.
  - Bit 2 = 1 : Le nom de fichier dans le FCB ne sera modifié que si le nom de fichier transmis contient un nom de fichier.
  - 0: Le nom de fichier sera modifié en tout état de cause. Si le nom de fichier transmis ne comporte pas de nom de fichier, le nom de fichier sera rempli d'espaces (code ASCII 32) dans le

FCB.

Bit 3 = 1 : L'extension de fichier dans le FCB ne sera modifiée que si le nom de fichier transmis contient une extension de fichier.

0: L'extension de fichier sera modifiée en tout état de cause. Si le nom de fichier transmis ne comporte pas d'extension de fichier, ce champ sera rempli d'espaces (code ASCII 32) dans le FCB.

Tous les autres bits doivent contenir la valeur 0.

Sortie :
- AL = 0 : Le nom de fichier transmis ne comportait pas de jokers
- AL = 1 : Le nom de fichier transmis comportait des jokers
- AL = 255 : La désignation de périphérique spécifiée est incorrecte.
- DS = Adresse de segment du premier caractère après le nom de fichier dans le buffer de nom de fichier spécifié
- SI = Adresse d'offset du premier caractère après le nom de fichier dans le buffer de nom de fichier spécifié
- ES = Adresse de segment du FCB
- DI = Adresse d'offset du FCB

Remarques :
- Le nom de fichier transmis doit être terminé par un caractère de fin (code ASCII 0).
- Si le nom de fichier transmis comporte le joker "*", tous les emplacements à partir de la position de "*" seront remplis de jokers "?" dans le champ correspondant du FCB.
- Le contenu des registres AH, BX, CX, DX, BP, CS, SS et du registre de flags n'est pas modifié par cette fonction.

Interruption 21h, Fonction 2Ah DOS (à partir de 1.0)
Lire la date

Cette fonction permet de déterminer la date actuelle.

Entrée : AH = 2Ah

Sortie :
- AL = Jour de la semaine (0=Dimanche, 1=Lundi, etc...)
- CX = Année
- DH = Mois
- DL = Jour

Remarques :
- Pour obtenir la date, le DOS appelle le driver d'horloge.
- Le contenu des registres AH, BX, SI, DI, BP, CS, DS, SS, ES et du registre de flags n'est pas modifié par cette fonction.

Interruption 21h, Fonction 2Bh DOS (à partir de 1.0)
Fixer la date

Cette fonction permet de fixer la date actuelle.

Entrée : AH = 2Bh
CX = Année
DH = Mois
DL = Jour

Sortie : AL = 0 : Tout va bien
AL = 255 : Date impossible

Remarques :
- La date transmise est communiquée au driver d'horloge.
- Si le PC n'est pas doté d'une horloge en temps réel sur piles ainsi que d'un driver d'horloge soutenant ce type d'horloge, la date ne sera conservée que jusqu'à l'arrêt de l'ordinateur ou jusqu'à un nouveau lancement du système.
- L'ancienne heure est conservée si l'heure n'est pas plausible.
- Le contenu des registres AH, BX, CX, DX, SI, DI, BP, CS, DS, SS, ES et du registre de flags n'est pas modifié par cette fonction.

Interruption 21h, Fonction 2Ch DOS
Lire l'heure

Cette fonction permet de déterminer l'heure actuelle.

Entrée : AH = 2Ch

Sortie : CH = Heures
CL = Minutes
DH = Secondes
DL = Centièmes de seconde

Remarques :
- Pour obtenir l'heure, le DOS appelle le driver d'horloge.
- Le contenu des registres AX, BX, SI, DI, BP, CS, DS, SS, ES et du registre de flags n'est pas modifié par cette fonction.

Interruption 21h, Fonction 2Dh DOS (à partir de 1.0)
Fixer l'heure

Cette fonction permet de fixer l'heure actuelle.

Entrée : AH = 2Dh
CH = Heures
CL = Minutes
DH = Secondes
DL = Centièmes de seconde

Sortie : AL = 0 : Tout va bien
AL = 255 : Heure impossible

Remarques :
- L'heure transmise est communiquée au driver d'horloge.
- Si le PC n'est pas doté d'une horloge en temps réel sur piles ainsi que d'un driver d'horloge soutenant ce type d'horloge, l'heure ne sera conservée (et actualisée) que jusqu'à l'arrêt de l'ordinateur ou jusqu'à un nouveau lancement du système.
- L'ancienne heure est conservée si l'heure n'est pas plausible.
- Le contenu des registres AH, BX, CX, DX, SI, DI, BP, CS, DS, SS, ES et du registre de flags n'est pas modifié par cette fonction.

Interruption 21h, Fonction 2Eh DOS (à partir de 1.0)
Fixer le flag Verify

Le flag Verify définit si après une opération d'écriture sur un driver de bloc (disquette, disque dur) il convient de vérifier si les données écrites ont été correctement transmises. Cette vérification ralentit naturellement l'accès aux périphériques concernés mais accroît la sécurité de transmission des données.

Entrée : AH = 2Eh
DL = 0
AL = 0 : Ne pas vérifier les données
AL = 1 : Vérifier les données

Sortie : aucune

Remarques :
- Au niveau utilisateur, ce flag peut également être manipulé à l'aide des instructions VERIFY ON et VERIFY OFF.
- Cette fonction ne modifie le contenu d'aucun registre du processeur, y compris le registre de flags.

Interruption 21h, Fonction 2Fh DOS (à partir de 1.0)
Déterminer adresse de la DTA

Cette fonction fournit en résultat l'adresse de la zone de transfert disque (DTA) qui sert de buffer de données pour tous les accès de fichier soutenus par FCB.

Entrée : AH = 2Fh

Sortie : ES = Adresse de segment de la DTA
BX = Adresse d'offset de la DTA

Remarques :
- Cette fonction permet bien de déterminer l'adresse de la DTA mais cela ne permet nullement d'en déduire la taille de la DTA.
- Immédiatement après le lancement d'un programme, la DTA commence à la cellule de mémoire 128 du PSP et présente une longueur de 128 octets.
- Le contenu des registres AX, CX, DX, SI, DI, BP, CS, DS, SS et du registre de flags n'est pas modifié par cette fonction.

Interruption 21h, Fonction 30h DOS (à partir de 1.0)
Déterminer le numéro de version du DOS

Cette fonction permet de déterminer le numéro de version du DOS sous lequel un programme est en train de tourner.

Entrée : AH = 30h

Sortie : AL = Numéro de version principal
AH = Numéro de version complémentaire
BH = Code OEM

Remarques :
- Le numéro de version principal est le numéro qui apparaît avant le point. Pour le numéro de version 2.1, le numéro de version principal sera donc 2.
- Le numéro de version complémentaire est le numéro qui apparaît après le point. Ce numéro est toujours indiqué avec deux chiffres. Pour le numéro de version 2.1, le numéro de version complémentaire sera donc 10.
- Dans le registre BH, le numéro de code OEM n'est pas généralement explicite puisque ce code n'est pas normalisé. Cependant, la règle adoptée est que le PC-DOS (d'IBM) porte le code 0.

- Si la valeur 0 est renvoyée dans le registre AL, c'est que le programme tourne sous la version 1 du DOS, qui ne connaissait pas encore cette fonction.
- Le contenu des registres DX, SI, DI, BP, CS, DS, SS, ES et du registre de flags n'est pas modifié par cette fonction.

Interruption 21h, Fonction 31h DOS (à partir de 1.0)
Terminer programme mais laisser en mémoire

Cette fonction permet de terminer le programme exécuté et de rendre le contrôle au programme qui l'avait appelé.

Contrairement aux autres fonctions pour terminer un programme, la mémoire occupée par le programme n'est cependant pas libérée pour une utilisation ultérieure. Le programme actuel reste donc résident en mémoire.

Entrée : AH = 31h
AL = Code de fin
DX = Nombre de paragraphes à réserver

Sortie : aucune

Remarques :
- Le code de fin dans le registre AL sert à signaler au programme d'appel si le programme qu'il avait appelé a pu ou non être correctement exécuté. Le programme d'appel peut se faire communiquer cette valeur en appelant la fonction 4Dh. Dans un programme batch, cette valeur peut être testée à l'aide des instructions ERRORLEVEL et IF.
- Le nombre de paragraphes (groupes de 16 octets) à réserver indique combien d'octets, à partir du PSP, ne devront pas être libérés pour une utilisation ultérieure.
- Les blocs de mémoire réservés à l'aide de la fonction 48h ne sont pas concernés par la valeur dans le registre DX car ils ne peuvent être à nouveau libérés qu'en appelant la fonction 49h.

Interruption 21h, Fonction 32h DOS (à partir de 1.0)
Placer un pointeur sur le BPD d'un lecteur quelconque

Cette fonction permet de placer un pointeur sur le bloc de paramètres DOS (BPD) d'un lecteur quelconque.

Entrée : AH = 32h
DL = code de lecteur (0 = actuel, 1 = A, 2 = B, etc.)

Sortie : Flag Carry = 1 : Erreur
0 : Tout va bien, dans ce cas
DS:BX = Pointeur FAR sur le BPD

Remarques :

- S'il s'agit d'un changement de disquette effectué sur le périphérique spécifié, DOS s'adresse d'abord au lecteur pour compléter le BPD avec les données concernant le lecteur et le format. En revanche, s'il s'agit de disques durs, ces informations se trouvent déjà dans la mémoire si bien qu'il n'est pas nécessaire d'accéder à un périphérique.
- Lors de l'appel de cette fonction, une erreur ne peut survenir que si un code de lecteur erroné a été spécifié. Dans ce cas, le code d'erreur 15 sera retourné dans le registre AL.
- La structure du bloc de paramètres DOS varie en fonction des diverses versions DOS. Reportez-vous à cet effet au chapitre XXX.
- Le contenu des registres AH, CX, DX, SI, DI, BP, CS, SS et ES n'a pas été modifié par cette fonction.

Interruption 21h, Fonction 33h, Sous-fonction 0 DOS (à partir de 1.0)
Lire le flag Break

Le flag Break définit si le fait que la touche Control-C ait été actionnée doit être testé lors de tout appel d'une fonction DOS ou seulement lorsqu'est appelée une des fonctions d'entrée et sortie de caractères. Lorsque la touche Control-C est actionnée et que cet évènement est effectivement testé, cela déclenche une interruption 23h. Cette fonction permet donc de lire la teneur actuelle de ce flag.

Entrée : AH = 33h
AL = 0

Sortie : DL = 0 : Test seulement pour l'entrée et sortie de caractères
DL = 1 : Test lors de tout appel de fonction

Remarques :

- Le flag Break ne fait pas partie du bloc d'environnement d'un programme et il ne vaut donc pas uniquement pour un programme bien précis. Il affecte au contraire tous les programmes qui appellent les fonctions DOS d'entrée et sortie de caractères qui, elles-mêmes, testent l'apparition d'un caractère Control-C ou le fait que la touche Break ait été actionnée.

- Le contenu des registres AX, BX, CX, DH, SI, DI, BP, CS, DS, SS, ES et du registre de flags n'est pas modifié par cette fonction. Le contenu de tous les autres registres peut avoir été modifié.

Interruption 21h, Fonction 33h, Sous-fonction 1 DOS (à partir de 1.0)
Fixer le flag Break

Le flag Break définit si le fait que la touche Control-C ait été actionnée doit être testé lors de tout appel d'une fonction DOS ou seulement lorsqu'est appelée une des fonctions d'entrée et sortie de caractères. Lorsque la touche Control-C est actionnée et que cet évènement est effectivement testé, cela déclenche une interruption 23h. Cette fonction permet donc de mettre ou annuler ce flag.

Entrée : AH = 33h
AL = 1
DL = 0 : Test seulement pour l'entrée et sortie de caractères
DL = 1 : Test lors de tout appel de fonction

Sortie : aucune

Remarques :
- Le flag Break ne fait pas partie du bloc d'environnement d'un programme et il ne vaut donc pas uniquement pour un programme bien précis. Il affecte au contraire tous les programmes qui appellent les fonctions DOS d'entrée et sortie de caractères, qui elles-mêmes testent l'apparition d'un caractère Control-C ou le fait que la touche Break ait été actionnée.
- Cette fonction ne modifie le contenu d'aucun registre du processeur, y compris le registre de flags.

Interruption 21h, Fonction 34h DOS (à partir de 2.0)
Placer un pointeur sur le flag INDOS

Après appel de cette fonction, DOS communique l'adresse du flag INDOS qui est particulièrement utile pour les programmes TSR.

Entrée : AH = 34h

Sortie : ES:BX = Pointeur FAR sur le flag INDOS

Remarques :
- En ce qui concerne le flag INDOS, il s'agit d'un octet qui compte les appels récursifs de l'interruption 21h. Il contient la valeur 0 tant que DOS n'est pas actif et c'est pourquoi les fonctions DOS peuvent être appelées à partir d'un programme TSR. Si le flag contient

cependant une valeur supérieure à 0, DOS est alors justement actif puisqu'une valeur supérieure à 1 symbolise un nombre correspondant d'appels récursifs. Dans ce cas, DOS ne peut pas être interrompu par un appel de fonction émanant d'un programme TSR au risque de provoquer un plantage dans le pire des cas.

- Pour plus d'informations sur ce thème, lisez la section concernant la programmation de programmes TSR au chapitre XXX.
- Le contenu des registres CX, DX, SI, DI, BP, CS, DS, SS n'est pas modifié par cette fonction.

Interruption 21h, Fonction 35h DOS (à partir de 1.0)
Lire contenu d'un vecteur d'interruption

Cette fonction fournit en résultat le contenu actuel d'un vecteur d'interruption et donc l'adresse de la routine d'interruption correspondante.

Entrée : AH = 35h
AL = Numéro d'interruption

Sortie : ES = Adresse de segment de la routine d'interruption
BX = Adresse d'offset de la routine d'interruption

Remarques :
- Pour garantir la compatibilité de vos programmes avec les versions futures du DOS, il est préférable d'appeler cette fonction pour lire un vecteur d'interruption plutôt que de lire le contenu de ce vecteur directement dans la table de vecteurs d'interruption.
- Le contenu des registres AX, CX, DX, SI, DI, BP, CS, DS, SS et du registre de flags n'est pas modifié par cette fonction.

Interruption 21h, Fonction 36h DOS (à partir de 1.0)
Déterminer capacité disque résiduelle

Cette fonction renvoie des informations sur le périphérique spécifié (c'est-à-dire le driver de bloc spécifié) qui permettent de calculer la place mémoire encore libre.

Entrée : AH = 36h
DL = Code de périphérique

Sortie : AX = 65535 : Périphérique absent, sinon nombre de secteurs par cluster
BX = Nombre de clusters libres

CX = Nombre d'octets par secteur
DX = Nombre total de clusters du périphérique

Remarques :
- Pour le code de périphérique transmis, 0 représente le lecteur actuel, 1 le lecteur A, 2 le lecteur B, etc...
- La mémoire encore libre sur le support se calcule en multipliant le nombre d'octets par secteur par le nombre de secteurs par cluster puis par le nombre de clusters libres.
- Le contenu des registres SI, DI, BP, CS, DS, SS, ES et du registre de flags n'est pas modifié par cette fonction.

Interruption 21h, Fonction 37h, Sous-fonction 00h DOS (à partir de 2.0)
Déterminer le code pour commutateur de lignes de commande

Le caractère valable pour tous les commutateurs de lignes de commande lors de l'appel d'un programme peut être obtenu à l'aide de cette fonction.

Entrée : AH = 37h
AL = 00h

Sortie : DL = code ASCII du caractère

Remarques :
- En règle générale, on admet que le caractère servant de commutateur de lignes de commande soit le signe de la division (/). Par conséquent, rares sont les programmes qui demandent le caractère de commutation des lignes de commande et peuvent ne pas réagir à la modification de ce caractère.
- Le contenu des registres BX, CX, DH, SI, DI, BP, CS, DS, SS, ES n'est pas modifié par cette fonction.

Interruption 21h, Fonction 37h, Sous-fonction 01h DOS (à partir de 2.0)
Fixer le caractère de commutation des lignes de commande

L'appel de cette fonction permet de définir le caractère que tous les commutateurs de lignes de commande doivent reconnaître lors de l'appel d'un programme.

Entrée : AH = 37h
AL = 01h
DL = code ASCII du caractère

Sortie : Flag Carry : 0, caractère accepté
1, Erreur

Remarques :

- Actuellement, les caractères autorisés pour les commutateurs de lignes de commande sont (=), (/) et (-). Les autres caractères ne sont pas acceptés et sont rejetés.
- Vu que les programmes utilisant la sous-fonction 00h sont peu nombreux et que le caractère adopté comme commutateur de lignes de commande est toujours le caractère (/), l'appel de la sous-fonction 01h n'agit que sur de rares programmes. Mais on peut citer toutefois les divers programmes DOS internes et externes qui accordent une importance à ce caractère.
- Le contenu des registres BX, CX, DX, SI, DI, BP, CS, DS, SS, ES n'est pas modifié par cette fonction.

Interruption 21h, Fonction 38h DOS (Version 2)
Déterminer les symboles et formats caractéristiques du pays

Cette fonction permet de déterminer les différents paramètres caractéristiques du pays qui a été spécifié avec l'instruction COUNTRY dans le fichier CONFIG.SYS.

Entrée :
AH = 38h
AL = 0
DS = Adresse de segment d'un buffer
DX = Adresse d'offset d'un buffer

Sortie : aucune

Remarques :

- Avant d'appeler cette fonction, nous vous conseillons de déterminer à l'aide de la fonction 48 quelle version du DOS est utilisée, pour que vous puissiez tenir compte des différences entre les versions 2 et 3 dans la définition des registres pour l'appel et le retour de cette fonction.
- Le buffer doit avoir une taille de 32 octets au moins. La fonction y inscrira les différents paramètres caractéristiques du pays.
- Les différents octets du buffer contiennent, après appel de la fonction, les informations suivantes :

Octets 0 - 1 :	Format de date 0 = USA : Mois-Jour-Année 1 = Europe : Jour-Mois-Année 2 = Japon : Année-Mois-Jour
Octet 2 :	Code ASCII du symbole de la monnaie
Octet 3 :	0
Octet 4 :	Code ASCII du caractère de millier
Octet 5 :	0
Octet 6 :	Code ASCII du symbole décimal

Octet 7 : 0
Octets 8 - 31 : réservés

- Cette fonction ne modifie le contenu d'aucun registre du processeur, y compris le registre de flags.

Interruption 21h, Fonction 38h, Sous-fonction 0 DOS (à partir de la version 3)
Déterminer les symboles et formats caractéristiques d'un pays

Cette fonction permet de déterminer les différents paramètres caractéristiques d'un pays. Elle permet d'obtenir aussi bien les paramètres pour le pays qui a été spécifié avec l'instruction COUNTRY dans le fichier CONFIG.SYS que ceux correspondant à n'importe quel autre pays. Les différents paramètres sont placés par la fonction dans le buffer qui lui a été transmis.

Entrée : AH = 38h
AL = 0
DS = Adresse de segment d'un buffer
DX = Adresse d'offset d'un buffer
AL = 0 : Lire paramètre pour le pays actuel
AL = 1 - 254 : Code du pays dont il s'agit de lire les paramètres caractéristiques
AL = 255 : Le code du pays dont il s'agit de lire les paramètres caractéristiques figure dans le registre BX

Sortie : Flag Carry = 0 tout va bien.
Flag Carry = 1 : Code du pays incorrect

Remarques :

- Avant d'appeler cette fonction, nous vous conseillons de déterminer à l'aide de la fonction 48 quelle version du DOS est utilisée, pour que vous puissiez tenir compte des différences entre les versions 2 et 3 dans la définition des registres pour l'appel et le retour de cette fonction.
- Le buffer doit avoir une taille de 34 octets au moins. La fonction y inscrira les différents paramètres caractéristiques du pays.
- Les différents octets du buffer contiennent, après appel de la fonction, les informations suivantes :

Octets 0 - 1 : Format de date
0 = USA : Mois-Jour-Année
1 = Europe : Jour-Mois-Année
2 = Japon : Année-Mois-Jour
Octets 2 - 6 : Symbole de la monnaie (chaîne ASCII terminée par un caractère de fin)
Octet 7 : Code ASCII du caractère de millier

Octet 8 :	0
Octet 9 :	Code ASCII du symbole décimal
Octet 10 :	0
Octet 11 :	Code ASCII du séparateur dans la date
Octet 12 :	0
Octet 13 :	Code ASCII du séparateur de l'heure
Octet 14 :	0
Octet 15 :	Format de la monnaie Bit 0 = 0 : Symbole de la monnaie avant la valeur Bit 0 = 1 : Symbole de la monnaie après la valeur Bit 1 = 0 : Pas d'espace entre la valeur et le symbole de la monnaie Bit 1 = 1 : Un espace entre la valeur et le symbole de la monnaie
Octet 16 :	Précision (nombre de chiffres après le point décimal)
Octet 17 :	Format horaire Bit 0 = 0 : Horloge 12 heures Bit 0 = 1 : Horloge 24 heures
Octets 18 - 21 :	Adresse d'une routine de conversion de caractères (voir plus bas)
Octets 22 - 33 :	réservés

- Le contenu des adresses 18 à 21 représente les adresses d'offset et de segment d'une procédure FAR qui sert à convertir en majuscules les caractères spéciaux d'un pays, dans le jeu de caractères du PC. Cette routine considère le contenu du registre AL comme le code d'une minuscule à convertir en majuscule. Si une majuscule de ce type existe, elle figurera dans le registre AL après appel de la fonction. Si cette majuscule n'existe pas, par contre, le contenu du registre AL restera inchangé. Nous pourrions par exemple utiliser cette routine pour convertir un é en E, par exemple.
- Le contenu des registres AX, BX, CX, DX, SI, DI, BP, CS, DS, SS et ES n'est pas modifié par cette fonction.

Interruption 21h, Fonction 38h, Sous-fonction 1 DOS (à partir de la version 3)
Fixer le pays

Cette fonction permet de fixer le pays actuel. Ainsi sont du même coup fixés différents paramètres caractéristiques du pays. Ces paramètres peuvent être lus à l'aide de la sous-fonction 0 de cette même fonction. Par rapport aux fonctions précédentes du DOS qui prévoyaient que le pays ne pouvait être fixé qu'à l'aide de l'instruction COUNTRY

dans le fichier CONFIG.SYS, cette fonction présente l'avantage de permettre de fixer ou changer le pays, même après le lancement du système.

Entrée : AH = 38h
DX = 65535
AL = 1 - 254 : Numéro de pays
AL = 255 : Numéro supérieur à 254, auquel cas
BX = Numéro du pays

Sortie : Flag Carry =0 : Tout va bien
Flag Carry =1 : Code du pays incorrect

Remarques :

- Avant d'appeler cette fonction, il est préférable de déterminer la version actuelle du DOS à l'aide de la fonction 48, pour s'assurer que cette fonction est réellement disponible.
- Il n'est pas possible de fixer séparément les différents paramètres d'un pays. Vous pouvez seulement fixer le code du pays et le DOS fixera automatiquement les paramètres prédéfinis pour ce pays.
- Le contenu des registres AH, BX, CX, DX, SI, DI, BP, CS, DS, SS et ES n'est pas modifié par cette fonction.

Interruption 21h, Fonction 39h DOS (à partir de 1.0)
Créer sous-répertoire

Cette fonction permet de créer un nouveau sous-répertoire sur le périphérique spécifié.

Entrée : AH = 39h
DS = Adresse de segment du chemin du sous-répertoire
DX = Adresse d'offset du chemin du sous-répertoire

Sortie : Flag Carry =0 : Nouveau sous-répertoire créé
Flag Carry =1 : Erreur, dans ce cas
AX = 3 : Chemin non trouvé
AX = 5 : Accès refusé

Remarques :

- Le chemin de sous-répertoire transmis est une chaîne ASCII terminée par un caractère de fin (code ASCII 0).
- Si le chemin de sous-répertoire comporte une désignation de périphérique, l'accès se fera sur le périphérique spécifié, sinon le sous-répertoire sera créé sur le périphérique actuel.
- Une erreur peut se produire si un élément du chemin n'existe pas, si un sous-répertoire existe déjà sous le nom spécifié ou bien si le

sous-répertoire devait être créé dans le répertoire racine alors que ce dernier est déjà plein.

- Le contenu des registres BX, CX, DX, SI, DI, BP, CS, DS, SS et ES n'est pas modifié par cette fonction.

Interruption 21h, Fonction 3Ah DOS (à partir de 1.0)
Supprimer sous-répertoire

Cette fonction permet de supprimer un sous-répertoire sur le périphérique spécifié.

Entrée : AH = 3Ah
DS = Adresse de segment du chemin du sous-répertoire
DX = Adresse d'offset du chemin du sous-répertoire

Sortie : Flag Carry =0 : Sous-répertoire supprimé
Flag Carry =1 : Erreur, dans ce cas
AX = 3 : Chemin non trouvé
AX = 5 : Accès refusé

Remarques :
- Le chemin de sous-répertoire transmis est une chaîne ASCII terminée par un caractère de fin (code ASCII 0).
- Si le chemin de sous-répertoire comporte une désignation de périphérique, l'accès se fera sur le périphérique spécifié, sinon le sous-répertoire sera supprimé sur le périphérique actuel.
- Une erreur peut se produire si un élément du chemin n'existe pas, si le sous-répertoire est le sous-répertoire actuel ou bien si des fichiers figurent encore dans le sous-répertoire à supprimer.
- Le contenu des registres BX, CX, DX, SI, DI, BP, CS, DS, SS et ES n'est pas modifié par cette fonction.

Interruption 21h, Fonction 3Bh DOS (à partir de 1.0)
Fixer sous-répertoire actuel

Cette fonction permet de fixer le sous-répertoire actuel sur le périphérique spécifié.

Entrée : AH = 3Bh
DS = Adresse de segment du chemin du sous-répertoire
DX = Adresse d'offset du chemin du sous-répertoire

Sortie : Flag Carry =0 : Sous-répertoire actuel fixé
Flag Carry =1 : Erreur, dans ce cas
AX = 3 : Chemin non trouvé

Remarques :

- Le chemin de sous-répertoire transmis est une chaîne ASCII terminée par un caractère de fin (code ASCII 0).
- Si le chemin de sous-répertoire comporte une désignation de périphérique, l'accès se fera sur le périphérique spécifié, sinon le sous-répertoire actuel sera modifié sur le périphérique actuel.
- Une erreur peut se produire si un élément du chemin n'existe pas.
- Le contenu des registres BX, CX, DX, SI, DI, BP, CS, DS, SS et ES n'est pas modifié par cette fonction.

Interruption 21h, Fonction 3Ch DOS (à partir de 1.0)
Créer ou vider fichier (Handle)

Après l'appel de cette fonction, le DOS examine si le fichier spécifié existe déjà. Si c'est le cas, ce fichier est vidé et son contenu antérieur est irrémédiablement perdu. Si cependant le fichier spécifié n'existe pas encore, il est créé.

Entrée : AH = 3Ch
CX = Attribut de fichier
Bit 0 = 1 : Fichier peut seulement être lu
Bit 1 = 1 : Fichier caché
Bit 2 = 1 : Fichier système
DS = Adresse de segment du nom de fichier
DX = Adresse d'offset du nom de fichier

Sortie : Flag Carry =0 : Tout va bien, dans ce cas AX = Handle du fichier
Flag Carry =1 : Erreur, dans ce cas AX = code d'erreur
3 : Chemin non trouvé
4 : Plus de handle libre
5 : Accès refusé

Remarques :

- Les différents bits de l'attribut de fichier peuvent parfaitement être combinés.
- Le nom de fichier doit être fourni sous forme d'une chaîne ASCII terminée par un caractère de fin (code ASCII 0). Ce nom de fichier peut comporter une désignation de périphérique, une désignation de chemin complète et un nom de fichier proprement dit mais il ne doit pas contenir de jokers. Si la désignation de périphérique ou la désignation de chemin sont omises, l'accès se fera sur le périphérique actuel ou sur le répertoire actuel.

- Une erreur peut apparaître lors de l'appel de cette fonction si un élément de la désignation de chemin n'existe pas, si le fichier est censé être créé dans le répertoire racine alors que ce dernier est déjà plein ou encore si un fichier existe déjà sous ce nom et qu'il ne peut être vidé parce qu'il est protégé contre l'écriture (bit 0 de l'attribut de fichier = 1).
- Si l'appel de la fonction s'est conclu sans problème, toutes les autres fonctions handle pourront être appelées à l'aide du handle renvoyé car le fichier a aussi été ouvert.
- Le pointeur de fichier est dirigé sur le premier octet du fichier.
- Le contenu des registres BX, CX, DX, SI, DI, BP, CS, DS, SS et ES n'est pas modifié par cette fonction.

Interruption 21h, Fonction 3Dh DOS (à partir de 1.0)
Ouvrir fichier (Handle)

Cette fonction permet d'ouvrir un fichier déjà existant pour pouvoir y accéder à l'aide des autres fonctions.

Entrée : AH = 3Dh
AL = Mode d'accès
Bits 0 - 3 : Autorisation de lecture/écriture
0000(b) = Fichier peut seulement être lu
0001(b) = Fichier peut seulement être écrit
0010(b) = Fichier peut être lu ou écrit
Bits 4 - 6 : Mode de File Sharing (Partage de l'accès)
000(b) = seul le programme actuel peut accéder au fichier (mode FCB)
001(b) = seul le programme actuel peut accéder au fichier
010(b) = un autre programme peut lire le fichier mais non y écrire
011(b) = un autre programme peut écrire dans le fichier mais non le lire
100(b) = un autre programme peut lire et écrire le fichier
Bit 7 : Flag Handle
0 = le programme-enfant du programme actuel peut accéder au handle de ce fichier
1 = seul le programme actuel peut accéder au handle de ce fichier
DS = Adresse de segment du nom de fichier
DX = Adresse d'offset du nom de fichier

Sortie : Flag Carry =0 : Tout va bien, dans ce cas AX = Handle du fichier
Flag Carry =1 : Erreur, dans ce cas AX = code d'erreur

1 : Pas de logiciel de File Sharing
2 : Fichier non trouvé
3 : Chemin non trouvé ou fichier n'existe pas
4 : Plus de handle libre
5 : Accès refusé
12 : Mode d'accès non autorisé

Remarques :

- Le nom de fichier doit être fourni sous forme d'une chaîne ASCII terminée par un caractère de fin (code ASCII 0). Ce nom de fichier peut comporter une désignation de périphérique, une désignation de chemin complète et un nom de fichier proprement dit mais il ne doit pas contenir de jokers. Si la désignation de périphérique ou la désignation de chemin sont omises, l'accès se fera sur le périphérique actuel ou sur le répertoire actuel.
- Si l'appel de la fonction s'est conclu sans problème, toutes les autres fonctions handle pourront être appelées à l'aide du handle renvoyé.
- Le pointeur de fichier est dirigé sur le premier octet du fichier.
- Sous la version 2 du DOS, seuls les bits 0 à 2 du mode d'accès sont significatifs. Il convient de fixer tous les autrs bits sur 0 pour garantir une bonne exécution de l'appel de fonction même sous la version 3.
- Le mode de File Sharing des bits 4 à 6 n'a d'intérêt, même sous la version 3 du DOS, qu'à condition que le fichier figure sur une mémoire de masse faisant partie d'un réseau. Dans ce cas, ces 3 bits définissent si, et si oui dans quelles conditions, d'autres programmes, tournant sur d'autres PC du réseau, pourront accéder au fichier pendant qu'il sera ouvert à la suite de cet appel de fonction.
- L'erreur 12 ne peut apparaître que sous la version 3 du DOS, et uniquement à l'intérieur d'un réseau, si le fichier a déjà été ouvert par un autre programme et s'il a été décidé qu'aucun autre programme ne peut y accéder pour le moment.
- Le contenu des registres BX, CX, DX, SI, DI, BP, CS, DS, SS et ES n'est pas modifié par cette fonction.

Interruption 21h, Fonction 3Eh — DOS (à partir de 1.0)
Fermer fichier (Handle)

Cette fonction permet de refermer un fichier préalablement ouvert. Auparavant, toutefois, toutes les données qui doivent être écrites dans le fichier mais qui figurent encore dans les buffers internes du DOS sont écrites dans le fichier. Si le fichier a été modifié, l'entrée du répertoire correspondant à ce fichier est encore actualisée,

c'est-à-dire que la nouvelle taille du fichier y est inscrite ainsi que les date et heure actuelle, en tant que date et heure de la dernière modification.

Entrée : AH = 3Eh
BX = Handle à fermer

Sortie : Flag Carry =0 : Tout va bien
Flag Carry =1 : Erreur, dans ce cas
AX = 6 : Handle non autorisé ou fichier correspondant non ouvert

Remarques :
- Il faut veiller à ne pas appeler par mégarde cette fonction avec le numéro d'un des handles prédéfinis (0 à 4) car vous risqueriez ainsi de fermer par exemple le périphérique d'entrée ou le périphérique de sortie standard, la conséquence étant alors que vous ne pourriez plus recevoir de caractères du clavier ou que vous ne pourriez plus en afficher sur l'écran.
- Le contenu des registres BX, CX, DX, SI, DI, BP, CS, DS, SS et ES n'est pas modifié par cette fonction.

Interruption 21h, Fonction 3Fh DOS (à partir de 1.0)
Lire fichier (Handle)

Cette fonction permet de lire, à l'aide d'un handle, un nombre déterminé de caractères dans un fichier (ou sur un périphérique) préalablement ouvert. Ces caractères sont transférés dans un buffer. L'opération de lecture commence sur la position actuelle du pointeur de fichier.

Entrée : AH = 3Fh
BX = Handle du fichier ou du périphérique
CX = Nombre d'octets à lire
DS = Adresse de segment du buffer
DX = Adresse d'offset du buffer

Sortie : Flag Carry =0 : Tout va bien, dans ce cas AX = nombre d'octets lus
Flag Carry =1 : Erreur, dans ce cas AX = code d'erreur
5 : Accès refusé
6 : Handle non autorisé ou fichier non ouvert

Remarques :
- Cette fonction permet non seulement de lire des caractères sur un fichier mais aussi sur un périphérique comme par exemple le périphérique d'entrée standard (le clavier), qui porte le handle 0.
- Si le flag Carry est annulé après appel de cette fonction mais que le registre AX contient la valeur 0, cela signifie que le pointeur de

fichier se trouvait déjà sur la fin du fichier avant même que cette fonction n'ait été appelée. Dans ce cas, aucune donnée n'a pu être lue.

- Si le flag Carry est annulé après appel de cette fonction mais que le registre AX contient une valeur inférieure à celle du registre CX avant appel de la fonction, cela signifie que le nombre d'octets voulu n'a pu être lu parce que la fin du fichier a été atteinte auparavant.
- Après appel de cette fonction, le pointeur de fichier est dirigé sur le premier octet immédiatement à la suite du dernier octet lu.
- Le contenu des registres BX, CX, DX, SI, DI, BP, CS, DS, SS et ES n'est pas modifié par cette fonction.

Interruption 21h, Fonction 40h DOS (à partir de 1.0)
Ecrire dans fichier (Handle)

Cette fonction permet d'écrire, à l'aide d'un handle, un nombre déterminé de caractères dans un fichier (ou sur un périphérique) préalablement ouvert. Ces caractères sont tirés d'un buffer. L'opération d'écriture commence sur la position actuelle du pointeur de fichier.

Entrée :
AH = 40h
BX = Handle du fichier ou du périphérique
CX = Nombre d'octets à écrire
DS = Adresse de segment du buffer
DX = Adresse d'offset du buffer

Sortie :
Flag Carry =0 : Tout va bien, dans ce cas AX = nombre d'octets écrits
Flag Carry =1 : Erreur, dans ce cas AX = code d'erreur
5 : Accès refusé
6 : Handle non autorisé ou fichier non ouvert

Remarques :

- Cette fonction permet non seulement d'écrire des caractères sur un fichier mais aussi sur un périphérique comme par exemple le périphérique de sortie standard (l'écran), qui porte le handle 1.
- Si le flag Carry est annulé après appel de cette fonction mais que le registre AX contient la valeur 0, cela signifie que le périphérique sur lequel figure le fichier était déjà plein avant même que cette fonction n'ait été appelée.
- Si le flag Carry est annulé après appel de cette fonction mais que le registre AX contient une valeur inférieure à celle du registre CX avant appel de la fonction, cela signifie que le nombre d'octets voulu n'a pu être écrit parce que le périphérique sur lequel figure le fichier a été rempli auparavant.

- Après appel de cette fonction, le pointeur de fichier est dirigé sur le premier octet immédiatement à la suite du dernier octet écrit.
- Le contenu des registres BX, CX, DX, SI, DI, BP, CS, DS, SS et ES n'est pas modifié par cette fonction.

Interruption 21h, Fonction 41h DOS (à partir de 1.0)
Supprimer fichier (Handle)

Cette fonction permet de supprimer le fichier dont le nom est transmis à la fonction.

Entrée : AH = 41h
DS = Adresse de segment du nom de fichier
DX = Adresse d'offset du nom de fichier

Sortie : Flag Carry =0 : Tout va bien
Flag Carry =1 : Erreur, dans ce cas AX = code d'erreur
2 : Fichier non trouvé
5 : Accès refusé

Remarques :
- Le nom de fichier doit être fourni sous forme d'une chaîne ASCII terminée par un caractère de fin (code ASCII 0). Ce nom de fichier peut comporter une désignation de périphérique, une désignation de chemin complète et un nom de fichier proprement dit mais il ne doit pas contenir de jokers. Si la désignation de périphérique ou la désignation de chemin sont omises, l'accès se fera sur le périphérique actuel ou sur le répertoire actuel.
- Un erreur apparaît si un élément de la spécification de chemin n'existe pas ou si le fichier porte l'attribut "lecture seulement", auquel cas il n'est possible ni d'y écrire ni de le supprimer. Cet attribut peut toutefois être modifié à l'aide de la fonction 43h.
- Cette fonction ne permet de supprimer ni des sous-répertoires ni des noms de volumes.
- Le contenu des registres BX, CX, DX, SI, DI, BP, CS, DS, SS et ES n'est pas modifié par cette fonction.

Interruption 21h, Fonction 42h DOS (à partir de 1.0)
Déplacer le pointeur de fichier (handle)

Cette fonction permet de déplacer le pointeur de fichier d'un fichier déjà ouvert, à l'aide de son handle. On obtient ainsi une possibilité d'accès sélectif puisque les différents enregistrements ne sont plus nécessairement lus de façon séquentielle, c'est-à-dire dans

l'ordre dans lequel ils sont disposés. La nouvelle position du pointeur de fichier n'est pas spécifiée de façon absolue mais sous forme d'une distance (offset) par rapport à la position actuelle, par rapport au début du fichier ou encore par rapport à la fin du fichier. Cette distance est définie par un nombre de 32 bits.

Entrée : AH = 42h
AL = Code de la distance
0 : La distance se rapporte au début du fichier
1 : La distance se rapporte à la position actuelle du pointeur de fichier
2 : La distance se rapporte à la fin du fichier
BX = Handle
CX = Mot fort de la distance
DX = Mot faible de la distance

Sortie : Flag Carry =0 : Tout va bien, dans ce cas
DX = Mot fort du pointeur de fichier
AX = Mot faible du pointeur de fichier
Flag Carry =1 : dans ce cas AX = code d'erreur
1 : Code de distance non autorisé
6 : Handle non autorisé ou fichier non ouvert

Remarques :

- Si vous travaillez avec les codes d'offset 1 et 2, la distance spécifiée peut aussi être négative, soit pour ramener le pointeur de fichier en arrière, soit pour le placer avant le début du fichier. Il est ainsi possible de fixer le pointeur de fichier avant la fin du fichier. Dans ce cas, une erreur ne sera cependant signalée qu'au cours du prochain accès en lecture ou écriture à ce fichier.
- Quel que soit le code de distance transmis pour appeler la fonction, la position du pointeur de fichier renvoyée après appel de la fonction se rapporte toujours au début du fichier.
- Cette fonction permet aussi de déterminer la taille d'un fichier si on lui transmet 2 comme code de distance et 0 comme distance. Le pointeur de fichier étant ainsi amené sur le dernier octet du fichier, sa position, qui est renvoyée au programme ayant appelé la fonction, correspondra au nombre d'octets et donc à la taille du fichier.
- Le contenu des registres BX, CX, SI, DI, BP, CS, DS, SS et ES n'est pas modifié par cette fonction.

Interruption 21h, Fonction 43h, Sous-fonction 0 DOS (à partir de 1.0)
Déterminer attribut d'un fichier

Cette fonction permet de déterminer l'attribut d'un fichier quelconque.

Entrée : AH = 43h
AL = 0
DS = Adresse de segment du nom de fichier
DX = Adresse d'offset du nom de fichier

Sortie : Flag Carry =0 : Tout va bien, dans ce cas CX = Attribut du fichier
Bit 0 = 1 : Fichier doit seulement être lu mais on ne peut rien y écrire
Bit 1 = 1 : Fichier est caché (ne sera pas affiché par DIR)
Bit 2 = 1 : Fichier est un fichier système
Bit 5 = 1 : Fichier a été modifié depuis le dernier archivage

Flag Carry =1 : Erreur, dans ce cas AX = Code d'erreur
1 : Code de fonction inconnu
2 : Fichier non trouvé
3 : Chemin non trouvé

Remarques :
- Le nom de fichier doit être fourni sous forme d'une chaîne ASCII terminée par un caractère de fin (code ASCII 0). Il peut comporter une désignation de périphérique, une spécification de chemin complète ainsi qu'un nom de fichier proprement dit, mais pas de jokers. Si la désignation de périphérique ou la spécification de chemin sont omises, l'accès se fera sur le périphérique actuel ou sur le répertoire actuel.
- Une erreur apparaîtra si un élément de la spécification de chemin ou le fichier lui-même n'existe pas.
- Le contenu des registres BX, DX, SI, DI, BP, CS, DS, SS et ES n'est pas modifié par cette fonction.

Interruption 21h, Fonction 43h, Sous-fonction 1 DOS (à partir de 1.0)
Fixer attribut d'un fichier

Cette fonction permet de fixer l'attribut d'un fichier quelconque.

Entrée : AH = 43h
AL = 1
CX = Attribut du fichier

Bit 0 = 1 : Fichier doit seulement être lu mais on ne peut y écrire
Bit 1 = 1 : Fichier est caché (ne sera pas affiché par DIR)
Bit 2 = 1 : Fichier est un fichier système
Bit 3 = 0 : Volume
Bit 4 = 0 : Répertoire Bit 3 : 0(b)

Bit 5 = 1 : Fichier a été modifié depuis le dernier archivage
DS = Adresse de segment du nom de fichier
DX = Adresse d'offset du nom de fichier

Sortie : Flag Carry =0 : Tout va bien
Flag Carry =1 : Erreur, dans ce cas AX = Code d'erreur
1 : Code de fonction inconnu
2 : Fichier non trouvé
3 : Chemin non trouvé

Remarques :

- Le nom de fichier doit être fourni sous forme d'une chaîne ASCII terminée par un caractère de fin (code ASCII 0). Il peut comporter une désignation de périphérique, une spécification de chemin complète ainsi qu'un nom de fichier proprement dit, mais pas de jokers. Si la désignation de périphérique ou la spécification de chemin sont omises, l'accès se fera sur le périphérique actuel ou sur le répertoire actuel.
- Une erreur apparaîtra si un élément de la spécification de chemin ou le fichier lui-même n'existe pas.
- Cette fonction ne permet pas de manipuler les sous-répertoires ni les noms de volumes. C'est pourquoi les bits 3 et 4 de l'attribut de fichier doivent valoir 0 lors de l'appel de la fonction. Si on tente malgré tout d'accéder à un sous-répertoire ou à un nom de volume, le code d'erreur 5 sera renvoyé.
- Le contenu des registres BX, CX, DX, SI, DI, BP, CS, DS, SS et ES n'est pas modifié par cette fonction. Le contenu de tous les autres registres peut avoir été modifié.

Interruption 21h, Fonction 44h, Sous-fonction 0 DOS (à partir de 1.0)
Accès à un driver de périphérique (IOCTL) : Lecture de l'attribut de périphérique

Cette fonction permet de lire l'attribut de périphérique d'un driver de caractère, qui est stocké dans le chapeau de ce driver.

Entrée : AH = 44h
AL = 0
BX = Handle

Sortie : Flag Carry =0 : Tout va bien, dans ce cas DX = Attribut de périphérique

Bit 14 =	1 :	Le driver peut traiter des chaînes de caractères de commande particulières à travers les sous-fonctions 2 et 3 de la fonction IOCTL.
Bit 7 =	1 :	Le driver est un driver de caractères.
Bit 5 =	0 :	Le driver travaille en mode Cooked.
	1 :	Le driver travaille en mode Raw.
Bit 3 =	1 :	Le driver est un driver d'horloge.
Bit 2 =	1 :	Le driver est le driver NUL.
Bit 1 =	1 :	Le driver est le driver de sortie de la console (l'écran).
Bit 0 =	1 :	Le driver est le driver d'entrée de la console (le clavier).

Flag Carry =1 : Erreur, dans ce cas AX = Code d'erreur
1 : Code de fonction inconnu
6 : Handle non ouvert ou qui n'existe pas

Remarques :

- C'est un handle associé au périphérique voulu qui doit être spécifié et non le nom du driver de caractères appelé. Il peut s'agir par exemple de l'un des 5 handles prédéfinis (numéros 0 à 4). Il est toutefois également possible d'ouvrir au préalable un périphérique déterminé à l'aide de la fonction Ouvrir (fonction numéro 3Dh) puis de transmettre ce handle à la fonction. Comme l'entrée et la sortie standard (Les handles 0 et 1) peuvent avoir été redirigées, cette méthode permet de s'assurer que l'accès se fera bien sur le périphérique voulu.
- Si le bit 7 de l'attribut de périphérique ne vaut pas 1, c'est que le driver appelé n'est pas un driver de caractères et la signification des différents bits ne correspond donc pas à celle d'un driver de caractères.
- Le contenu des registres BX, CX, DX, SI, DI, BP, CS, DS, SS et ES n'est pas modifié par cette fonction.

Interruption 21h, Fonction 44h, Sous-fonction 1 DOS (à partir de 1.0)
Accès à un driver de périphérique (IOCTL) : Fixer l'attribut de périphérique

Cette fonction permet de fixer l'attribut de périphérique d'un driver de caractère, qui est stocké dans le chapeau de ce driver.

Entrée : AH = 44h
AL = 1
BX = Handle
DX = Attribut de périphérique

Bit 14 =	1 :	Le driver peut traiter des chaînes de carac-

tères de commande particulières à travers les sous-fonctions 2 et 3 de la fonction IOCTL.

Bit 7 =	1 :	Le driver est un driver de caractères.
Bit 5 =	0 :	Le driver travaille en mode Cooked.
	1 :	Le driver travaille en mode Raw.
Bit 3 =	1 :	Le driver est un driver d'horloge.
Bit 2 =	1 :	Le driver est le driver NUL.
Bit 1 =	1 :	Le driver est le driver de sortie de la console (l'écran).
Bit 0 =	1 :	Le driver est le driver d'entrée de la console (le clavier).

Sortie : Flag Carry =0 : Tout va bien
Flag Carry =1 : Erreur, dans ce cas AX = Code d'erreur
1 : Code de fonction inconnu
6 : Handle non ouvert ou qui n'existe pas

Remarques :

- C'est un handle associé au périphérique voulu qui doit être spécifié et non le nom du driver de caractères appelé. Il peut s'agir par exemple de l'un des 5 handles prédéfinis (numéros 0 à 4). Il est toutefois également possible d'ouvrir au préalable un périphérique déterminé à l'aide de la fonction Ouvrir (fonction numéro 3Dh) puis de transmettre ce handle à la fonction. Comme l'entrée et la sortie standard (Les handles 0 et 1) peuvent avoir été redirigées, cette méthode permet de s'assurer que l'accès se fera bien sur le périphérique voulu.
- Pour modifier différents bits de l'attribut de périphérique à l'aide de cette fonction, il est préférable de lire tout d'abord cet attribut de périphérique à l'aide de la fonction 0. Vous pouvez alors manipuler les bits voulus puis renvoyer l'attribut de périphérique au driver de périphérique, à l'aide de cette fonction.
- Cette fonction est surtout utile pour faire passer un driver de caractères du mode Raw au mode Cooked, et inversement, à l'aide du bit 5 de l'attribut de périphérique.
- Le contenu des registres BX, CX, DX, SI, DI, BP, CS, DS, SS et ES n'est pas modifié par cette fonction.

Interruption 21h, Fonction 44h, Sous-fonction 2 DOS (à partir de 1.0)
Accès à un driver de périphérique (IOCTL) : Recevoir des données d'un driver de caractères

En appelant cette fonction, un programme d'application peut recevoir des données d'un driver de caractères, de façon très directe. Le nombre d'octets à lire, qui seront copiés par le driver dans un buffer, est fixé par le programme d'appel. Le type et la structure

des données ne sont pas prédéfinis par le DOS mais peuvent être définis individuellement par chaque driver.

Entrée : AH = 44h
AL = 2
BX = Handle
CX = Nombre d'octets à lire
DS = Adresse de segment du buffer
DX = Adresse d'offset du buffer

Sortie : Flag Carry =0 : Tout va bien, dans ce cas
AX = Nombre d'octets transférés
Flag Carry =1 : Erreur, dans ce cas AX = Code d'erreur
1 : Code de fonction inconnu
6 : Handle non ouvert ou qui n'existe pas

Remarques :

- C'est un handle associé au périphérique voulu qui doit être spécifié et non le nom du driver de caractères appelé. Il peut s'agir par exemple de l'un des 5 handles prédéfinis (numéros 0 à 4). Il est toutefois également possible d'ouvrir au préalable un périphérique déterminé à l'aide de la fonction Ouvrir (fonction numéro 3Dh) puis de transmettre ce handle à la fonction. Comme l'entrée et la sortie standard (Les handles 0 et 1) peuvent avoir été redirigées, cette méthode permet de s'assurer que l'accès se fera bien sur le périphérique voulu.
- Une erreur apparaîtra si le handle spécifié est associé à un driver de bloc et non à un driver de caractères.
- Le contenu des registres BX, CX, DX, SI, DI, BP, CS, DS, SS et ES n'est pas modifié par cette fonction.

Interruption 21h, Fonction 44h, Sous-fonction 3 DOS (à partir de 1.0)
Accès à un driver de périphérique (IOCTL) : Envoyer des données à un driver de caractères

En appelant cette fonction, un programme d'application peut envoyer des données sur un driver de caractères, de façon très directe. Le nombre d'octets à écrire, qui seront tirés d'un buffer pour être transmis au driver, est fixé par le programme d'appel. Le type et la structure des données ne sont pas prédéfinis par le DOS mais peuvent être définis individuellement par chaque driver.

Entrée : AH = 44h
AL = 3
BX = Handle
CX = Nombre d'octets à transmettre

DS = Adresse de segment du buffer
DX = Adresse d'offset du buffer

Sortie : Flag Carry =0 : Tout va bien, dans ce cas
AX = Nombre d'octets transférés
Flag Carry =1 : Erreur, dans ce cas AX = Code d'erreur
1 : Code de fonction inconnu
6 : Handle non ouvert ou qui n'existe pas

Remarques :

- C'est un handle associé au périphérique voulu qui doit être spécifié et non le nom du driver de caractères appelé. Il peut s'agir par exemple de l'un des 5 handles prédéfinis (numéros 0 à 4). Il est toutefois également possible d'ouvrir au préalable un périphérique déterminé à l'aide de la fonction Ouvrir (fonction numéro 3Dh) puis de transmettre ce handle à la fonction. Comme l'entrée et la sortie standard (Les handles 0 et 1) peuvent avoir été redirigées, cette méthode permet de s'assurer que l'accès se fera bien sur le périphérique voulu.
- Une erreur apparaîtra si le handle spécifié est associé à un driver de bloc et non à un driver de caractères.
- Le contenu des registres BX, CX, DX, SI, DI, BP, CS, DS, SS et ES n'est pas modifié par cette fonction.

Interruption 21h, Fonction 44h, Sous-fonction 4 DOS (à partir de 1.0)
Accès à un driver de périphérique (IOCTL) : Recevoir des données d'un driver de bloc

En appelant cette fonction, un programme d'application peut recevoir des données d'un driver de bloc, de façon très directe. Le nombre d'octets à lire, qui seront copiés par le driver dans un buffer, est fixé par le programme d'appel. Le type et la structure des données ne sont pas prédéfinis par le DOS mais peuvent être définis individuellement par chaque driver.

Entrée : AH = 44h
AL = 4
BX = Désignation de périphérique
CX = Nombre d'octets à lire
DS = Adresse de segment du buffer
DX = Adresse d'offset du buffer

Sortie : Flag Carry =0 : Tout va bien, dans ce cas
AX = Nombre d'octets transférés
Flag Carry =1 : Erreur, dans ce cas AX = Code d'erreur
1 : Code de fonction inconnu
15 : Périphérique inconnu

Remarques :

- La désignation de périphérique n'indique pas directement le driver de périphérique mais le périphérique duquel les données devront être reçues. Le code 0 désigne ici le périphérique A, 1 le périphérique B, etc...
- Le contenu des registres BX, CX, DX, SI, DI, BP, CS, DS, SS et ES n'est pas modifié par cette fonction.

Interruption 21h, Fonction 44h, Sous-fonction 5 DOS (à partir de 1.0)
Accès à un driver de périphérique (IOCTL) : Transmettre des données à un driver de bloc

En appelant cette fonction, un programme d'application peut envoyer des données sur un driver de bloc, de façon très directe. Le nombre d'octets à lire, qui seront tirés d'un buffer pour être transmis au driver, est fixé par le programme d'appel. Le type et la structure des données ne sont pas prédéfinis par le DOS mais peuvent être définis individuellement par chaque driver.

Entrée :
AH = 44h
AL = 5
BX = Désignation de périphérique
CX = Nombre d'octets à transmettre
DS = Adresse de segment du buffer
DX = Adresse d'offset du buffer

Sortie :
Flag Carry =0 : Tout va bien, dans ce cas
AX = Nombre d'octets transférés
Flag Carry =1 : Erreur, dans ce cas AX = Code d'erreur
1 : Code de fonction inconnu
15 : Périphérique inconnu

Remarques :

- La désignation de périphérique n'indique pas directement le driver de périphérique mais le périphérique auquel les données devront être transmises. Le code 0 désigne ici le périphérique A, 1 le périphérique B, etc...
- Le contenu des registres BX, CX, DX, SI, DI, BP, CS, DS, SS et ES n'est pas modifié par cette fonction.

Interruption 21h, Fonction 44h, Sous-fonction 6 DOS (à partir de 1.0)
Accès à un driver de périphérique (IOCTL) : Tester l'état d'entrée

Cette fonction permet de déterminer si un driver de périphérique est prêt à transmettre des données à un autre programme.

Entrée : AH = 44h
AL = 6
BX = Handle

Sortie : Flag Carry =0 : Tout va bien, dans ce cas AX = Etat d'entrée
0 : Le driver n'est pas prêt
255 : Le driver est prêt
Flag Carry =1 : Erreur, dans ce cas AX = Code d'erreur
1 : Code de fonction inconnu
5 : Accès refusé
6 : Handle incorrect

Remarques :
- Le handle spécifié peut être associé à un driver de caractères ou à un fichier.
- Le contenu des registres BX, CX, DX, SI, DI, BP, CS, DS, SS et ES n'est pas modifié par cette fonction.

Interruption 21h, Fonction 44h, Sous-fonction 7 DOS (à partir de 1.0)
Accès à un driver de périphérique (IOCTL) : Tester l'état de sortie

Cette fonction permet de déterminer si un driver de périphérique est prêt à recevoir des données d'un autre programme.

Entrée : AH = 44h
AL = 7
BX = Handle

Sortie : Flag Carry =0 : Tout va bien, dans ce cas AX = Etat de sortie
0 : Le driver n'est pas prêt
255 : Le driver est prêt
Flag Carry =1 : Erreur, dans ce cas AX = Code d'erreur
1 : Code de fonction inconnu
5 : Accès refusé
6 : Handle incorrect

Remarques :
- Le handle spécifié peut être associé à un driver de caractères ou à un fichier.
- Si le handle est associé à un fichier, le driver de bloc correspondant annoncera toujours que le périphérique est prêt à recevoir des données, même si le support sur lequel figure le fichier est plein et donc même si, de ce fait, il n'est plus possible d'ajouter des données à la suite du fichier.
- Le contenu des registres BX, CX, DX, SI, DI, BP, CS, DS, SS et ES n'est pas modifié par cette fonction.

Interruption 21h, Fonction 44h, Sous-fonction 8 DOS (à partir de la version 3)
Accès à un driver de périphérique (IOCTL) : Support amovible ?

Cette fonction permet de déterminer si le support (disquette, disque dur) d'un périphérique est amovible.

Entrée : AH = 44h
AL = 8
BL = Désignation de périphérique

Sortie : Flag Carry =0 : Tout va bien, dans ce cas
AX = 0 : Support amovible
AX = 1 : Support inamovible
Flag Carry =1 : Erreur, dans ce cas AX = Code d'erreur
1 : Code de fonction inconnu
15 : Périphérique inconnu

Remarques :
- La désignation de périphérique n'indique pas directement le driver de périphérique mais le périphérique duquel les données devront être reçues.Le code 0 désigne ici le périphérique actuel, 1 le lecteur A, 2 le lecteur B, etc...
- Le contenu des registres BX, CX, DX, SI, DI, BP, CS, DS, SS et ES n'est pas modifié par cette fonction.

Interruption 21h, Fonction 44h, Sous-fonction 9 DOS (à partir de la version 3.1)
Accès à un driver de périphérique (IOCTL) : Test Device Remote

Cette fonction permet de déterminer si le lecteur spécifié (Device) est un élément du PC sur lequel le test est effectué (local) ou bien s'il figure sur un autre PC, dans le cadre d'un réseau (remote).

Entrée : AH = 44h
AL = 9
BL = Désignation de périphérique

Sortie : Flag Carry =0 : Tout va bien, dans ce cas DX = Attribut de périphérique
Bit 12 = 0 : local
Bit 12 = 1 : remote
Flag Carry =1 : Erreur, dans ce cas AX = Code d'erreur
1 : Code de fonction inconnu
15 : Périphérique inconnu

Remarques :
- Cette fonction ne peut être appelée que si un logiciel de réseau est installé.

- Le contenu des registres BX, CX, DX, SI, DI, BP, CS, DS, SS et ES n'est pas modifié par cette fonction.

Interruption 21h, Fonction 44h, Sous-fonction 0Ah DOS (à partir de la version 3.1)
Accès à un driver de périphérique (IOCTL) : Test Handle Remote

Cette fonction permet de déterminer si le fichier associé au handle spécifié (Device) est un élément du PC sur lequel le test est effectué (local) ou bien s'il figure sur un autre PC, dans le cadre d'un réseau (remote).

Entrée : AH = 44h
AL = 0Ah
BX = Handle

Sortie : Flag Carry =0 : Tout va bien, dans ce cas DX = Code IOCTL
Bit 15 = 0 : local
Bit 15 = 1 : remote
Flag Carry =1 : Erreur, dans ce cas AX = Code d'erreur
1 : Code de fonction inconnu
6 : Handle non ouvert ou qui n'existe pas

Remarques :
- Cette fonction ne peut être appelée que si un logiciel de réseau est installé.
- Le contenu des registres BX, CX, DX, SI, DI, BP, CS, DS, SS et ES n'est pas modifié par cette fonction.

Interruption 21h, Fonction 44h, Sous-fonction 0Bh DOS (à partir de la vesion 3)
Accès à un driver de périphérique (IOCTL) : Fixer répétition d'accès

Dans le cadre d'un réseau, il arrivera souvent qu'un programme souhaite accèder à un fichier alors qu'un autre programme est déjà en train d'y accéder et que ce fichier ne puisse être ouvert de ce fait. Dans une telle situation, il ne serait pas raisonnable de renoncer immédiatement à accéder au fichier car le fichier redeviendra généralement disponible après un très bref délai. C'est pourquoi le DOS gère, sur un plan interne, deux variables qui indiquent combien de fois une tentative d'accès à un fichier doit être répétée et quel délai doit s'écouler entre deux tentatives. Ce sont ces deux paramètres que cette fonction permet de fixer.

Entrée : AH = 44h
AL = 0Bh
BX = Nombre de répétitions des tentatives
CX = Délai entre deux tentatives

Sortie : Flag Carry =0 : Tout va bien
Flag Carry =1 : Erreur, dans ce cas AX = Code d'erreur
1 : Code de fonction inconnu

Remarques :
- Cette fonction ne doit être appelée que si un logiciel de réseau est installé.
- Le délai imparti au driver de périphérique dans le registre CX dépend de la vitesse du système informatique en cours. Au lieu de faire référence à une unité de temps constante, il indique notamment le nombre des répétitions d'une boucle vide dont la vitesse d'exécution dépend naturellement de celle du processeur.
- Le contenu des registres BX, CX, DX, SI, DI, BP, CS, DS, SS et ES n'est pas modifié par cette fonction.

Interruption 21h, Fonction 44h, Sous-fonction 0Ch DOS (à partir de 3.3)
Accès à un driver de périphérique (IOCTL) : Communication avec un driver de caractères

Cette fonction IOCTL représente l'interface entre des sous-fonctions assurant le travail avec les pages de code. Etant donné que les pages de code ne sont reconnues que par quelques drivers de caractères précis, ces fonctions ne sont généralement disponibles qu'avec ces drivers.

Entrée : AH = 44h
AL = 0Ch
BX = Handle lié au driver de caractères à adresser
CH = Code de périphérique
CL = Numéro de la sous-fonction à appeler
DS:DX = Pointeur FAR sur le buffer contenant des informations complémentaires

Sortie : Flag Carry = 0 : tout va bien
Flag Carry = 1 : Erreur, dans ce cas
AX = Code d'erreur

Remarques :
- Le périphérique concerné est défini à l'aide du code de périphérique contenu dans le registre CH. Les affectations suivantes sont prévues dans ce cas :
  - 0 = Le périphérique n'appartient à aucun des groupes suivants
  - 1 = COM1, COM2, COM3 ou COM4
  - 2 = Périphérique CON (écran et clavier)
  - 5 = LPT1, LPT2 ou LPT3
- Les sous-fonctions suivantes peuvent être appelées par le registre CL :

45h = Spécifier le compteur d'accès pour le driver d'imprimante (à partir de DOS 3.2)
4Ah = Sélectionner la page de code (à partir de DOS 3.3)
4Ch = Préparer l'initialisation des pages de code (à partir de DOS 3.3)
4Dh = Terminer l'initialisation des pages de code (à partir de DOS 3.3)
5Fh = Informations concernant l'écran (à partir de DOS 4.0)
65h = Déterminer le compteur d'accès pour le driver d'imprimante (à partir de DOS 3.2)
6Ah = Spécifier la page de code en cours (à partir de DOS 3.3)
6Bh = Définir la liste des pages de code installées (à partir de DOS 3.3)
7Fh = Lire les informations concernant l'écran (à partir de DOS 4.0)

- En règle générale, le schéma suivant s'applique à l'utilisation de ces fonctions :

  Lire, vérifier et sauvegarder les paramètres en cours à l'aide de la fonction 60h,
  Spécifier les nouveaux paramètres à l'aide de la fonction 40h,
  Exécuter la fonction nécessaire,
  Remettre en place les anciens paramètres avec la fonction 40h.

- Le contenu des registres BX, CX, DX, SI, DI, BP, CS, DS, SS et ES n'est pas modifié par cette fonction.

Interruption 21h, Fonction 44h, Sous-fonction 0Dh DOS (à partir de 3.2)
Accès à un driver de périphérique (IOCTL) : Communication avec un driver de bloc

Tout comme la fonction IOCTL 0Ch, cette fonction permet également d'accéder à de nombreuses sous-fonctions d'un driver de périphérique. Cependant, cette fonction IOCTL ne permet pas d'adresser un driver de caractères, mais un driver de bloc.

Entrée : AH = 44h
AL = 0Dh
BL = Numéro de lecteur
CH = 08h
CL = Numéro de la sous-fonction à appeler
DS:DX = Pointeur FAR sur le buffer contenant des informations complémentaires

Sortie : Flag Carry = 0 : Tout va bien
Flag Carry = 1 : Erreur, dans ce cas
AX = Code d'erreur

Remarques :

- En ce qui concerne le code de périphérique, 0 indique le lecteur en cours, 1 est utilisé pour A, 2 pour B, etc.
- Les sous-fonctions suivantes peuvent être appelées par le registre CL :

  40h = Spécifier les paramètres du périphérique (à partir de DOS 3.2)
  41h = Ecrire la piste sur le périphérique logique (à partir de DOS 3.2)
  42h = Formater et contrôler la piste sur le périphérique logique (à partir de DOS 3.2)
  47h = Spécifier le flag d'accès (à partir de DOS 4.0)
  60h = Déterminer les paramètres de périphérique (à partir de DOS 3.2)
  61h = Lire la piste depuis le périphérique logique (à partir de DOS 3.2)
  62h = Contrôler la piste sur le périphérique logique (à partir de DOS 3.2)
  67h = Lire le flag d'accès (à partir de DOS 4.0)

- En règle générale, le schéma suivant s'applique à l'utilisation de ces fonctions :

  Lire, vérifier et sauvegarder les paramètres en cours à l'aide de la fonction 60h,
  Spécifier les nouveaux paramètres à l'aide de la fonction 40h,
  Exécuter la fonction nécessaire,
  Remettre en place les anciens paramètres avec la fonction 40h.

- Le contenu des registres BX, CX, DX, SI, DI, BP, CS, DS, SS et ES n'est pas modifié par cette fonction.

Interruption 21h, Fonction 44h, Sous-fonction 0Eh DOS (à partir de 3.2)
Accès à un driver de périphérique (IOCTL) : Lire la dernière désignation de lecteur

Cette fonction permet de déterminer la désignation logique du lecteur dernièrement utilisée pour adresser un périphérique.

Entrée : AH = 44h
AL = 0Eh
BL = Code de périphérique (0 = actuel, 1 = A, 2 = B, etc.)

Sortie : Flag Carry = 0 : Tout va bien, dans ce cas
AL = 0 : Un seul périphérique logique appartient au périphérique spécifié
AL 1 : Numéro du périphérique logique attribué au périphérique spécifié (1 = A, 2 = B, etc.)
Flag Carry = 1 : Erreur, dans ce cas AX = Code d'erreur

Remarques :
- Cette fonction est surtout intéressante pour les systèmes équipés d'un seul lecteur de disquette. Dans ce cas, ce lecteur peut être appelé sous A et B. Cette fonction permet notamment de spécifier si le lecteur de disquette a été adressé dernièrement en tant que lecteur A ou B. Si le lecteur est adressé au moyen d'une telle désignation, un message DOS apparaît immédiatement pour demander d'insérer une disquette lors de l'accès au lecteur.
- Le contenu des registres BX, CX, DX, SI, DI, BP, CS, DS, SS et ES n'est pas modifié par cette fonction.

Interruption 21h, Fonction 44h, Sous-fonction 0Fh DOS (à partir de 3.2)
Accès à un driver de périphérique (IOCTL) : Définir la désignation de lecteur suivante

Cette fonction permet de spécifier la désignation de lecteur logique utilisée pour appeler un périphérique lors d'un prochain accès.

Entrée : AH = 44h
AL = 0Fh
BL = Code de périphérique (0 = actuel, 1 = A, 2 = B, etc.)

Sortie : Flag Carry = 0 : Tout va bien, dans ce cas
AL = Code de périphérique servant à appeler le lecteur lors d'un prochain accès
Flag Carry = 1 : Erreur, dans ce cas AX = Code d'erreur

Remarques :
- Cette fonction est surtout intéressante pour les systèmes équipés d'un seul lecteur de disquette parce que ce dernier peut être appelé sous A et B. Dans ce cas, cette fonction permet de faire commuter le lecteur de disquette de A vers B. Comme cette fonction travaille en alternance, le lecteur revient sur A à la suite d'un appel ultérieur.
- Le contenu des registres BX, CX, DX, SI, DI, BP, CS, DS, SS et ES n'est pas modifié par cette fonction.

Interruption 21h, Fonction 44h, Sous-fonction 10h DOS (à partir de 5.0)
Solliciter le soutien IOCTL au niveau Handle

A partir de la version 5.0, la mise en disponibilité de cette fonction peut être sollicitée avant l'appel d'une fonction IOCTL qui ne fait pas partie de l'environnement des fonctions de la version 3.2.

Entrée : AH = 44h
AL = 10h
BX = Handle sur le périphérique à adresser lors de l'appel de la fonction IOCTL
CL = Numéro de la fonction IOCTL souhaitée

Sortie : Flag Carry = 0 : La fonction est disponible par rapport au handle spécifié
Flag Carry = 1 : Erreur, la fonction n'est pas disponible par rapport au handle spécifié, dans ce cas AX = 1 (fonction non disponible)

Remarques :
- Alors que cette fonction ne permet de tester la disponibilité d'une fonction IOCTL précise que par rapport au périphérique qui se cache derrière un handle, la fonction IOCTL 11h permet en revanche d'effectuer un test par rapport à un périphérique précis.
- Le contenu des registres BX, CX, DX, SI, DI, BP, CS, DS, SS et ES n'est pas modifié par cette fonction.

Interruption 21h, Fonction 44h, Sous-fonction 11h DOS (à partir de 5.0)
Solliciter le soutien IOCTL au niveau du périphérique

A partir de la version 5.0, la mise en disponibilité de cette fonction peut être sollicitée avant l'appel d'une fonction IOCTL qui ne fait pas partie de l'environnement des fonctions de la version 3.2.

Entrée : AH = 44h
AL = 11h
BX = Numéro de périphérique (0 = actuel, 1 = A, B = 2, etc.)
CL = Numéro de la fonction IOCTL souhaitée

Sortie : Flag Carry = 0 : La fonction est disponible par rapport au périphérique spécifié
Flag Carry = 1 : Erreur, la fonction n'est pas disponible par rapport au périphérique, dans ce cas AX = 1 (fonction non disponible)

Remarques :

- Alors que cette fonction ne permet de tester la disponibilité d'une fonction IOCTL précise que par rapport à un périphérique précis, la fonction IOCTL 10h permet en revanche d'effectuer un test par rapport à un handle donné et le périphérique qui s'y cache derrière.
- Le contenu des registres BX, CX, DX, SI, DI, BP, CS, DS, SS et ES n'est pas modifié par cette fonction.

Interruption 21h, Fonction 45h DOS (à partir de 1.0)
Doubler Handle

Cette fonction permet de mettre en place un second handle pour un handle spécifié. Ce second handle sera associé au même fichier ou au même périphérique que le premier. Si le premier handle se rapportait à un fichier, le pointeur de fichier de ce handle sera couplé avec le pointeur de fichier du nouveau handle.

Entrée : AH = 45h
BX = Handle

Sortie : Flag Carry =0 : Tout va bien, dans ce cas AX = le nouveau Handle
Flag Carry =1 : Erreur, dans ce cas AX = Code d'erreur
4 : Pas d'autre handle disponible
6 : Handle spécifié non ouvert ou qui n'existe pas

Remarques :

- Cette fonction permet, après qu'un fichier ait été modifié, de faire actualiser l'entrée de ce fichier dans le répertoire sans être obligé de refermer le fichier. Il suffit en effet de réclamer à l'aide de cette fonction un nouveau handle pour le fichier puis de refermer le fichier avec ce handle à l'aide de la fonction 3Eh.
- Lorsque le pointeur de fichier d'un des deux handles est déplacé du fait de l'appel d'une fonction de lecture ou écriture, le pointeur de fichier de l'autre handle est automatiquement déplacé de la même façon.
- Le contenu des registres BX, CX, DX, SI, DI, BP, CS, DS, SS et ES n'est pas modifié par cette fonction.

Interruption 21h, Fonction 46h DOS (à partir de 1.0)
Assimiler handles

Cette fonction attend deux handles associés à deux fichiers ou périphériques différents. Le second handle est alors assimilé par la fonction au premier, c'est-à-dire qu'il se retrouve associé au même fichier ou au même périphérique que le premier handle. La

valeur de son pointeur de fichier coïncide également avec celle du pointeur de fichier du premier handle.

Entrée : AH = 46h
BX = Premier handle
CX = Second handle (à assimiler)

Sortie : Flag Carry =0 : Tout va bien
Flag Carry =1 : Erreur, dans ce cas AX = Code d'erreur
4 : Pas d'autre handle disponible
6 : Handle spécifié non ouvert ou qui n'existe pas

Remarques :

- Si le second handle est associé à un fichier ouvert lors de l'appel de cette fonction, ce fichier est tout d'abord fermé.
- Lorsque le pointeur de fichier d'un des deux handles est déplacé du fait de l'appel d'une fonction de lecture ou écriture, le pointeur de fichier de l'autre handle est automatiquement déplacé de la même façon.
- Le contenu des registres BX, CX, DX, SI, DI, BP, CS, DS, SS et ES n'est pas modifié par cette fonction.

Interruption 21h, Fonction 47h DOS (à partir de 1.0)
Déterminer répertoire actuel

Cette fonction inscrit dans un buffer la spécification de chemin complète du répertoire actuel sur le lecteur spécifié.

Entrée : AH = 47h
DL = Désignation de périphérique
DS = Adresse de segment du buffer
SI = Adresse d'offset du buffer

Sortie : Flag Carry =0 : Tout va bien
Flag Carry =1 : Erreur, dans ce cas
AX = 15 : Périphérique inconnu

Remarques :

- Pour la désignation de périphérique, 0 représente le lecteur actuel, 1 le lecteur A, 2 le lecteur B, etc...
- La spécification de chemin placée dans le buffer est terminée par un caractère de fin (code ASCII 0). Elle n'est précédée d'aucune désignation de périphérique ni d'un '\' pour le répertoire racine. Si le répertoire racine est le répertoire actuel, le caractère de fin se trouvera donc être le premier caractère du buffer.

- Le contenu des registres BX, CX, DX, SI, DI, BP, CS, DS, SS et ES n'est pas modifié par cette fonction.

Interruption 21h, Fonction 48h DOS (à partir de 1.0)
Réserver mémoire RAM

Cette fonction permet de réserver une zone de mémoire à l'usage d'un programme.

Entrée : AH = 48h
BX = Nombre de paragraphes réservés

Sortie : Flag Carry =0 : Tout va bien, dans ce cas
AX = Adresse de segment de la zone de mémoire
Flag Carry =1 : Erreur, dans ce cas
AX = Code d'erreur
7 : Bloc de contrôle de la mémoire détruit
8 : Mémoire résiduelle insuffisante
BX = Nombre de paragraphes encore disponibles

Remarques :
- Un paragraphe comporte 16 octets.
- Si la mémoire réclamée a pu être réservée, elle commence à l'adresse AX:0000.
- Dans un programme COM, l'appel de cette fonction échouera systématiquement puisque la totalité de la place mémoire disponible est toujours affectée à un programme COM lors de son lancement.
- Le contenu des registres CX, DX, SI, DI, BP, CS, DS, SS et ES n'est pas modifié par cette fonction.

Interruption 21h, Fonction 49h DOS (à partir de 1.0)
Libérer mémoire RAM

Cette fonction permet de libérer une zone de mémoire qui avait été auparavant réservée à l'aide de la fonction 48h, pour qu'elle soit à nouveau disponible pour d'autres programmes.

Entrée : AH = 49h
BX = Nombre de paragraphes réservés

Sortie : Flag Carry =0 : Tout va bien
Flag Carry =1 : Erreur, dans ce cas
AX = Code d'erreur
7 : Bloc de contrôle de la mémoire détruit

9 : La zone de mémoire commençant à l'adresse de segment spécifiée n'avait pas été réservée

Remarques :

- Il n'est pas nécessaire d'indiquer à la fonction la taille de la zone de mémoire à libérer car cette taille est connue du DOS.
- Si la fonction est appelée avec une adresse de segment erronée dans le registre ES, il se peut que soit libérée une zone de mémoire attribuée à un tout autre programme. Or cela peut avoir des conséquences imprévisibles allant jusqu'à un plantage du système.
- Le contenu des registres BX, CX, DX, SI, DI, BP, CS, DS, SS et ES n'est pas modifié par cette fonction.

Interruption 21h, Fonction 4Ah DOS (à partir de 1.0)
Modifier la taille d'une zone de mémoire

Cette fonction permet de modifier la taille d'une zone de mémoire qui avait été auparavant réservée avec la fonction 48h. Cette zone peut aussi bien être agrandie que diminuée.

Entrée : AH = 4Ah
BX = Nouvelle taille de la zone de mémoire en paragraphes
ES = Adresse de segment de la zone de mémoire

Sortie : Flag Carry =0 : Tout va bien
Flag Carry =1 : Erreur, dans ce cas
AX = Code d'erreur
7 : Bloc de contrôle de la mémoire détruit
8 : Mémoire résiduelle insuffisante
9 : Bloc illégal
BX = Nombre de paragraphes encore disponibles

Remarques :

- Un paragraphe comporte 16 octets.
- Si la fonction est appelée avec une adresse de segment erronée dans le registre ES, il se peut que soit libérée une zone de mémoire attribuée à un tout autre programme. Or cela peut avoir des conséquences imprévisibles allant jusqu'à un plantage du système.
- Dans un programme COM, il est conseillé d'utiliser cette fonction pour libérer toute la mémoire dont le programme n'a pas vraiment besoin car la totalité de la mémoire RAM disponible est toujours affectée à un programme COM lors de son lancement. Cela est notamment indispensable avant d'appeler la fonction EXEC (fonction numéro 4Bh).

- Le contenu des registres CX, DX, SI, DI, BP, CS, DS, SS et ES n'est pas modifié par cette fonction.

Interruption 21h, Fonction 4Bh, Sous-fonction 0 DOS (à partir de 1.0)
EXEC : Exécuter un autre programme

Cette fonction permet à un programme d'en faire exécuter un autre de façon à ce que l'exécution du premier programme reprenne dès que l'exécution du second programme sera achevée. A cet effet, il faut transmettre à la fonction le nom du programme à exécuter ainsi que l'adresse d'un bloc de paramètres qui contienne les informations dont la fonction a besoin.

Entrée :
AH = 4Bh
AL = 0
ES = Adresse de segment du bloc de paramètres
BX = Adresse d'offset du bloc de paramètres
DS = Adresse de segment du nom de programme
DX = Adresse d'offset du nom de programme

Sortie :
Flag Carry =0 : Tout va bien
Flag Carry =1 : Erreur, dans ce cas AX = Code d'erreur
1 : Code de fonction inconnu
2 : Chemin ou programme non trouvé
5 : Accès refusé
8 : Zone mémoire insuffisante
10 : Bloc d'environnement erroné
11 : Format erroné

Remarques :

- Le nom du programme doit être fourni sous forme d'une chaîne ASCII terminée par un caractère de fin (code ASCII 0). Il peut comporter une désignation de périphérique, une spécification de chemin complète ainsi qu'un nom de fichier, mais pas de jokers. Si la désignation de périphérique ou la spécification de chemin sont omises, l'accès se fera sur le périphérique actuel ou sur le répertoire actuel.
- Seuls des programmes EXE ou COM peuvent ainsi être exécutés. Pour faire exécuter un fichier batch, il faut appeler l'interpréteur de commandes (COMMAND.COM) avec le paramètre '/c' suivi du nom du fichier batch voulu.
- Le bloc de paramètres doit présenter le format suivant :

Octets 0 - 1 :	Adresse de segment du bloc d'environnement
Octets 2 - 3 :	Adresse d'offset des paramètres de commande

Octets 4 - 5 :	Adresse de segment des paramètres de commande
Octets 6 - 7 :	Adresse d'offset du premier FCB
Octets 8 - 9 :	Adresse de segment du premier FCB
Octets 10 - 11 :	Adresse d'offset du second FCB
Octets 12 - 13 :	Adresse de segment du second FCB

- Si l'adresse de segment du bloc d'environnement communiquée est la valeur 0, c'est que le programme à appeler dispose du même bloc d'environnement que le programme d'appel.
- Les paramètres de commande doivent être stockés dans la mémoire de la façon suivante : un premier octet spécifie le nombre de caractères dans la ligne d'instruction ; viennent ensuite les différents caractères ASCII dont la fin est marquée par un Carriage Return (code ASCII 13). Ce Carriage Return ne doit toutefois pas être pris en compte pour le nombre de caractères.
- Le premier FCB transmis est copié à l'adresse 5Ch, le second à l'adresse 6Ch dans le PSP du programme appelé. Si le programme appelé n'attend aucune information de ces deux FCB, n'importe quelles valeurs peuvent être inscrites dans les champs FCB du bloc de paramètres.
- Après appel de cette fonction, tous les registres sont modifiés, hormis les registres CS et IP. Il ne faut donc pas oublier de sauvegarder le contenu des registres utiles avant d'appeler cette fonction, de façon à pouvoir les restaurer une fois le programme exécuté.
- Le programme appelé peut disposer de tous les handles dont pouvait disposer le programme d'appel.

Interruption 21h, Fonction 4Bh, Sous-fonction 3 DOS (à partir de 1.0)
EXEC : Charger un autre programme comme Overlay

Cette fonction permet à un programme de charger un autre programme en mémoire sans que ce programme soit pour autant automatiquement exécuté.

Entrée :
AH = 4Bh
AL = 3
ES = Adresse de segment du bloc de paramètres
BX = Adresse d'offset du bloc de paramètres
DS = Adresse de segment du nom de programme
DX = Adresse d'offset du nom de programme

Sortie :
Flag Carry =0: Tout va bien
Flag Carry =1: Erreur, dans ce cas AX = Code d'erreur

1 : Code de fonction inconnu
2 : Chemin ou programme non trouvé
5 : Accès refusé
8 : Zone mémoire insuffisante
10 : Bloc d'environnement erroné
11 : Format erroné

Remarques :

- Le nom du programme doit être fourni sous forme d'une chaîne ASCII terminée par un caractère de fin (code ASCII 0). Il peut comporter une désignation de périphérique, une spécification de chemin complète ainsi qu'un nom de fichier, mais pas de jokers. Si la désignation de périphérique ou la spécification de chemin sont omises, l'accès se fera sur le périphérique actuel ou sur le répertoire actuel.
- Le bloc de paramètres doit présenter le format suivant :

  Octets 0 - 1 : Adresse de segment à laquelle devra être chargé l'overlay (l'adresse d'offset correspondante est 0)
  Octets 2 - 3 : Facteur de relogement

- Le facteur de relogement spécifié devra être 0 pour les programmes COM et l'adresse de segment à laquelle le programme doit être chargé pour les programmes EXE.
- Le contenu des registres BX, CX, DX, SI, DI, BP, CS, DS, SS et ES n'est pas modifié par cette fonction.

Interruption 21h, Fonction 4Bh, Sous-fonction 05h DOS (à partir de 5.0)
Adapter les EXEC personnels

A partir de la version DOS 5.0, les applications qui chargent d'autres programmes ou overlays à l'aide de la fonction DOS Exec doivent utiliser cette fonction pour éviter des problèmes lors du chargement de ces programmes ou overlays.

Entrée : AH = 4Bh
AL = 05h
DS:DX = Pointeur FAR sur la structure Exec-State

Sortie : Flag Carry = 0 : Tout va bien
Flag Carry = 1 : Erreur, dans ce cas
AX = Code d'erreur
1 : code de fonction inconnu
2 : programme non trouvé
3 : programme non trouvé
4 : trop de fichiers ouverts

5 : accès refusé
8 : zone de mémoire insuffisante
10 : bloc d'environnement erroné
11 : format erroné

Remarques :

- L'appel de cette fonction doit s'effectuer entre le chargement du programme ou overlay et son exécution. Aucune fonction DOS ou BIOS ou une interruption logicielle de quelque nature que ce soit ne doit être appelée entre l'appel de cette fonction et le lancement du programme ou overlay.
- La structure Exec-State contient des informations sur le programme ou overlay. Elle regroupe 18 octets et présente la structure suivante.

Structure Exec-State		
Adr.	Contenu	Type
00h	Réservé, doit contenir 0	1 WORD
02h	1 = Programme EXEC 2 = Overlay	1 WORD
04h	Pointeur sur une chaîne ASCII contenant le nom du programme ou overlay (l'indication du chemin est autorisée dans cette chaîne)	1 PTR
08h	Adresse de segment du PSP du programme ou overlay	1 WORD
0Ah	Point d'entrée dans le programme ou overlay	1 PTR
0Eh	Taille du programme ou overlay y compris PSP	1 DWORD
Longueur : 12h (18 octets)		

Interruption 21h, Fonction 4Ch DOS (à partir de 1.0)
Terminer programme avec un code de fin

Cette fonction permet de terminer un programme en définissant un code de fin que le programme d'appel pourra tester à l'aide de la fonction 4Dh. La mémoire RAM occupée par le programme à terminer est libérée après appel de cette fonction, de sorte qu'elle peut à nouveau être attribuée à d'autres programmes.

Entrée : AH = 4Ch
AL = Code de fin

Sortie : aucune

Remarques :
- Cette fonction est à utiliser de préférence aux autres fonctions pour terminer un programme.
- Lorsque cette fonction est appelée, les 3 vecteurs d'interruption dont le contenu avait été stocké dans le PSP avant le lancement du programme sont restaurés.
- Avant que le contrôle ne soit rendu au programme d'appel, tous les handles qui ont été ouverts par le programme appelé, ainsi que tous les fichiers correspondants, sont refermés. Cela ne concerne toutefois que les fichiers auquel on accédait par FCB.
- Le code de fin peut être examiné dans un fichier batch à l'aide des instructions ERRORLEVEL et IF.

Interruption 21h, Fonction 4Dh DOS (à partir de 1.0)
Déterminer code de fin

Un programme qui en a appelé un autre à l'aide de la fonction EXEC peut déterminer à l'aide de cette fonction quel code de fin le programme appelé a renvoyé en fin d'exécution.

Entrée : AH = 4Dh

Sortie : AH = Type de fin du programme
0 : Fin normale
1 : Fin à la suite d'un caractère Control-C ou parce que touche Break actionnée
2 : A cause d'une erreur d'accès à un périphérique
3 : Après appel de la fonction 31h ou INT 27h
AL = Code de fin

Remarques :
- Le code de fin du programme appelé ne peut être lu qu'une seule fois à l'aide de cette fonction.
- Le contenu des registres AX, BX, CX, DX, SI, DI, BP, CS, DS, SS et ES n'est pas modifié par cette fonction.

Interruption 21h, Fonction 4Eh DOS (à partir de 1.0)
Rechercher première entrée du répertoire

Cette fonction attend le nom d'un fichier qu'il s'agit de rechercher. Le fichier peut être doté d'un attribut déterminé, de sorte que des sous-répertoires ou des noms de volumes peuvent également être recherchés.

Entrée : AH = 4Eh
CX = Attribut du fichier
DS = Adresse de segment du nom de fichier
DX = Adresse d'offset du nom de fichier

Sortie : Flag Carry =0 : Tout va bien
Flag Carry =1 : Erreur, dans ce cas AX = Code d'erreur
2 : Chemin non trouvé
18 : Aucun fichier trouvé avec l'attribut spécifié

Remarques :

- Le nom du fichier doit être fourni sous forme d'une chaîne ASCII terminée par un caractère de fin (code ASCII 0). Ce nom de fichier peut comporter une désignation de périphérique, une spécification de chemin complète, le nom de fichier proprement dit et des jokers. Si la désignation de périphérique ou la spécification de chemin sont omises, l'accès se fera sur le périphérique actuel ou sur le répertoire actuel.
- L'attribut 0 permet de rechercher les fichiers normaux. Si différents bits sont fixés dans le champ d'attribut, seront recherchés non seulement les fichiers correspondants mais aussi tous les fichiers normaux.
- Si un fichier spécifié a pu être trouvé, les 43 premiers octets de la DTA fournissent des informations sur ce fichier :

Octets 0 - 20 :	Réservés
Octet 21 :	Attribut du fichier
Octets 22 - 23 :	Heure de la dernière modification du fichier
Octets 24 - 25 :	Date de la dernière modification du fichier
Octets 26 - 27 :	Mot faible de la taille du fichier
Octets 28 - 29 :	Mot fort de la taille du fichier
Octets 30 - 42 :	Nom de fichier et extension, sous forme d'une chaîne ASCII terminée par un caractère de fin (code ASCII 0).

- Cette fonction ne doit être employée que pour rechercher la première apparition d'un fichier. Lorsqu'il s'agit de rechercher un groupe de fichiers (donc lorsque le nom de fichier contient des jokers), c'est la fonction 4Fh qu'il faut appeler pour poursuivre la recherche.
- Le contenu des registres BX, CX, DX, SI, DI, BP, CS, DS, SS et ES n'est pas modifié par cette fonction.

Interruption 21h, Fonction 4Fh DOS (à partir de 1.0)
Rechercher prochaine entrée du répertoire

Après appel réussi de la fonction 4Eh ou après d'autres appels de cette fonction, cette dernière permet de rechercher d'autres fichiers correspondant au nom de fichier spécifié.

Entrée : AH = 4Fh

Sortie : Flag Carry =0 : Tout va bien
Flag Carry =1 : Erreur, dans ce cas AX = Code d'erreur
AX = 18 : Plus trouvé de fichier avec l'attribut spécifié

Remarques :

- Si un fichier spécifié a pu être trouvé, les 43 premiers octets de la DTA fournissent des informations sur ce fichier :

Octets 0 - 20 :	Réservés
Octet 21 :	Attribut du fichier
Octets 22 - 23 :	Heure de la dernière modification du fichier
Octets 24 - 25 :	Date de la dernière modification du fichier
Octets 26 - 27 :	Mot faible de la taille du fichier
Octets 28 - 29 :	Mot fort de la taille du fichier
Octets 30 - 42 :	Nom de fichier et extension, sous forme d'une chaîne ASCII terminée par un caractère de fin (code ASCII 0).

- Cette fonction ne doit être appelée qu'après que la fonction 4Eh ait été appelée en premier lieu.
- Cette fonction ne doit être appelée que si le contenu de la DTA n'a pas été modifié depuis le dernier appel de la fonction 4Eh ou 4Fh car la DTA contient des informations essentielles pour poursuivre la recherche de fichiers.
- Le contenu des registres BX, CX, DX, SI, DI, BP, CS, DS, SS et ES n'est pas modifié par cette fonction.

Interruption 21h, Fonction 50h DOS (à partir de 2.0)
Fixer PSP actif

Cette fonction spécifie le PSP en cours. Elle est par exemple utile dans les programmes TSR pour que DOS lise les données à partir de son PSP et non celui du programme tournant au premier plan lors de l'accès au fichier.

Entrée : AH = 50h
BX = Adresse de segment du PSP

Sortie : Flag Carry = 0 : Tout va bien
Flag Carry = 1 : Erreur, dans ce cas
AX = Code d'erreur

Remarques :
- Implicitement, DOS considère qu'un PSP commence fondamentalement à l'adresse d'offset 0000h à l'intérieur d'un segment. Il faut en tenir compte lors de l'appel de cette fonction.
- Le contenu des registres BX, CX, DX, SI, DI, BP, CS, DS, SS et ES n'est pas modifié par cette fonction.

Interruption 21h, Fonction 51h DOS (à partir de 2.0)
Lire le PSP actif

Cette fonction permet de connaître l'adresse de segment du PSP du programme en cours. Elle sert particulièrement aux programmes TSR pour lire l'adresse du PSP du programme de premier plan avant de définir leur propre PSP à l'aide de la fonction 50h.

Entrée : AH = 51h

Sortie : Flag Carry = 0 : Tout va bien, dans ce cas
BX = Adresse de segment du PSP en cours
Flag Carry = 1 : Erreur, dans ce cas
AX = Code d'erreur

Remarques :
- Le PSP spécifié commence toujours à l'adresse offset 0000h par rapport à l'adresse de segment indiquée.
- Vous devez donner la priorité à cette fonction par rapport à la fonction 62h qui est destinée spécialement à cet effet. En outre, elle est plus documentée que la fonction 51h. Mais elle n'est disponible qu'à partir de la version DOS 3.0 alors que la fonction 51h a été déjà introduite avec la version 2.0.
- Le contenu des registres BX, CX, DX, SI, DI, BP, CS, DS, SS et ES n'est pas modifié par cette fonction.

Interruption 21h, Fonction 52h DOS (à partir de 2.0)
Placer un pointeur sur le bloc info DOS

Cette fonction retourne un pointeur sur le bloc info DOS. Elle permet d'obtenir de nombreuses informations intéressantes qui ne sont pas toujours accessibles par un programme.

Entrée : AH = 52h

Sortie : Flag Carry = 0 : Tout va bien, dans ce cas
ES:BX = Pointeur FAR sur le DIB
Flag Carry = 1 : Erreur, dans ce cas
AX = Code d'erreur

Remarques :
- La structure du bloc info DOS est décrite au chapitre XXX.
- Le contenu des registres CX, DX, SI, DI, BP, CS, DS et SS n'est pas modifié par cette fonction.

Interruption 21h, Fonction 53h DOS (à partir de 2.0)
Convertir le BPB en DPB

Ces fonctions non documentées permettent de convertir un bloc de paramètres BIOS en un bloc de paramètres Drive.

Entrée : AH = 53h
DS:SI = Pointeur FAR sur le buffer contenant le BPB converti
ES:BP = Pointeur sur le buffer dans lequel le bloc de paramètres Drive doit être créé (lisez plus loin)

Sortie : Flag Carry = 0 : Tout va bien
Flag Carry = 1 : Erreur, dans ce cas
AX = Code d'erreur

Remarques :
- La structure des blocs de paramètres BIOS et Drive est décrite au chapitre XXX.

Interruption 21h, Fonction 54h DOS (à partir de 1.0)
Lire flag Verify

Le flag Verify définit si les données écrites sur un support tel que le disque dur ou la disquette doivent être contrôlées après une opération d'écriture pour vérifier que la

transmission a été correctement effectuée. Cette fonction permet de lire l'état de ce flag qui s'applique à tous les programmes et non pas seulement au programme actuel.

Entrée : AH = 54h

Sortie : AL = Flag Verify
0 : Verify désactivé
1 : Verify activé

Remarques :
- Le contenu du flag Verify peut être fixé à l'aide de la fonction 2Eh.
- Le contenu des registres AH, BX, CX, DX, SI, DI, BP, CS, DS, SS et ES n'est pas modifié par cette fonction.

Interruption 21h, Fonction 55h DOS (à partir de 2.0)
Créer un nouveau PSP

Cette fonction crée un nouveau PSP en copiant l'ancien PSP dans le nouveau et en adaptant ensuite les diverses informations dans le nouveau PSP. Il s'agit en particulier d'informations liées à l'état du PSP en mémoire. Par conséquent, elles peuvent être transférées de l'ancien PSP vers le nouveau avec des modifications.

Entrée : AH = 55h
DX = Adresse de segment du nouveau PSP
CX = Adresse de segment de l'ancien PSP

Sortie : Flag Carry = 0 : Tout va bien
Flag Carry = 1 : Erreur

Remarques :
- Cette fonction non documentée est conçue pour pouvoir créer un PSP pour un programme avant de le charger. Eu égard à la fonction 4Bh, elle s'avère toutefois inutile car la fonction 4Bh charge automatiquement un programme, crée un nouveau PSP et adapte les adresses de segment dans le PSP et le programme chargé.
- Le contenu des registres BX, CX, DX, SI, DI, BP, CS, DS, SS et ES n'est pas modifié par cette fonction.

Interruption 21h, Fonction 56h DOS (à partir de 1.0)
Renommer ou transférer fichier (Handle)

Cette fonction permet de renommer un fichier ou de le transférer dans un autre répertoire d'une unité de mémoire de masse. Le transfert ne peut toutefois se faire que dans les limites des différents répertoires d'un même périphérique. Il est donc impossible

de transférer de cette façon un fichier d'un répertoire du disque dur dans un répertoire d'une disquette.

Entrée : AH = 56h
DS = Adresse de segment de l'ancien nom de fichier
DX = Adresse d'offset de l'ancien nom de fichier
ES = Adresse de segment du nouveau nom de fichier
DI = Adresse d'offset du nouveau nom de fichier

Sortie : Flag Carry =0 : Tout va bien
Flag Carry =1 : Erreur, dans ce cas AX = Code d'erreur
2 : Fichier non trouvé
3 : Chemin non trouvé
5 : Accès refusé
11 : Périphérique différent

Remarques :

- Les noms de fichiers doivent être fournis sous forme de chaînes ASCII terminées par un caractère de fin (code ASCII 0). Ces noms de fichiers peuvent comporter une désignation de périphérique, une spécification de chemin complète, le nom de fichier proprement dit, mais pas de jokers. Si la désignation de périphérique ou la spécification de chemin sont omises, l'accès se fera sur le périphérique actuel ou sur le répertoire actuel.
- Une erreur peut se produire si le fichier est censé être transféré dans le répertoire racine alors que ce dernier est déjà plein.
- Cette fonction ne permet pas d'accéder aux sous-répertoires ou aux noms de volumes.
- Le contenu des registres BX, CX, DX, SI, DI, BP, CS, DS, SS et ES n'est pas modifié par cette fonction.

Interruption 21h, Fonction 57h, Sous-fonction 0 DOS (à partir de 1.0)
Déterminer date et heure de la dernière modification d'un fichier

Cette fonction permet de déterminer les date et heure de création ou du moins de la dernière modification d'un fichier.

Entrée : AH = 57h
AL = 0
BX = Handle

Sortie : Flag Carry =0 : Tout va bien, dans ce cas
CX = Heure
DX = Date
Flag Carry =1: Erreur, dans ce cas AX = Code d'erreur

1 : Fonction inconnue
6 : Handle inconnu

Remarques :

- Le fichier doit avoir été préalablement ouvert ou créé à l'aide d'une fonction Handle, de façon à ce qu'il soit possible d'y accéder à travers son handle.
- L'heure dans le registre CX présente le format suivant :

  Bits 0 - 4 : Secondes par unités de 2
  Bits 5 - 10 : Minutes
  Bits 11 - 15 : Heures

- La date dans le registre DX présente le format suivant :

  Bits 0 - 4 : Jour du mois
  Bits 5 - 8 : Mois
  Bits 9 - 15 : Année (par rapport à 1980)

- Le contenu des registres BX, SI, DI, BP, CS, DS, SS et ES n'est pas modifié par cette fonction.

Interruption 21h, Fonction 57h, Sous-fonction 1 DOS (à partir de 1.0)
Fixer date et heure de la dernière modification d'un fichier

Cette fonction permet de fixer les date et heure de création ou du moins de la dernière modification d'un fichier. Cette information est sauvegardée dans l'entrée du fichier dans le répertoire du périphérique voulu.

Entrée :
AH = 57h
AL = 1
BX = Handle
CX = Heure
DX = Date

Sortie :
Flag Carry =0 : Tout va bien
Flag Carry =1 : Erreur, dans ce cas AX = Code d'erreur
1 : Fonction inconnue
6 : Handle inconnu

Remarques :

- Le fichier doit avoir été préalablement ouvert ou créé à l'aide d'une fonction Handle, de façon à ce qu'il soit possible d'y accéder à travers son handle.
- L'heure dans le registre CX présente le format suivant :

  Bits 0 - 4 : Secondes par unités de 2
  Bits 5 - 10 : Minutes
  Bits 11 - 15 : Heures

- La date dans le registre DX présente le format suivant :

  Bits 0 - 4 : Jour du mois
  Bits 5 - 8 : Mois
  Bits 9 - 15 : Année (par rapport à 1980)

- Le contenu des registres BX, CX, DX, SI, DI, BP, CS, DS, SS et ES n'est pas modifié par cette fonction.

Interruption 21h, Fonction 58h, Sous-fonction 0 DOS (à partir de la version 3)
Lire principe de répartition de la mémoire

Lorsqu'un programme réclame au DOS une zone de mémoire, à l'aide de la fonction 48h, la mémoire se trouve généralement déjà divisée en différentes zones affectées aux divers programmes. Comme ces zones de mémoire sont rarement exactement de la taille requise, le DOS dispose de différentes possibilités pour affecter une zone de mémoire au programme. Il peut rechercher une zone de mémoire appropriée en partant du bas de la mémoire et affecter alors au programme la première zone présentant au moins la taille requise. Il peut au contraire commencer sa recherche par le haut de la mémoire et affecter également la première zone convenable. La méthode la plus efficace consiste cependant à rechercher une zone de mémoire qui soit à peine plus grande que la zone réclamée, de façon à consommer le moins de mémoire possible. Dans ce cas, aucun compte n'est tenu de la situation de cette zone de mémoire à l'intérieur de l'ensemble de la mémoire.

Cette fonction permet donc de déterminer laquelle de ces trois méthodes est utilisée pour répartir la mémoire RAM.

Entrée : AH = 58h
AL = 0

Sortie : Flag Carry =0 : Tout va bien, dans ce cas
AX = 0 : Recherche en partant du bas
AX = 1 : Recherche suivant la meilleure méthode
AX = 2 : Recherche en partant du haut
Flag Carry =1 : Erreur, dans ce cas
AX = 1 : Code de fonction inconnu

Remarques :

- Le principe de répartition de la mémoire s'applique à tous les programmes et non pas seulement au programme actuel.
- Le code de la répartition de la mémoire se définit comme suit :

  Bit 0-1 = 00b : premier bloc convenable
  = 01b : meilleur bloc convenable
  = 10b : dernier bloc convenable
  Bit 2-6 = 0b

Bit 7	= 0	: Rechercher d'abord dans la mémoire conventionnelle (à partir de la version 5)
	= 1	: Rechercher d'abord dans le UMB (à partir de la version 5)
Bit 8-15	= 0b	

- Le contenu des registres BX, CX, DX, SI, DI, BP, CS, DS, SS et ES n'est pas modifié par cette fonction.

Interruption 21h, Fonction 58h, Sous-fonction 1 DOS (à partir de la version 3)
Fixer principe de répartition de la mémoire

Lorsqu'un programme réclame au DOS une zone de mémoire, à l'aide de la fonction 48h, la mémoire se trouve généralement déjà divisée en différentes zones affectées aux divers programmes. Comme ces zones de mémoire sont rarement exactement de la taille requise, le DOS dispose de différentes possibilités pour affecter une zone de mémoire au programme. Il peut rechercher une zone de mémoire appropriée en partant du bas de la mémoire et affecter alors au programme la première zone présentant au moins la taille requise. Il peut au contraire commencer sa recherche par le haut de la mémoire et affecter également la première zone convenable. La méthode la plus efficace consiste cependant à rechercher une zone de mémoire qui soit à peine plus grande que la zone réclamée, de façon à consommer le moins de mémoire possible. Dans ce cas, aucun compte n'est tenu de la situation de cette zone de mémoire à l'intérieur de l'ensemble de la mémoire.

Cette fonction permet donc de fixer laquelle de ces trois méthodes sera utilisée pour répartir la mémoire RAM.

Entrée : AH = 58h
AL = 1
BX = Stratégie

Sortie : Flag Carry =0 : Tout va bien
Flag Carry =1 : Erreur, dans ce cas
AX = 1 : Code de fonction inconnu

Remarques :

- Le principe de répartition de la mémoire s'applique à tous les programmes et non pas seulement au programme actuel.
- Le code de la répartition de la mémoire se définit comme suit :

Bit 0-1	= 00b	: premier bloc convenable
	= 01b	: meilleur bloc convenable
	= 10b	: dernier bloc convenable
Bit 2-6	= 0b	
Bit 7	= 0	: Rechercher d'abord dans la

mémoire conventionnelle (à partir de la version 5)
= 1 : Rechercher d'abord dans le UMB (à partir de la version 5)

Bit 8-15 = 0b

- Le contenu des registres BX, CX, DX, SI, DI, BP, CS, DS, SS et ES n'est pas modifié par cette fonction.

Interruption 21h, Fonction 58h, Sous-fonction 02h DOS (à partir de 5.0)
Demander l'utilisation des UMB

A partir de la version 5.0, les blocs de mémoire supérieure (UMB) situés entre la limite des 640 Ko et 1 Mo peuvent être inclus dans la gestion de mémoire DOS. Cette fonction informe le programme d'appel si les UMB participent actuellement à la gestion de la mémoire.

Entrée : AH = 58h
AL = 02h

Sortie : Flag Carry = 0 : Tout va bien, dans ce cas
AL = 0 : UMB non utilisés
AL = 1 : UMB inclus dans la gestion de la mémoire
Flag Carry = 1 : Erreur, dans ce cas
AX = 1 : UMB non reconnu car commande DOS = UMB non spécifiée
AX = 7 : Gestion de la mémoire détruite

Remarques :
- Le code d'erreur 7 est retourné lorsque DOS remarque une inconsistance dans sa gestion de mémoire. Elle est généralement due à un accès erroné à cette zone de mémoire à l'intérieur d'un programme DOS.
- Le contenu des registres BX, CX, DX, SI, DI, BP, CS, DS, SS et ES n'est pas modifié par cette fonction.

Interruption 21h, Fonction 58h, Sous-fonction 03h DOS (à partir de 5.0)
Déterminer l'utilisation des UMB

A partir de la version 5.0, les blocs de mémoire supérieure (UMB) situés entre la limite des 640 Ko et 1 Mo peuvent être inclus dans la gestion de la mémoire DOS. Cette fonction détermine si les UMB peuvent faire partie de la gestion de la mémoire.

Entrée : AH = 58h
AL = 03h
BX = 0 : UMB non inclus
1 : UMB inclus dans la gestion de mémoire

Sortie : Flag Carry = 0 : Tout va bien
Flag Carry = 1 : Erreur, dans ce cas
AX = 1 : UMB non reconnus car commande DOS=UMB non spécifiée
AX = 7 : Gestion de mémoire détruite

Remarques :
- Le code d'erreur 7 est retourné lorsque DOS remarque une inconsistance dans sa gestion de mémoire. Elle est généralement due à un accès erroné à cette zone de mémoire à l'intérieur d'un programme DOS.
- Le contenu des registres BX, CX, DX, SI, DI, BP, CS, DS, SS et ES n'est pas modifié par cette fonction.

Interruption 21h, Fonction 59h DOS (à partir de version 3)
Lire informations d'erreur étendues

Lorsqu'une erreur est apparue à la suite de l'appel d'une des fonctions de l'interruption 21h ou de l'interruption 24h, cette fonction vous permet d'obtenir de plus amples informations sur cette erreur, à savoir d'une part une description détaillée de l'erreur, de sa cause et de son origine mais aussi, d'autre part, une indication sur la conduite à tenir pour résoudre le problème.

Entrée : AH = 59h
BX = 0

Sortie : AX = Description de l'erreur
BH = Cause de l'erreur
BL = Réaction conseillée
CH = Origine de l'erreur

Remarques :
- Les codes suivants sont employés pour décrire l'erreur (dans AX) :

0 : Aucune erreur n'est apparue
1 : Numéro de fonction inconnu
2 : Fichier non trouvé
3 : Chemin non trouvé
4 : Trop de fichiers ouverts simultanément
5 : Accès refusé
6 : Handle inconnu
7 : Bloc de contrôle de la mémoire détruit

8 : Mémoire disponible insuffisante
9 : Adresse de mémoire incorrecte
10 : Environnement incorrect
11 : Format incorrect
12 : Code d'accès incorrect
13 : Données incorrectes
14 : (réservé)
15 : Lecteur inconnu
16 : Le répertoire actuel ne peut être ouvert
17 : Périphériques différents
18 : Plus d'autre fichier
19 : Support protégé contre l'écriture
20 : Périphérique inconnu
21 : Périphérique n'est pas prêt
22 : Instruction inconnue
23 : Erreur CRC
24 : Largeur de données incorrecte
25 : Recherche infructueuse
26 : Type de périphérique inconnu
27 : Secteur non trouvé
28 : Plus de papier sur l'imprimante
29 : Erreur d'écriture
30 : Erreur de lecture
31 : Erreur générale
32 : Erreur File-Sharing
33 : Erreur File-Locking
34 : Changement de disquette non autorisé
35 : FCB indisponible
36 : Plus de place dans la liste réseau
37-49 : (réservé)
50 : Fonction réseau appelée non supportée
51 : Pas d'accès à l'ordinateur à distance
52 : Nom répété plusieurs fois à l'intérieur du réseau
Risque d'équivoque
53 : Périphérique portant le nom spécifié inconnu dans le réseau
54 : Réseau occupé
55 : Le périphérique n'existe plus dans le réseau
56 : Erreur NetBios
57 : Erreur dans une carte réseau
58 : Réponse erronée de la part du réseau
59 : Erreur réseau non attendue
60 : Carte réseau incompatible sur l'ordinateur à distance
61 : File d'attente de l'imprimante saturée
62 : Mémoire insuffisante pour imprimer le fichier
63 : Sortie du fichier sur l'imprimante interrompue
64 : Nom du réseau effacé

65 : Accès au réseau refusé
66 : Type de périphérique réseau incorrect
67 : Nom du réseau inconnu
68 : Nom du réseau trop long
69 : Erreur NetBios
70 : Partage de fichier instauré momentanément
71 : Demande de participer au réseau non acceptée
72 : Redirection vers le périphérique et le fichier momentanément désactivée
73-79 : (réservé)
80 : Fichier existe déjà
82 : Répertoire ne peut être créé
83 : Arrêt après appel de l'interruption 24h
84 : Trop de redirections
85 : Redirection doublée
86 : Mot de passe incorrect
87 : Paramètre incorrect
88 : Erreur dans le périphérique du réseau
89 : Fonction non reconnue dans le réseau
90 : Elément système nécessaire non installé

- Les codes suivants sont employés pour décrire la cause de l'erreur (dans BH) :

1 : Plus de place mémoire sur le support
2 : Interdiction temporaire d'accès (Sera vraisemblablement bientôt levée)
3 : Accès non autorisé
4 : Erreur interne du logiciel système
5 : Erreur électronique
6 : Erreur dans le logiciel système mais qui n'a pas été occasionnée par un programme d'application
7 : Erreur dans un programme d'application
8 : Fichier non trouvé
9 : Fichier de format ou type incorrect
10 : Fichier verrouillé
11 : Support incorrect sur le lecteur
12 : Objet réseau existe déjà
13 : Autre erreur

- Les codes suivants sont employés pour la réaction conseillée en vue de résoudre le problème (dans BL) :

1 : Répéter opération plusieurs fois puis laisser à l'utilisateur le soin de décider s'il veut interrompre l'opération ou ignorer l'erreur.
2 : Répéter opération plusieurs fois, après un certain délai, puis laisser à l'utilisateur le soin de décider s'il veut interrompre l'opération ou ignorer l'erreur.

3 : Demander à l'utilisateur d'entrer des données ou des informations.
4 : Terminer le programme par la voie normale
5 : Fin immédiate du programme
6 : Ignorer l'erreur
7 : Demander à l'utilisateur d'éliminer la cause de l'erreur puis de répéter l'opération

- Les codes suivants sont employés pour décrire l'origine de l'erreur (dans CH) :

1 : Inconnue
2 : Driver de bloc (disquette, disque dur, etc...)
3 : Réseau
4 : Périphérique sériel
5 : Mémoire RAM

- Seul le contenu des registres CS, SS et SP n'est pas modifié par cette fonction. Le contenu de tous les autres registres est détruit.

Interruption 21h, Fonction 5Ah DOS (à partir de version 3)
Créer fichier temporaire (Handle)

Cette fonction permet de créer un fichier qui servira uniquement au stockage intermédiaire de données pendant l'exécution d'un programme mais qui sera supprimé après exécution du programme. Le programme d'appel n'a donc pas à se préoccuper du choix d'un nom de programme. Ce nom sera en effet choisi automatiquement par le DOS. Le programme d'appel doit simplement indiquer dans quel répertoire le fichier temporaire devra être mis en place. L'accès au fichier se fera ensuite à travers le handle qui lui a été attribué, de sorte que le nom de fichier ne jouera vraiment aucun rôle. Comme un programme peut ouvrir plusieurs fichiers simultanément à l'aide de cette fonction, le DOS construit le nom du fichier à partir de la date et de l'heure actuelles. Comme la fonction ne peut jamais être appelée deux fois au même moment, cette méthode garantit que chaque fichier temporaire porte un nom différent.

Entrée : AH = 5Ah
CX = Attribut du fichier
DS = Adresse de segment du répertoire
DX = Adresse d'offset du répertoire

Sortie : Flag Carry =0 : Tout va bien, dans ce cas
AX = Handle
DS = Adresse de segment du nom de fichier complet
DX = Adresse d'offset du nom de fichier complet
Flag Carry =1 : Erreur, dans ce cas AX = Code d'erreur
3 : Chemin non trouvé

4 : Pas de handle disponible
5 : Accès refusé

Remarques :

- Le nom du répertoire doit être fourni sous forme d'une chaîne ASCII terminée par un caractère de fin (code ASCII 0). Il peut contenir une désignation de périphérique ainsi qu'une spécification de chemin complète. Si la désignation de périphérique ou la spécification de chemin sont omises, l'accès se fera sur le lecteur actuel ou sur le chemin actuel.
- Les différents bits de l'attribut de fichier ont la signification suivante :

  Bit 0 = 1 : Fichier peut seulement être lu
  Bit 1 = 1 : Fichier caché
  Bit 2 = 1 : Fichier système

- Les fichiers qui ont été créés à l'aide de cette fonction ne sont pas automatiquement supprimés à la fin du programme. Pour les supprimer, il faut tout d'abord refermer chaque fichier (fonction 3Eh) puis les supprimer à l'aide du nom de fichier complet (fonction 41h).
- Le contenu des registres BX, CX, DX, SI, DI, BP, CS, DS, SS et ES n'est pas modifié par cette fonction.

Interruption 21h, Fonction 5Bh — DOS (à partir de 3.0)
Créer un nouveau fichier (handle)

Tout comme la fonction 3Ch, cette fonction crée également un nouveau fichier et retourne un handle relatif à l'accès. Contrairement à la fonction 3Ch, un fichier existant sous le même nom n'est pas détruit et se trouve protégé contre la réécriture.

Entrée : AH = 3Ch
CX = Attribut de fichier
Bit 0 = 1 : Le fichier peut uniquement être lu
Bit 1 = 1 : Fichier caché
Bit 2 = 1 : Fichier système
DS:DX = Pointeur FAR sur le buffer contenant le nom de fichier

Sortie : Flag Carry = 0 : Tout va bien, dans ce cas AX = Handle du fichier
Flag Carry = 1 : Erreur, dans ce cas AX = Code d'erreur
3 : Chemin non trouvé
4 : Plus de handle libre
5 : Accès refusé
80 : Le fichier existe déjà

Remarques :

- Les divers bits de l'attribut de fichier peuvent être combinés.

- Le nom de fichier doit être une chaîne ASCII terminée par un caractère de fin (code ASCII 0). Outre une désignation de périphérique, il doit contenir une désignation de chemin complète et un nom de fichier mais pas de caractères génériques. Si la désignation de périphérique ou de chemin fait défaut, l'accès portera sur le périphérique ou le répertoire en cours.
- Une erreur lors de l'appel de cette fonction se produit toujours lorsqu'un élément de la désignation de chemin n'existe pas, le fichier doit être créé dans le répertoire principal qui est saturé ou lorsqu'un fichier portant le même nom existe déjà.
- Si l'appel de la fonction se termine correctement, toutes les autres fonctions handle peuvent être appelés à l'aide du handle transmis puisque le fichier a été ouvert.
- Lorsque l'appel de cette fonction a abouti, le pointeur de fichier vient se placer sur le premier octet du fichier.
- Le contenu des registres BX, CX, DX, SI, DI, BP, CS, DS, SS et ES n'est pas modifié par cette fonction.

Interruption 21h, Fonction 5Ch, Sous-fonction 00h DOS (à partir de 3.0)
Protéger une zone de fichier contre l'accès

L'accès des différentes zones d'un fichier par d'autres programmes doit être interdit en particulier dans les réseaux. Cette fonction aide à assurer la protection contre d'éventuelles modifications de ces zones.

Entrée :
AH = 5Ch
AL = 00h
BX = Handle du fichier
CX = Mot de poids fort de l'adresse du premier octet à l'intérieur du fichier à verrouiller
DX = Mot de poids faible de l'adresse du premier octet à verrouiller
SI = Mot de poids fort du nombre d'octets à verrouiller
DI = Mot de poids faible du nombre d'octets à verrouiller

Sortie :
Flag Carry = 0 : Tout va bien
Flag Carry = 1 : Erreur, dans ce cas AX = Code d'erreur
1 : Logiciel réseau non activé
6 : Handle incorrect
33 : La zone spécifiée est déjà entièrement ou partiellement verrouillée

Remarques :
- Cette fonction ne peut être appelée qu'après avoir appelé la commande SHARE ou le logiciel réseau.

- L'offset de début de la zone à verrouiller ainsi que sa longueur doivent être indiqués sous forme de LONGINT positifs composés respectivement de 4 octets.
- Une région verrouillée à l'aide de cette fonction doit être libérée avec la sous-fonction 01h. Il faut surtout empêcher de terminer le programme sans avoir libéré la zone verrouillée au risque d'obtenir des résultats imprévus lors de l'accès au fichier.
- Les programmes appelés par la fonction EXEC héritent certes des handles d'accès au fichier via le principe hiérarchique, mais ne disposent pas des zones verrouillées qui leur restent inaccessibles au même titre que tout autre programme.
- Le contenu des registres BX, CX, DX, SI, DI, BP, CS, DS, SS et ES n'est pas modifié par cette fonction.

Interruption 21h, Fonction 5Ch, Sous-fonction 01h DOS (à partir de 3.0)
Libérer une zone verrouillée dans un fichier

Cette fonction libère une zone préalablement verrouillée par a sous-fonction 00h à l'intérieur d'un fichier.

Entrée :
AH = 5Ch
AL = 01h
BX = Handle du fichier
CX = Mot de poids fort de l'adresse du premier octet à libérer à l'intérieur du fichier
DX = Mot de poids faible de l'adresse du premier octet à libérer
SI = Mot de poids fort du nombre d'octets à libérer
DI = Mot de poids faible du nombre d'octets à libérer

Sortie :
Flag Carry = 0 : Tout va bien
Flag Carry = 1 : Erreur, dans ce cas AX = Code d'erreur
1 : Logiciel réseau non activé
6 : Handle incorrect
33 : La zone spécifiée n'est pas verrouillée

Remarques :
- Cette fonction ne peut être appelée qu'après avoir appelé la commande SHARE ou le logiciel réseau.
- L'offset de début de la zone à verrouiller ainsi que sa longueur doivent être indiqués sous forme de LONGINTS positifs composés respectivement de 2 mots et 4 octets. Ils doivent correspondre exactement aux indications données lors d'un précédent appel de la sous-fonction 00h.

- Le contenu des registres BX, CX, DX, SI, DI, BP, CS, DS, SS et ES n'est pas modifié par cette fonction.

Interruption 21h, Fonction 5Dh DOS (à partir de 1.0)
Réservée à un usage interne

Interruption 21h, Fonction 5Eh, Sous-fonction 00h DOS (à partir de 3.1)
Indiquer le nom de l'ordinateur dans le réseau

Tout ordinateur connecté au réseau peut porter un nom représenté par une chaîne ASCII. L'ordinateur exécutant peut se servir de cette fonction pour demander ce nom. Le numéro NetBios de l'ordinateur est fourni simultanément.

Entrée : Ah = 5Eh
AL = 00h
DS:DX = Pointeur FAR sur le buffer dans lequel il faut copier le nom

Sortie : Flag Carry = 0 : Tout va bien, dans ce cas
CH<> = 0 : Pas de nom
0 : Nom dans le buffer spécifié, dans ce cas
CL = Numéro NetBios de l'ordinateur
Flag Carry = 1 : Erreur, dans ce cas AX = Code d'erreur
1 : Pas de logiciel réseau chargé

Remarques :
- Cette fonction ne peut être appelée qu'une fois le logiciel réseau lancé.
- Après l'appel de cette fonction, le buffer spécifié contient une chaîne ASCII représentant le nom de l'ordinateur sur lequel s'est effectué l'appel de la fonction. La longueur de la chaîne est toujours de 15 caractères. Elle se termine par le code ASCII nul et comprend ainsi 16 octets.
- Le contenu des registres BX, DX, SI, DI, BP, CS, DS, SS et ES n'est pas modifié par cette fonction.

Interruption 21h, Fonction 5Eh, Sous-fonction 02h DOS (à partir de 3.1)
Déterminer la chaîne d'initialisation de l'imprimante réseau

A l'intérieur d'un réseau, chaque ordinateur peut se servir de cette fonction pour déterminer une chaîne d'initialisation tout à fait spécifique chargée de transmettre la sortie depuis l'ordinateur concerné vers l'imprimante réseau. Il est par exemple possible de définir un mode d'exploitation particulier et d'utiliser l'imprimante réseau reliée à divers ordinateurs dans des modes d'exploitation variés.

Entrée : AH = 5Eh
AL = 02h
BX = Identification numérique de l'imprimante adressée par rapport à la liste du réseau
CX = Longueur de la chaîne d'initialisation en caractères
DS:SI = Pointeur FAR sur le buffer contenant la chaîne d'initialisation

Sortie : Flag Carry = 0 : Tout va bien
Flag Carry = 1 : Erreur, dans ce cas AX = Code d'erreur
1 : Logiciel réseau non chargé

Remarques :
- Cette fonction ne peut être appelée qu'une fois le logiciel réseau lancé.
- La liste du réseau est une liste permettant d'affecter des noms de périphérique locaux aux périphériques, répertoires et fichiers d'un réseau. A ce sujet, reportez-vous également aux diverses sous-fonctions de la fonction 5Fh permettant de retourner une valeur d'index correspondante.
- Le contenu des registres BX, CX, DX, SI, DI, BP, CS, DS, SS et ES n'est pas modifié par cette fonction.

Interruption 21h, Fonction 5Eh, Sous-fonction 03h DOS (à partir de 3.1)
Lire la chaîne d'initialisation d'une imprimante réseau

La chaîne d'initialisation d'une imprimante réseau spécifiée par la sous-fonction 02h peut être obtenue à l'aide de cette fonction.

Entrée : AH = 5Eh
AL = 03h
BX = Identification numérique de l'entrée souhaitée dans la liste du réseau
DS:SI = Pointeur FAR sur le buffer devant recevoir la chaîne d'initialisation

Sortie : Flag Carry = 0 : Tout va bien, dans ce cas
CX = Longueur de la chaîne d'initialisation transmise
Flag Carry = 1 : Erreur, dans ce cas AX = Code d'erreur
1 : Logiciel réseau non chargé

Remarques :
- Cette fonction ne peut être appelée qu'une fois le logiciel réseau lancé.
- Le buffer spécifié doit contenir suffisamment de place pour recevoir 64 caractères.
- L'identification numérique de l'imprimante peut être obtenue à l'aide de la sous-fonction 02h de la fonction 5Fh. Elle est définie par la sous-fonction 03h de la fonction 5Fh lors de la création de l'entrée dans la liste du réseau.
- Le contenu des registres BX, DX, SI, DI, BP, CS, DS, SS et ES n'est pas modifié par cette fonction.

Interruption 21h, Fonction 5Fh, Sous-fonction 02h DOS (à partir de 3.1)
Lire une entrée de la liste du réseau

Retourne une entrée de la liste du réseau définie préalablement par la sous-fonction 03h de cette fonction.

Entrée : AH = 5Fh
AL = 02h
BX = Numéro de l'entrée souhaitée comme index dans la liste du réseau
DS:SI = Pointeur FAR sur le buffer devant recevoir le nom local
ES:DI = Pointeur FAR sur le buffer devant recevoir le nom réseau du périphérique

Sortie : Flag Carry = 0 : Tout va bien
BH = Bit 0 : 0 = Périphérique existant
: 1 = Périphérique inexistant
BL = Type de périphérique
: 03 = Imprimante
: 04 = Lecteur (périphérique, répertoire ou fichier)
CX = Identification numérique pour l'accès au périphérique via la liste du réseau
Flag Carry = 1 : Erreur, dans ce cas AX = Code d'erreur
1 : Logiciel réseau non chargé
18 : Index spécifié trop grand

Remarques :
- Cette fonction ne peut être appelée qu'une fois le logiciel réseau lancé.

- Le buffer réservé pour le nom local doit prévoir une place de 16 caractères et 128 pour celui du nom réseau. Dans les deux cas, l'information souhaitée est retournée sous la forme d'une chaîne ASCII terminée par le code ASCII nul.
- Vous pouvez utiliser cette fonction pour obtenir toutes les entrées de la liste du réseau. Commencez l'appel à partir de l'indice 0 puis continuez avec des indices croissants jusqu'à ce que la fonction retourne le code d'erreur 18.
- Le contenu des registres SI, DI, CS, DS, SS et ES n'est pas modifié par cette fonction. En revanche, le contenu de tous les autres registres est modifié même si le registre ne sert pas à retourner des informations.

Interruption 21h, Fonction 5Fh, Sous-fonction 03h DOS (à partir de 3.1)
Définir une entrée dans la liste du réseau

Ces fonctions permettent de définir une nouvelle entrée dans la liste du réseau utilisée pour représenter un nom local à partir du nom d'une imprimante ou d'un lecteur.

Entrée : AH = 5Fh
AL = 03h
BL = Type de périphérique
: 03 = Imprimante
: 04 = Lecteur (périphérique, répertoire ou fichier)
CX = Identification numérique pour accéder au périphérique via la liste du réseau
DS:SI = Pointeur FAR sur le buffer contenant le nom local du périphérique
ES:DI = Pointeur FAR sur le buffer contenant le nom réseau du périphérique

Sortie : Flag Carry = 0 : Tout va bien
Flag Carry = 1 : Erreur, dans ce cas AX = Code d'erreur
1 : Logiciel réseau non chargé
3 : Chemin d'accès inexistant
5 : Accès refusé
8 : Mémoire insuffisante pour incorporer une autre entrée dans la liste du réseau

Remarques :
- Cette fonction ne peut être appelée qu'une fois le logiciel réseau lancé.
- La longueur du nom du périphérique doit être de 15 caractères et doit se terminer par le code ASCII nul. On obtient ainsi une longueur maximale de 16 octets.

- La longueur du nom réseau du périphérique peut atteindre 127 caractères et doit se terminer également par le code ASCII nul. La longueur maximal du buffer concerné s'élève ici à 128 octets.
- Le contenu des registres BX, CX, SI, DI, BP, CS, DS, SS et ES n'est pas modifié par cette fonction.

Interruption 21h, Fonction 5Fh, Sous-fonction 04h DOS (à partir de 3.1)
Supprimer une entrée de la liste du réseau

Cette fonction permet de supprimer de la liste du réseau une entrée définie préalablement par la sous-fonction 03h. La redirection vers le périphérique concerné s'effectue alors automatiquement.

Entrée : AH = 5Fh
AL = 04h
DS:SI = Pointeur FAR sur le buffer contenant le nom local du périphérique

Sortie : Flag Carry = 0 : Tout va bien
Flag Carry = 1 : Erreur, dans ce cas AX = Code d'erreur
1 : Logiciel réseau non chargé
3 : Entrée inconnue

Remarques :
- Cette fonction ne peut être appelée qu'une fois le logiciel réseau lancé.
- La longueur du nom du périphérique doit être de 15 caractères et doit se terminer par le code ASCII nul. On obtient ainsi une longueur maximale de 16 octets.
- Le contenu des registres BX, CX, SI, DI, BP, CS, DS, SS et ES n'est pas modifié par cette fonction.

Interruption 21h, Fonction 60h DOS (à partir de 3.0)
Augmenter le nom de fichier

Cette fonction sert à augmenter un nom de fichier si bien qu'il peut contenir une désignation complète de chemin et de périphérique outre le nom de fichier. On parle alors d'un nom de fichier "canonique".

Entrée : AH = 60h
DS:SI = Pointeur FAR sur le buffer contenant le nom de fichier à compléter sous forme d'une chaîne ASCII

ES:DI = Pointeur FAR sur le buffer devant recevoir le nom complet sous forme d'une chaîne ASCII

Sortie : Flag Carry = 0 : Tout va bien
Flag Carry = 1 : Erreur, dans ce cas AX = Code d'erreur

Remarques :
- Outre l'augmentation du nom, cette fonction permet également de déterminer si un périphérique donné existe physiquement ou s'il est représenté par un autre périphérique. A cet effet, transmettez une chaîne à la fonction qui ne se compose uniquement que de la désignation du périphérique à consulter suivie d'un double point et d'un backslash pour le répertoire principal. Si cette chaîne est identique à la chaîne retournée, cela signifie que le périphérique existe physiquement. Le cas contraire indique qu'il s'agit d'une représentation et la valeur de retour désigne le périphérique et le répertoire sur lequel se trouve le lecteur.
- Le contenu des registres BX, CX, SI, DI, BP, CS, DS, SS et ES n'est pas modifié par cette fonction.

Interruption 21h, Fonction 61h DOS (à partir de 1.0)

Réservée à un usage interne

Interruption 21h, Fonction 62h DOS (à partir de version 3)
Déterminer adresse de PSP

Cette fonction permet de lire l'adresse du PSP du programme actuellement exécuté.

Entrée : AH = 62h

Sortie : BX = Adresse de segment du PSP

Remarques :
- Le PSP commence à l'adresse BX:0000.
- Le contenu des registres AX, CX, DX, SI, DI, BP, CS, DS, SS, ES et du registre de flags n'est pas modifié par cette fonction.

Interruption 21h, Fonction 63h DOS (à partir de 1.0)

Réservée à un usage interne

Interruption 21h, Fonction 64h DOS (à partir de 1.0)

Réservée à un usage interne

Interruption 21h, Fonction 65h DOS (à partir de 1.0)

Réservée à un usage interne

Interruption 21h, Fonction 66h, Sous-fonction 01h DOS (à partir de 3.3)
Lire la page de code en cours

Cette fonction retourne le numéro de la page de code en cours et le numéro de la page standard spécifiées lors du lancement du système.

Entrée : AH = 66h
AL = 01h

Sortie : Flag Carry = 0 : Tout va bien, dans ce cas
BX = Numéro de la page de code en cours
DX = Numéro de la page de code standard
Flag Carry = 1 : Erreur, dans ce cas
AX = Code d'erreur
2 = La page de code n'existe pas

Remarques :
- Le contenu des registres CX, SI, DI, BP, CS, DS, SS, ES et du registre de flags n'est pas modifié par cette fonction.

Interruption 21h, Fonction 66h, Sous-fonction 02h DOS (à partir de 3.3)
Déterminer la page de code en cours

Cette fonction détermine la page de code en cours dans la mesure où les commandes NLSFUNC et MODE CP PREPARE ont été préalablement appelées sur l'interface DOS et le driver COUNTRY.SYS a été implanté à travers le fichier de configuration.

Entrée : AH = 66h
AL = 02h
BX = Numéro de la page de code à sélectionner

Sortie : Flag Carry = 0 : Tout va bien
Flag Carry = 1 : Erreur, dans ce cas AX = Code d'erreur
2 : La page de code ne peut pas être chargée
65 : Le périphérique ne peut pas être commuté vers une nouvelle page de code

Remarques :
- Le contenu des registres BX, CX, DX, SI, DI, BP, CS, DS, SS, ES et du registre de flags n'est pas modifié par cette fonction.

Interruption 21h, Fonction 67h DOS (à partir de 3.3)
Spécifier le nombre de handles disponibles

Le nombre de handles disponibles ainsi que le nombre de fichiers qu'un programme peut adresser simultanément peut être défini en appelant cette fonction. Alors que jusqu'à la version DOS 3.3, un programme ne pouvait disposer que de 20 handles au maximum, cette fonction permet à un programme de travailler simultanément sur plusieurs milliers de fichiers.

Entrée : AH = 67h
BX = Nombre de handles disponibles

Sortie : Flag Carry = 0 : Tout va bien
Flag Carry = 1 : Erreur, dans ce cas AX = Code d'erreur

Remarques :
- Tant que DOS travaille avec l'option par défaut définie à 20, les handles sont remis dans le PSP du programme. Avec une valeur plus grande, les handles ne peuvent plus tenir dans le PSP, ce qui oblige DOS à allouer une cellule mémoire chargée de recevoir les handles à l'avenir. L'appel de la fonction échoue si la mémoire RAM est insuffisante.
- Un programme ne peut pas accéder à plus de handles que le nombre prévu dans la commande FILES du fichier de configuration CONFIG.SYS. Bien que DOS ne signale pas une erreur en présence

d'une valeur plus grande, le nombre des handles disponibles reste toutefois limité à la valeur spécifiée dans FILES.

- Le contenu des registres BX, CX, DX, SI, DI, BP, CS, DS, SS, ES et du registre de flags n'est pas modifié par cette fonction.

Interruption 21h, Fonction 68h DOS (à partir de 3.3)
Libérer le buffer de fichier

En raison de l'appel de cette fonction, DOS est obligé d'inscrire toutes les données non encore écrites dans un fichier mais se trouvant encore dans les buffers de fichier internes dans le fichier spécifié. Cela empêche par exemple de perdre les données à cause d'une panne d'électricité.

Entrée : AH = 68h
BX = Handle du fichier adressé

Sortie : Flag Carry = 0 : Tout va bien
Flag Carry = 1 : Erreur, dans ce cas AX = Code d'erreur

Remarques :

- Avant que cette fonction ne soit introduite dans la version DOS 3.3, il existait deux méthodes pour obliger DOS à libérer les buffers de fichier internes dans le fichier concerné. A certains égards, elles ont perdu leur intérêt face à la fonction décrite ici et ne sont d'ailleurs plus utilisées surtout si on exécute un programme sous la version DOS 3.3 ou supérieure.
- Le contenu des registres BX, CX, DX, SI, DI, BP, CS, DS, SS, ES et du registre de flags n'est pas modifié par cette fonction.

Interruption 21h, Fonction 69h DOS (à partir de 1.0)

Réservée à un usage interne

Interruption 21h, Fonction 6Ah DOS (à partir de 1.0)

Réservée à un usage interne

Interruption 21h, Fonction 6Bh DOS (à partir de 1.0)

Réservée à un usage interne

Interruption 21h, Fonction 6Ch DOS (à partir de 4.0)
Fonction OPEN étendue

A partir de la version 4.0, cette fonction permet de créer un fichier lors de l'appel OPEN ou d'écraser un fichier existant. La fonction augmente ainsi les possibilités normales de la fonction OPEN 3Dh.

Entrée : AH = 6Ch
AL = 0
BX = Mode d'accès
CX = Attribut de fichier
DX = Réaction DOS
DS:SI = Pointeur FAR sur le buffer contenant le nom de fichier.

Sortie : Flag Carry = 0 : Tout va bien, dans ce cas AX = Handle du fichier
CX = Statut
Flag Carry = 1 : Erreur, dans ce cas AX = Code d'erreur
1 : Logiciel de partage de fichier absent
2 : Fichier non trouvé
3 : Chemin non trouvé ou fichier inexistant
4 : Plus de handle disponible
5 : Accès refusé
12 : Mode d'accès interdit

Remarques :

- Le nom de fichier doit être une chaîne ASCII terminée par un caractère de fin (code ASCII 0). Outre la désignation du périphérique, il doit contenir la désignation complète du chemin et un nom de fichier, mais pas de jokers. Si la désignation de périphérique ou de chemin fait défaut, l'accès portera sur le périphérique ou le répertoire en cours.

- Le mode d'accès se construit comme suit :

  Bit 0-2 : Autorisation de lecture/écriture
  000b = Le fichier est destiné uniquement à la lecture
  001b = Le fichier est destiné uniquement à l'écriture
  010b = Le fichier est destiné à la lecture et l'écriture
  Bit 3 : 0b
  Bit 4-6 : Mode de partage de fichier
  000b = Seul le programme en cours peut accéder au

fichier (mode de compatibilité)
001b = Seul le programme en cours peut accéder au fichier
010b = Un autre programme peut lire le fichier mais ne peut pas y écrire
011b = Un autre programme peut écrire dans le fichier mais ne peut pas le lire
100b = Un autre programme peut écrire dans le fichier et le lire

Bit 7 : Flag Handle
0 = Le programme enfant du programme en cours peut également accéder au handle de ce fichier
1 = Seul le programme en cours peut accéder au handle de ce fichier

Bit 8-12 : 0b
Bit 13:0 = En cas d'erreur grave, appeler l'interruption 24h
1 = En cas d'erreur grave, retourner le code d'erreur correspondant dans AX mais ne pas appeler l'interruption 24h
Bit 14:1 = Lors de tout accès en écriture sur le fichier, adapter immédiatement l'entrée du répertoire

Bit 15 : 0b

- Avant l'appel de la fonction, l'attribut de fichier du registre CX peut être chargé avec les attributs suivants.

  Bit 0 = 1 : Le fichier peut uniquement être lu (Read-Only)
  Bit 1 = 1 : Le fichier est protégé (non affiché par DIR)
  Bit 2 = 1 : C'est un fichier système
  Bit 5 = 1 : Fichier modifié depuis la dernière sauvegarde
  Bit 6-15 : 0b

- Lorsque le fichier à ouvrir n'existe pas encore ou existe déjà, la réaction de la fonction apparaît dans les deux Nibles du registre DX. Les bits 0 à 3 sont réservés à la réaction lorsque le fichier n'existe pas encore. Dans ce cas, les valeurs prennent la signification suivante :

  0000b = Terminer la fonction avec un code d'erreur
  0001b = Créer le fichier

- Les bits 4 à 7 signalent la réaction lorsque l'accès porte sur un fichier existant. Les valeurs signifient alors :

  0000b = Terminer la fonction avec un code d'erreur
  0001b = Ouvrir le fichier existant

- Sous DOS 4.0 également, la fonction 3Dh peut être naturellement utilisée pour ouvrir un fichier.

- Si l'appel de la fonction se déroule correctement, on peut utiliser le handle transmis pour appeler toutes les autres fonctions handle. Le registre CX contient alors un statut de la fonction informant sur les actions de la fonction. Dans ce cas,

  0 = Fichier ouvert
  1 = Nouveau fichier créé ou ouvert
  2 = Fichier existant écrasé ou ouvert

- Le pointeur de fichier est placé sur le premier octet du fichier.
- Dans la version 4.0 également, le mode de partage de fichier dans les bits 4 à 6 du mode d'accès n'est intéressant que si le fichier se trouve sur une mémoire de masse faisant partie d'un réseau. Dans un tel cas, ces trois bits déterminent si d'autres programmes du réseau peuvent accéder à ce fichier lors de son ouverture et précisent les droits d'accès de ces programmes. Si ce bit possède la valeur 0, DOS traite ce fichier dans le Compatibility-Mode où l'existence d'un réseau est ignorée par rapport à ce fichier. Ainsi, seul le programme en cours peut accéder effectivement à ce fichier.
- L'erreur 12 ne survient que sous la version DOS 3 et uniquement à l'intérieur d'un réseau lorsque le fichier est déjà ouvert par un autre programme et qu'il a été confirmé qu'aucun autre programme ne peut momentanément y accéder.
- Le contenu des registres BX, DX, SI, DI, BP, CS, DS, SS et ES n'est pas modifié par cette fonction.

Interruption 22h DOS (à partir de 1.0)
Routine pour terminer un programme

Le vecteur d'interruption de cette routine, qui figure à l'adresse 0000:0088, contient l'adresse d'une routine servant à terminer un programme. Cette routine sera appelée par toutes les fonctions pour terminer un programme, c'est-à-dire par l'interruption 20h et les fonctions 0h, 21h et 4Ch de l'interruption 21h. Elle rend ensuite le contrôle au programme-père, c'est-à-dire au programme qui avait appelé le programme qu'il s'agit de terminer. Cependant, bien que le vecteur d'interruption contienne l'adresse d'une routine qui pourrait donc être appelée à l'aide de l'instruction (langage machine) INT, cela ne doit jamais être tenté car cette routine est conçue comme une procédure FAR et non comme une routine d'interruption.

Si un programme modifie le contenu de ce vecteur d'interruption au cours de son exécution, la routine du DOS ne pourra plus être appelée pour terminer un programme. Pour éviter cela, le DOS sauvegarde le contenu de ce vecteur d'interruption dans le PSP du programme à exécuter avant de lui passer la main. Lorsqu'un programme est terminé à l'aide des fonctions indiquées plus haut, le contenu du vecteur d'interruption est tout d'abord recopié du PSP dans le vecteur, après quoi la routine voulue peut être appelée.

Interruption 23h DOS
Touche Break actionnée

Cette interruption, ou plutôt le vecteur d'interruption correspondant, est également utilisée pour stocker l'adresse d'une routine. Il s'agit ici de la routine qui est appelée chaque fois qu'un caractère Control-C est identifié ou bien que la touche Break est actionnée. Il serait donc possible d'appeler cette routine à l'aide de l'instruction (langage machine) INT mais cela ne doit jamais être tenté car cette routine est conçue comme une procédure FAR et non comme une routine d'interruption.

Un programme peut, au cours de son exécution, modifier le contenu de ce vecteur d'interruption, qui est stocké à partir de la cellule de mémoire 0000:008C, pour que ce ne soit plus la routine du DOS mais une routine du programme qui soit appelée lorsqu'un caractère Control-C est rencontré. C'est un moyen, par exemple, d'empêcher que son exécution puisse être interrompue par la routine du DOS lorsqu'un caractère Control-C est reçu. Lorsque cette routine sera appelée, tous les registres présenteront exactement la valeur qu'ils avaient lorsqu'avait été appelée la fonction DOS interrompue. Le programme peut alors entreprendre une réaction appropriée à la situation (par exemple en sortant un message sur l'écran) puis rendre le contrôle au DOS à l'aide de l'instruction (langage machine) IRET. Il est également possible de rendre le contrôle au DOS à l'aide d'une instruction (langage machine) FAR RETURN mais dans ce cas la réaction du DOS dépendra de l'état du flag Carry. Si ce flag est mis, le DOS interrompra l'exécution du programme. S'il est au contraire annulé, rien ne se passera et l'exécution du programme reprendra normalement.

Lorsqu'un programme modifie le contenu de ce vecteur d'interruption, il peut parfaitement arriver qu'il soit terminé sans avoir restauré l'ancien contenu de ce vecteur. Or il se peut que la mémoire RAM qu'il occupait ait été à nouveau libérée et qu'elle soit réemployée par d'autres programmes, auquel cas la routine Control-C risque d'avoir été effacée par d'autres données. Il peut en résulter un plantage du système si un caractère Control-C apparaît, puisqu'un code tout à fait inadapté figure maintenant à la place de l'ancienne routine Control-C.

Pour éviter cela, le DOS sauvegarde le contenu de ce vecteur d'interruption dans le PSP du programme à exécuter avant de lui passer la main. Lorsqu'un programme est terminé, le contenu du vecteur d'interruption est tout d'abord recopié du PSP dans le vecteur, après quoi le contrôle est rendu au programme-père.

Interruption 24h DOS
Erreur critique

Cette interruption, ou plutôt le vecteur d'interruption correspondant, est également utilisée pour stocker l'adresse d'une routine. Il s'agit ici de la routine qui est appelée chaque fois qu'une erreur critique est détectée lors d'un accès à l'électronique, lors d'un accès disquette par exemple.

Il serait donc possible d'appeler cette routine à l'aide de l'instruction (langage machine) INT mais cela ne doit jamais être tenté car cette routine est conçue comme une procédure FAR et non comme une routine d'interruption.

Un programme peut, au cours de son exécution, modifier le contenu de ce vecteur d'interruption, qui est stocké à partir de la cellule de mémoire 0000:0090, pour que ce ne soit plus la routine du DOS mais une routine du programme qui soit appelée lorsqu'une erreur dece type se produit. C'est un moyen, par exemple, d'empêcher que son exécution puisse être interrompue par la routine du DOS lorsque le programme doit accéder au lecteur de disquette alors qu'aucune disquette n'y est placée. Lorsque cette routine est appelée à la suite d'une erreur critique, le bit 7 du registre AH indique le type d'erreur. S'il vaut 0, il s'agit d'une erreur disquette ou disque dur. Pour toute autre erreur, il vaudra 1. Une erreur disquette/disque dur ne sera cependant signalée qu'après plusieurs tentatives infructueuses pour accéder au périphérique. Les 8 bits inférieurs du registre DI (le contenu des 8 bits supérieurs étant indéfini) contiennent lors de l'appel de la routine un code qui décrit plus précisément l'erreur survenue. Les codes suivants peuvent se présenter :

0 : Disquette protégée contre l'écriture
1 : Accès à un périphérique inconnu
2 : Lecteur n'est pas prêt
3 : Instruction inconnue
4 : Erreur CRC
5 : Largeur de données incorrecte
6 : Recherche infructueuse
7 : Type de périphérique inconnu
8 : Secteur non trouvé
9 : Plus de papier sur l'imprimante
10 : Erreur d'écriture
11 : Erreur de lecture
12 : Erreur générale

La routine d'erreur doit veiller à ce qu'une fois qu'elle se sera achevée les registres SS, SP, DS, ES, BX, CX et DX contiennent les mêmes valeurs que lorsqu'elle avait été appelée. Au cours de son exécution, elle ne doit, d'autre part, accéder qu'aux fonctions 1 et 12 de l'interruption 21h. Il convient de la terminer par une instruction (langage machine) IRET en renvoyant au DOS, dans le registre AL, un code qui guidera la conduite ultérieure du DOS. Le DOS accepte les codes suivants :

0 : Ignorer l'erreur
1 : Répéter l'opération (l'accès)
2 : Terminer programme en appelant l'interruption 23h
3 : Interrompre simplement l'appel de fonction actuel (cette dernière possibilité n'existe qu'à partir de la version 3)

Lorsqu'un programme modifie le contenu de ce vecteur d'interruption, il peut parfaitement arriver qu'il soit terminé sans avoir restauré l'ancien contenu de ce vecteur. Or il se peut que la mémoire RAM qu'il occupait ait été à nouveau libérée et qu'elle soit réemployée par d'autres programmes, auquel cas la routine d'erreur risque d'avoir été effacée par d'autres données. Il peut en résulter un plantage du système si une erreur survient, puisqu'un code tout à fait inadapté figure maintenant à la place de l'ancienne routine d'erreur.

Pour éviter cela, le DOS sauvegarde le contenu de ce vecteur d'interruption dans le PSP du programme à exécuter avant de lui passer la main. Lorsqu'un programme est terminé, le contenu du vecteur d'interruption est tout d'abord recopié du PSP dans le vecteur, après quoi le contrôle est rendu au programme-père.

Interruption 25h DOS
Lecture absolue

Cette interruption permet de lire un ou plusieurs secteurs logiques consécutifs sur une disquette ou sur un disque dur. Cette lecture peut accéder à tous les secteurs du support et non pas seulement à la zone de fichiers du DOS au sens strict. Le DOS utilise lui-même cette interruption pour lire le répertoire racine et la FAT d'un support. Les données sont transférées du support dans un buffer du programme d'appel.

Entrée :
AL = Désignation de périphérique
CX = Nombre de secteurs à lire
DX = Premier secteur à lire
DS = Adresse de segment du buffer
BX = Adresse d'offset du buffer

Sortie :
Flag Carry =0 : Tout va bien
Flag Carry =1 : Erreur, dans ce cas AX = Code d'erreur
1 : Instruction incorrecte
2 : Marque d'adresse incorrecte
4 : Secteur non trouvé
8 : Erreur DMA
16 : Erreur CRC
32 : Erreur du contrôleur de disque
64 : Recherche infructueuse
128 : Périphérique ne répond pas

Remarques :

- Pour la désignation de périphérique, 0 représente le lecteur A, 1 le lecteur B, etc...
- Après appel de cette fonction, le contenu de tous les registres peut avoir été modifié hormis celui des registres de segment.
- Après appel de cette interruption, le pointeur de pile n'a pas la même position que lors de l'appel car deux octets qui avaient été placés sur la pile n'en sont pas retirés. Ces octets représentent le registre de flags, qui peut être retiré de la pile à l'aide de l'instruction (langage machine) POPF. Il suffit donc d'ajouter 2 à la valeur du pointeur de pile pour qu'il retrouve sa valeur d'origine. Si cette correction est négligée, la pile risque de croître sans cesse, jusqu'à déborder. C'est la raison pour laquelle cette interruption ne peut malheureusement être appelée à l'aide de fonctions de langages évolués pour appeler des interruptions, telles que celles que nous vous avons présentées au début de cet ouvrage.
- Le fonctionnement de cette fonction a été modifié sous la version DOS 4.0 ! Voyez à ce sujet le chapitre 6.16.

Interruption 26h DOS
Ecriture absolue

Cette interruption permet d'écrire un ou plusieurs secteurs logiques consécutifs sur une disquette ou sur un disque dur. Cette écriture peut accéder à tous les secteurs du support et non pas seulement à la zone de fichiers du DOS au sens strict. Le DOS utilise lui-même cette interruption pour écrire le répertoire racine et la FAT d'un support. Les données sont transférées d'un buffer du programme d'appel vers le support.

Entrée :
AL = Désignation de périphérique
CX = Nombre de secteurs à écrire
DX = Premier secteur à écrire
DS = Adresse de segment du buffer
BX = Adresse d'offset du buffer

Sortie :
Flag Carry =0 : Tout va bien
Flag Carry =1 : Erreur, dans ce cas AX = Code d'erreur
1 : Instruction incorrecte
2 : Marque d'adresse incorrecte
4 : Secteur non trouvé
8 : Erreur DMA
16 : Erreur CRC
32 : Erreur du contrôleur de disque
64 : Recherche infructueuse
128 : Périphérique ne répond pas

Remarques :

- Pour la désignation de périphérique, 0 représente le lecteur A, 1 le lecteur B, etc...
- Après appel de cette fonction, le contenu de tous les registres peut avoir été modifié hormis celui des registres de segment.
- Après appel de cette interruption, le pointeur de pile n'a pas la même position que lors de l'appel car deux octets qui avaient été placés sur la pile n'en sont pas retirés. Ces octets représentent le registre de flags, qui peut être retiré de la pile à l'aide de l'instruction (langage machine) POPF. Il suffit donc d'ajouter 2 à la valeur du pointeur de pile pour qu'il retrouve sa valeur d'origine. Si cette correction est négligée, la pile risque de croître sans cesse, jusqu'à déborder. C'est la raison pour laquelle cette interruption ne peut malheureusement être appelée à l'aide de fonctions de langages évolués pour appeler des interruptions, telles que celles que nous vous avons présentées au début de cet ouvrage.
- Le fonctionnement de cette fonction a été modifié sous la version DOS 4.0 ! Voyez à ce sujet le chapitre 6.16.

Interruption 27h DOS
Terminer programme mais laisser en mémoire

Cette fonction permet de terminer le programme exécuté et de rendre le contrôle au programme qui avait appelé le programme actuel.

Comme lorsqu'est appelée la fonction 31h de l'interruption du DOS 21h, cette fonction ne libère cependant pas la mémoire occupée par le programme, qui reste donc résident dans la mémoire.

Entrée : CS = Adresse de segment du PSP
DX = Nombre d'octets à réserver + 1

Sortie : aucune

Remarques :

- Cette fonction convient uniquement à l'appel de programmes COM.
- Le nombre d'octets à réserver se rapporte au commencement du PSP.
- Les blocs de mémoire qui ont été réservés à l'aide de la fonction 48h ne sont pas affectés par la valeur dans le registre DX car ils ne peuvent être à nouveau libérés qu'en appelant la fonction 49h.
- Il est préférable d'appeler la fonction 31h de l'interruption 21h plutôt que cette interruption.

- Une erreur apparaîtra si cette interruption est appelée avec une valeur entre FFF1h et FFFFh dans le registre DX.
- Les fichiers ouverts ne sont pas refermés par l'appel de cette interruption.

E. L'interruption multiplexeur 2Fh

L'interruption 2Fh est appelée multiplexeur car elle constitue une interface d'accès aux différentes programmes résidents sous DOS. Elle permet notamment aux programmes résidents de mettre leurs fonctions à la disposition des autres programmes

Interruption 2Fh, code MUX 01h, Fonction 00h
PRINT : Teste l'état d'installation

Grâce à cette fonction un programme peut détecter si la partie résidente de la commande PRINT de DOS est chargée.

Entrée : AH = 01h
Al = 00h

Sortie : AL = FFh : PRINT installé

Remarque :
- Lorsque PRINT est installé, la fonction retourne la valeur FFh en AL. Ce n'est qu'à cette condition que les autres fonctions de PRINT pourront être invoquées. L'exploitation des fonctions de PRINT suppose donc que cette fonction ait été déclenchée au préalable.

Interruption 2Fh, code MUX 01h, fonction 01h
PRINT : Ajoute un fichier à la file d'attente

Cette fonction permet à un programme d'ajouter un fichier à la fin de la file en attente d'impression pour qu'il soit imprimé par PRINT en tâche de fond.

Entrée : AH = 00h
Al = 01h
DS:DX = PPointeur FAR sur une structure de données contenant des informations sur le fichier (cf infra)

Sortie : Indicateur de retenue
= 0 : ok
=1 : erreur, dans ce cas AX=code d'erreur (cf infra)

Remarque :
- La structure de données référencée par DS:DX comporte 5 octets et doit être conforme au modèle suivant :

Offset	Signification	Type
00h	Tjs 0	1 Byte
01h	Pointeur FAR sur chaîne ASCIIZ contenant le nom du fichier	1 Pointeur FAR

Les caractères génériques ne sont pas autorisés dans le nom, mais les désignations de lecteur et de chemin d'accès sont permises.

En cas d'erreur, le code retourné est le suivant :

0001h	Fonction inconnue
0002h	Fichier non trouvé
0003h	Chemin d'accès non trouvé
0004	Trop de fichiers ouverts
0005h	Accès au fichier refusé
0008h	File d'attente saturée
000Fh	Périphérique inconnu

Cette fonction ne doit être invoquée que si un appel à la fonction 00h a prouvé que la partie résidente de PRINT était installée

Interruption 2Fh, code MUX 01h, fonction 02h
PRINT : Retirer un fichier de la file d'attente

Grâce à cetet fonction un ou plusieurs fichiers peuvent être supprimés dans la file d'attente. Même le fichier en cours d'impression peut être retiré, ce qui arrête évidemment son impression.

Entrée : AH = 01h
AL = 02h
DS:DX = Pointeur sur nom de fichier sous forme de chaîne ASCIIZ

Sortie : Indicateur de retenue
=0 : ok
=1 : Erreur, dans ce cas
AX= code d'erreur (cf infra)

Remarque :
- Les caractères génériques sont permis dans le nom de fichier indiqué, ce qui permet en un seul appel de retirer plusieurs fichiers de la file d'attente
- Le seul code d'erreur possible est 0002h, "Fichier non trouvé", qui indique que le fichier en question n'est pas dans la file d'attente.
- Cette fonction ne doit être invoquée que si un appel à la fonction 00h a prouvé que la partie résidente de PRINT était installée

Interruption 2Fh, code MUX 01h, fonction 03h
PRINT : Supprime la file d'attente

Tous les fichiers sont retirés de la file d'attente et l'impression du fichier en cours est stoppée.

Entrée : AH = 01h
AL = 03h

Sortie : néant

Remarque :
- Cette fonction ne doit être invoquée que si un appel à la fonction 00h a prouvé au préalable que la partie résidente de PRINT est déjà installée

Interruption 2Fh, code MUX 01h, fonction 04h
PRINT : Interrompt l'impression pour tester son état

Cette fonction permet de connaître le contenu courant de la file d'attente et le nombre d'erreurs survenues

Entrée : AH = 01h
AL = 04h

Sortie : DX = Compteur d 'erreurs
DS:SI = pointeur FAR sur la file d'attente

Remarque :
- L'impression est interrompue et ne reprendra que si on fait appel à la fonction 05h.
- Le pointeur retourné en DS:SI référence le buffer dans lequel DOS maintient la file d'attente comme une suite d'éléments de 64 octets. Chacun de ces éléments contient la chaîne ASCIIZ du nom du fichier à imprimer. Le dernier élément de la liste commence par un caractère nul.

- La liste peut être inspectée mais elle ne doit pas être modifiée.
- Cette fonction ne doit être invoquée que si un appel à la fonction 00h a prouvé au préalable que la partie résidente de PRINT est déjà installée

Interruption 2Fh, code MUX 01h, fonction 05h

Cette fonction permet de reprendre l'impression après interruption par la fonction 04h.

Entrée : AH = 01h
AL = 05h

Sortie : néant

Remarque :
- L'usage de cette fonction n'a évidemment de sens que si on a invoqué précédemment la fonction 04h

Interruption 2Fh, code MUX 01h, fonction 06h
PRINT : Détermine l'imprimante en service

Tant qu'il reste au moins un fichier dans la file d'attente, cette fonction indique l'imprimante qui effectue l'impression.

Entrée : AH = 01h
AL = 06h

Sortie : Indicateur de retenue
=0 : file d'attente vide
=1 : File non vide, dans ce cas AX=0008h
DS:SI = Pointeur FAR sur l'en-tête du driver de périphérique

Remarque :
- La manière dont cette fonction retourne son résultat est un peu bizarre. L'information cherchée, à savoir le pointeur sur l'en-tête du périphérique, n'est obtenue qu'en liaison avec un indicateur de retenue armé et le code d'erreur 0008 (file non vide). Si la file d'attente est vide et si de ce fait le pointeur n'est pas disponible, l'indicateur de retenue est désarmé.

Interruption 2Fh, code MUX 06h, fonction 00h
ASSIGN : Teste l'état d'installation

Grâce à cette fonction un programme peut détecter si la partie résidente de la commande ASSIGN est chargée.

Entrée : AH = 06h
AL = 00h

Sortie : AL = FFh : ASSIGN installé

Interruption 2Fh, code MUX 10h, fonction 00h
SHARE : Teste l'état d'installation

Cette fonction permet de tester la présence de la partie résidente de la commande SHARE de DOS.

Entrée : AH = 10h
AL = 00h

Sortie : Al =FFh : SHARE installé

Interruption 2Fh, code MUX 1Ah, fonction 00h
ANSI.SYS : Teste l'état d'installation

Cette fonction permet de tester la présence du driver de périphérique ANSI.SYS

Entrée : AH = 1Ah
AL = 00h

Sortie : Al =FFh : ANSI.SYS installé

Interruption 2Fh, code MUX 43h, fonction 00h
HIMEM.SYS : Teste l'état d'installation

Cette fonction permet de tester la présence du driver de périphérique HIMEM.SYS qui permet d'accéder à la mémoire étendue selon le standard XMS.

Entrée : AH = 43h
AL = 00h

Sortie : AL =80h: HIMEM.SYS installé

Remarque :
- Notez bien que cette fonction contrairement à ses homologues qui effectuent également un test d'installation ne renvoie pas la valeur FFh en cas de succès mais la valeur 80h.
- Pour assurer le compatibilité avec HIMEM.SYS, cet appel est aussi supporté par d'autres gestionnaires de mémoire XMS.

Interruption 2Fh, code MUX 43h, fonction 10h
HIMEM.SYS : Indique l'adresse d'appel des fonctions XMS

Contrairement à ce qui se passe pour les autres gestionnaires de mémoire, les fonctions XMS de HIMEM.SYS ne sont pas déclenchées par des interruptions mais par une routine FAR CALL dont l'adresse est obtenue par le moyen de la présente fonction.

Entrée : AH = 43h
AL = 10h

Sortie : ES:BX = Pointeur FAR pour appeler les fonctions XMS

Remarque :
- Cette fonction ne doit être invoquée que si un appel à la fonction 00h a prouvé au préalable que HIMEM.SYS est déjà installé

Interruption 2Fh, code MUX 48h, fonction 00h
DOSKEY : Teste l'état d'installation

Cette fonction permet de savoir si le programme de DOS DOSKEY.COM est chargé

Entrée : AH = 48h
AL = 00h

Sortie : AL =00h : non installé

Remarque :
- DOSKEY n'existe que depuis la version 5.0 de DOS
- Notez qu'à l'inverse de fonctions semblables seule est définie la valeur qui indique que le programme DOSKEY.COM n'est pas chargé!

Interruption 2Fh, code MUX 48h, fonction 10h
DOSKEY : Prend en compte la saisie d'un utilisateur

Entrée : AH = 48h
AL = 10h
DS:DX = Pointeur sur structure de données (cf infra)

Sortie : néant

Remarque : ■ La structure de données référencée par le pointeur en DS:DX s'étend sur 130 octets et mémorise entre autres les saisies de l'utilisateur. Elle doit être conforme au modèle suivant :

Offset	Signification	Type
00h	Taille du buffer de saisie doit être 128	1 OCTET
01h	Nombre de caractères lus - 1	1 OCTET
02h	Buffer de saisie	128 OCTETS

Le nombre de caractères lus est reporté par DOSKEY dans la structure de données. Avant appel à DOSKEY, ce champ, contrairement à celui de la taille, n'a pas besoin d'être initialisé.

Interruption 2Fh, code MUX ADh, fonction 80h
KEYB.COM : Détermine le numéro de la version

Cette fonction indique le numéro de la version du driver de clavier de DOS appelé KEYB.COM

Entrée : AH = ADh
AL = 80h
BX =0

Sortie : BH = partie majeure du numéro de la version
BL = partie mineure du même numéro

Remarque : ■ Si BH et BL sont égaux à 0, c'est que KEYB.COM n'est pas installé

Interruption 2Fh, code MUX ADh, fonction 81h
KEYB.COM : Fixe la page de codes active

Cette fonction permet de fixer la page de codes active

Entrée : AH = ADh
AL = 81
BX = Numéro de la page de codes

Sortie : Indicateur de retenue
= 0 : ok
=1 : erreur. Dans ce cas
AX=0001 "Page de codes inconnue"

Sortie : néant

Remarque :

- Les pages de codes suivantes sont reconnues par KEYB

Code	Jeu de caractères
437	USA
850	Multilingue (tous pays européens)
860	Portugal
863	Canada français
865	Scandinavie

- Cette fonction ne devrait être invoquée que si un appel à la fonction 80h a prouvé au préalable que KEYB.COM est installé
- Pour en savoir davantage sur les pages de codes consultez votre manuel de DOS car dans le cadre de cet ouvrage les pages de codes ne jouent aucun rôle.

Interruption 2Fh, code MUX boit h, fonction 00h
GRAFTABL : Teste l'état d'installation

Cette fonction permet de savoir si la partie résidente de la commande GRAFTABL de DOS est chargée.

Entrée : AH = B0h
AL = 00h

Sortie : AI = FFh : GRAFTABL chargé

Interruption 2Fh, code MUX B7h, fonction 00h

Cette fonction permet de savoir si la partie résidente de la commande APPEND de DOS est chargée.

Entrée : AH = B7h
AL = 00h

Sortie : AL = FFh : APPEND chargé

Interruption 2Fh, code MUX B7h, fonction 02h
APPEND : Détermine la compatibilité avec DOS 5

Entrée : AH = B7h
AL = 02h

Sortie : AX = FFFFh : compatible avec DOS 5

Remarque :
- Cette fonction ne doit être invoquée que si un appel à la fonction 00h a prouvé au préalable que APPEND est bien installé

Interruption 2Fh, code MUX B7h, fonction 04h
APPEND : Indique la liste des répertoires APPEND

Grâce à cette fonction on peut prendre connaissance des différents répertoires confiés à APPEND pour la recherche des fichier.

Entrée : AH = B7h
AL = 04h

Sortie : ES:DI = Pointeur FAR sur le buffer des répertoires APPEND

Remarque :
- Le buffer référencé par le pointeur en ES:DI contient les répertoires sous forme de chaîne ASCIIZ. Commme c'est le cas pour la commande PATH, les différentes répertoires sont séparés par des points virgules.
- Le contenu de ce buffer ne doit pas être modifié par celui qui l'inspecte

- Cette fonction ne doit être invoquée que si un appel à la fonction 00h a prouvé au préalable que APPEND est bien chargé

Interruption 2Fh, code MUX B7h, fonction 06h
APPEND : Détermine le mode opératoire

Cette fonction permet de prendre connaissance des différents modes opératoires fixés par les paramètres de la ligne de commande au moment du chargement de APPEND.

Entrée : AH =B7h
AL =06h

Sortie : BX = Indicateur APPEND

Remarque :

- Les différents bits de l'indicateur retourné représentent les modes opératoires. Les bits non mentionnés dans le tableau ci-dessous n'ont pas de signification et sont à 0 :

Bit	Signification
0	1 = APPEND est actif
12	1 = Les répertoires APPEND ne sont pris en compte que si les noms des fichiers cherchés comportent des désignations explicites d'unité correspondant exactement à celles transmises à APPEND
13	1 = commutateur /PATH:ON actif
14	1 = commutateur /E actif
15	1 = commutateur /X:ON actif

- Cette fonction ne doit être invoquée que si un appel à la fonction 00h a prouvé au préalable que APPEND est bien chargé.

Interruption 2Fh, code MUX B7h, fonction 07h
APPEND : Fixe le mode opératoire

Cette fonction est symétrique de la fonction 06h et permet de fixer les paramètres de fonctionnement de APPEND

Entrée : AH = B7h
AL = 07h
BX = Indicateur APPEND

Sortie : néant

Remarque :
- Voir ci-dessus la signification de l'indicateur APPEND
- Cette fonction ne doit être invoquée que si un appel à la fonction 00h a prouvé au préalable que APPEND est bien chargé

F. Description des fonctions de l'EMM

Le standard EMS inventé par les sociétés Lotus, Intel et Microsoft existe déjà depuis près de dix années. Il régule l'accès aux cartes de mémoire étendue travaillant selon le principe du Banking et autorisant toujours l'accès à une petite partie de la totalité de la mémoire étendue.

La plupart des cartes EMS reconnaissent la version 3.2 de ce standard et par conséquent les fonctions EMS définies sous les versions 3.0 et 3.2. Très peu de cartes de mémoire étendue reconnaissent la spécification EMS 4.0 et les fonctions qui s'y rapportent. Cela ne s'applique pas seulement à la majorité des émulateurs EMS actuellement commercialisés pour simuler la mémoire EMS via la mémoire étendue. Généralement, ils sont conçus à partir de la version 4.0.

Interruption 67h, Fonction 40h LIM/EMS (à partir de 3.0)
Déterminer état

Cette fonction détermine l'état (d'erreur) de l'EMM (Expanded Memory Manager), tel qu'il est également renvoyé après appel de toutes les autres fonctions EMS.

Entrée : AH = 40h

Sortie : AH = état de l'EMM
- 00h : tout va bien
- 80h : erreur interne, éventuellement EMM détruit
- 81h : défaut de fonctionnement de l'électronique EMS
- 82h : EMM occupé
- 83h : Handle incorrect
- 84h : Fonction appelée non reconnue
- 85h : Plus de handles disponibles
- 86h : Erreur lors de la sauvegarde ou le renvoi de la concordance entre pages logiques et physiques
- 87h : Le nombre de pages demandées est supérieur au nombre de pages physiques disponibles
- 88h : Le nombre de pages logiques demandées est supérieur au nombre actuellement disponible
- 89h : Tentative d'allouer 0 page
- 8Ah : Numéro de page logique non valable
- 8Bh : Numéro de page physique non valable
- 8Ch : Cellule de mémoire devant stocker la table de sauvegarde pleine
- 8Dh : La zone de sauvegarde contient déjà une table réservée au handle spécifié
- 8Eh : Il n'existe aucune table de sauvegarde pour le handle spécifié

8Fh : Erreur dans la spécification du numéro de la sous-fonction

Erreur pouvant survenir à partir de la version 4.0 de l'EMS :

90h : Type d'attribut non défini
91h : Caractéristique non reconnue
92h : Superposition entre les zones source et cible, une partie de la zone est écrasée
93h : La longueur spécifiée est supérieure à la longueur réelle de la zone allouée
94h : La zone conventionnelle et la zone de la mémoire EMS s'entrecroisent
95h : L'offset indiqué se situe en dehors de la page logique
96h : Le nombre d'octets à traiter est supérieur à 1 Mo
97h : Les zones source et cible possèdent le même handle, l'échange ne peut pas avoir lieu
98h : Type de zone mémoire non défini pour la source ou la cible
99h : Code d'erreur inutilisé
9Ah : Le groupe de registres de remplacement spécifié n'est pas reconnu
9Bh : Tous les groupes de registres de remplacement sont alloués
9Ch : Les groupes de registres ne sont pas reconnus et le groupe de registre ne correspond pas au numéro zéro
9Dh : Le groupe de registres de remplacement spécifié n'est pas défini ou alloué
9Eh : Les canaux DMA dédiés ne sont pas reconnus
9Fh : Le canal DMA dédié n'est pas valable
A0h : Pas de handle trouvé sous le nom spécifié
A1h : Un handle existe déjà sous ce nom
A2h : Débordement d'adresses au-delà de 1 Mo
A3h : Pointeur spécifié ou contenu du buffer désigné non valable
A4h : Accès à la fonction interrompu par le système d'exploitation

Remarques :

- Comme toutes les autres fonctions de l'EMM, cette fonction ne doit être appelée qu'après s'être assuré qu'une mémoire EMM de la version 3.0 ou supérieure est effectivement installée. Voyez aussi le chapitre 12 à ce propos.
- Il est conseillé d'appeler cette fonction avant de commencer à travailler avec la mémoire EMS car cela permet de constater facilement si le matériel et le logiciel correspondants fonctionnent sans problème.

Interruption 67h, Fonction 41h LIM/EMS (à partir de 3.0)
Déterminer l'adresse de segment du Page Frame

Cette fonction détermine l'adresse de segment du Page Frame.

Entrée : AH = 41h

Sortie : AH = 0, o.k, auquel cas :
BX = adresse de segment du Page Frame
AH > 0, erreur, cf. fonction 40h

Remarques :
- Comme toutes les autres fonctions de l'EMM, cette fonction ne doit être appelée qu'après s'être assuré qu'une mémoire EMM de la version 3.0 ou supérieure est effectivement installée. Voyez aussi le chapitre 12 à ce propos.
- L'adresse renvoyée ainsi que la fonction 58h/01h (Déterminer le nombre de pages EMS physiques, à partir de 4.0) permettent de calculer les adresses des autres pages physiques, la première page commençant à l'adresse PAGE FRAME:0000. Les autres pages suivent ensuite à intervalles de 16 Ko. Si le numéro de version de l'EMM est inférieur à 4.0, le nombre des pages physiques EMS est alors toujours de 4.

Interruption 67h, Fonction 42h LIM/EMS (à partir de 3.0)
Déterminer nombre de pages EMS

Cette fonction indique au programme d'appel combien de pages EMS (de 16 Ko chacune) sont installées et combien sont encore libres, c'est-à-dire combien n'ont pas encore été allouées.

Entrée : AH = 42h

Sortie : AH = 0, o.k, auquel cas :
BX = Nombre de pages libres (non encore allouées)
DX = Nombre total de pages EMS
AH > 0, erreur, cf. Fonction 40h

Remarques :
- Comme toutes les autres fonctions de l'EMM, cette fonction ne doit être appelée qu'après s'être assuré qu'une mémoire EMM de la version 3.0 ou supérieure est effectivement installée. Voyez aussi le chapitre 12 à ce propos.
- Pour calculer la taille en Ko de la mémoire EMS libre, on multiplie le nombre de pages libres par 16.

Interruption 67h, Fonction 43h LIM/EMS (à partir de 3.0)
Allouer Mémoire EMS

Cette fonction permet à un programme de se faire allouer un certain nombre de pages (de 16 Ko) pour ses opérations.

Entrée : AH = 43h
BX = Nombre de pages logiques à allouer (de 16 Ko)

Sortie : AH = 0, o.k, auquel cas :
DX = Handle pour l'accès à la mémoire allouée
AH > 0, erreur, cf. Fonction 40h

Remarques :

- Comme toutes les autres fonctions de l'EMM, cette fonction ne doit être appelée qu'après s'être assuré qu'une mémoire EMM de la version 3.0 ou supérieure est effectivement installée. Voyez aussi le chapitre 12 à ce propos.
- Le handle renvoyé servira ensuite à accéder à, puis à libérer la mémoire allouée. Il doit donc être mémorisé par l'utilisateur. Si ce handle est "perdu" après appel de la fonction, le programme d'appel n'aura plus aucune possibilité de se le faire communiquer à nouveau, de sorte que la mémoire allouée ne pourra plus être libérée pour être à nouveau rendue accessible à d'autres programmes.
- L'appel de cette fonction peut échouer si le nombre de pages réclamées est supérieur au nombre de pages libres ou bien si, bien qu'il reste encore suffisamment de pages, cette fonction a été appelée tellement souvent que l'EMM ne peut plus fournir d'autre handle.
- Les handles portent normalement les numéros FF00h, FE01h, FD02h, FC03h etc.

Interruption 67h, Fonction 44h LIM/EMS (à partir de 3.0)
Fixer Mapping

Cette fonction permet de faire incruster l'une des pages précédemment allouées à l'aide de la fonction 4h dans l'une des 4 pages physiques du Page Frame.

Entrée : AH = 44h
AL = Numéro de page physique (0 à 3)
BX = Numéro de page logique
DX = Handle

Sortie : AH = Etat d'erreur
00h : tout va bien
AH > 0, Erreur, cf. Fonction 40h

Remarques :

- Comme toutes les autres fonctions de l'EMM, cette fonction ne doit être appelée qu'après s'être assuré qu'une mémoire EMM de la version 3.0 ou supérieure est effectivement installée. Voyez aussi le chapitre 12 à ce propos.
- Le handle à spécifier lors de l'appel de cette fonction doit avoir été obtenu à la suite d'un appel antérieur de la fonction 43h de l'EMM.
- Les pages logiques sont numérotées à partir de 0, de sorte que la valeur 0 doit être spécifiée pour accéder à la première page allouée. La plus grande valeur autorisée est donc égale au nombre de pages allouées moins 1.
- Pour pouvoir accéder à la page physique, il faut connaître l'adresse de segment du Page Frame, qui peut être obtenue à l'aide de la fonction 41h.

Interruption 67h, Fonction 45h LIM/EMS (à partir de 3.0)
Libérer Pages

Cette fonction permet de restituer à l'EMM et de rendre ainsi accessibles à d'autres programmes les pages précédemment allouées à l'aide de la fonction 43h.

Entrée : AH = 45h
DX = Handle

Sortie : AH = Etat d'erreur
00h : tout va bien
AH > 0, Erreur, cf. Fonction 40h

Remarques :

- Comme toutes les autres fonctions de l'EMM, cette fonction ne doit être appelée qu'après s'être assuré qu'une mémoire EMM de la version 3.0 ou supérieure est effectivement installée. Voyez aussi le chapitre 12 à ce propos.
- Le handle à spécifier lors de l'appel de cette fonction doit avoir été obtenu à la suite d'un appel antérieur de la fonction 4 de l'EMM.
- L'appel de cette fonction libère toutes les pages allouées à travers ce handle. Il n'est pas possible de sélectionner les pages à libérer. Cependant, la fonction 51h (Modifier le nombre de pages logiques allouées, à partir de 4.0) permet d'ajouter ou de retirer une ou plusieurs pages parmi les pages déjà allouées.

- Après un appel réussi de cette fonction, le handle spécifié n'est plus valable et ne peut donc plus être employé pour accéder à la mémoire EMS.
- Si la fonction revient avec un message d'erreur, il convient de répéter au moins trois fois l'appel de cette fonction car sinon les pages qui n'ont pu être libérées resteraient allouées une fois le programme terminé et ne pouraient donc être remises à la disposition d'autres programmes.

Interruption 67h, Fonction 46h LIM/EMS (à partir de 3.0)
Déterminer la version EMM

Cette fonction permet d'obtenir le numéro de version de l'EMM (Expanded Memory Manager).

Entrée : AH = 46h

Sortie : AH = 0, o.k, auquel cas :
AL = numéro de version EMM
AH > 0, erreur, cf. Fonction 40h

Remarques :
- Comme toutes les autres fonctions de l'EMM, cette fonction ne doit être appelée qu'après s'être assuré qu'une mémoire EMM de la version 3.0 ou supérieure est effectivement installée. Voyez aussi le chapitre 12 à ce propos.
- Le numéro de version EMM figure dans le registre AL sous forme d'un nombre BCD, le numéro de version principale (le chiffre avant le point) étant placé dans les 4 bits de plus fort poids et le numéro de sous-version (le chiffre après le point) étant placé dans les 4 bits de plus faible poids. Voyez à ce propos les programmes d'exemple du chapitre 14.

Interruption 67h, Fonction 47h LIM/EMS (à partir de 3.0)
Sauvegarder Mapping

En appelant cette fonction, vous faites sauvegarder la concordance actuelle entre les quatre pages physiques du Page Frame et les pages logiques qui leur ont été attribuées.

Entrée : AH = 47h
DX = Handle

Sortie : AH = Etat d'erreur
00h : tout va bien
AH > 0, Erreur, cf. Fonction 40h

Remarques :
- Comme toutes les autres fonctions de l'EMM, cette fonction ne doit être appelée qu'après s'être assuré qu'une mémoire EMM de la version 3.0 ou supérieure est effectivement installée. Voyez aussi le chapitre 12 à ce propos.
- Le handle à spécifier lors de l'appel de cette fonction doit avoir été obtenu à la suite d'un appel antérieur de la fonction 43h de l'EMM.
- Cette fonction est conçue pour être appelée par des programmes TSR ou le système d'exploitation dans le cadre d'un environnement multitâche mais elle peut aussi être utilisée par n'importe quel autre programme.

Interruption 67h, Fonction 48h LIM/EMS (à partir de 3.0)
Rétablir Mapping

Cette fonction rétablit la concordance entre pages logiques et physiques (Mapping) qui avait été précédemment sauvegardée à l'aide de la fonction 47h.

Entrée : AH = 48h
DX = Handle

Sortie : AH = Etat d'erreur
00h : tout va bien
AH > 0, Erreur, cf. Fonction 40h

Remarques :
- Comme toutes les autres fonctions de l'EMM, cette fonction ne doit être appelée qu'après s'être assuré qu'une mémoire EMM de la version 3.0 ou supérieure est effectivement installée. Voyez aussi le chapitre 12 à ce propos.
- Le handle à spécifier lors de l'appel de cette fonction doit avoir été obtenu à la suite d'un appel antérieur de la fonction 42h de l'EMM.
- Cet appel est toujours infructueux lorsque le Mapping correspondant au handle spécifié n'a pas encore été sauvegardé à l'aide de la fonction 47h ou bien s'il avait déjà été rétabli au cours d'un appel antérieur de la fonction 48h.
- Cette fonction est conçue pour être appelée par des programmes TSR ou le système d'exploitation dans le cadre d'un environnement multitâche mais elle peut aussi être utilisée par n'importe quel autre programme.

Interruption 67h, Fonction 49h LIM/EMS (à partir de 3.0)
Non documentée

Le rôle et les effets de cette fonction ne sont pas connus. Elle ne s'utilise que sur le plan interne dans l'EMM.

Interruption 67h, Fonction 4Ah LIM/EMS (à partir de 3.0)
Non documentée

Le rôle et les effets de cette fonction ne sont pas connus. Elle ne s'utilise que sur le plan interne dans l'EMM.

Interruption 67h, Fonction 4Bh LIM/EMS (à partir de 3.0)
Déterminer le nombre de handles

Cette fonction permet de connaître le nombre de zones de mémoire ayant été actuellement allouées à l'aide de la fonction 4h, autrement dit le nombre de handles attribués.

Entrée : AH = 4Bh

Sortie : AH = 0, o.k, auquel cas :
BX = nombre de handles attribués
AH > 0, erreur, cf. Fonction 40h

Remarques :

- Comme toutes les autres fonctions de l'EMM, cette fonction ne doit être appelée qu'après s'être assuré qu'une mémoire EMM de la version 3.0 ou supérieure est effectivement installée. Voyez aussi le chapitre 12 à ce propos.
- Le nombre de handles attribués n'est pas nécessairement identique au nombre de programmes accédant actuellement à la mémoire EMS car un programme peut réclamer à l'aide de la fonction 43h autant de blocs de mémoire EMS et donc autant de handles qu'il le souhaite.

Interruption 67h, Fonction 4Ch LIM/EMS (à partir de 3.0)
Déterminer le nombre de pages allouées

Cette fonction indique au programme d'appel combien de pages ont été allouées au handle spécifié.

Entrée : AH = 4Ch
DX = Handle

Sortie : AH = 0, o.k, auquel cas :
BX = Nombre de pages logiques allouées
AH > 0, erreur, cf. Fonction 40h

Remarques :
- Comme toutes les autres fonctions de l'EMM, cette fonction ne doit être appelée qu'après s'être assuré qu'une mémoire EMM de la version 3.0 ou supérieure est effectivement installée. Voyez aussi le chapitre 12 à ce propos.
- Le nombre de pages allouées renvoyé doit être compris entre 1 et 512.

Interruption 67h, Fonction 4Dh LIM/EMS (à partir de 3.0)
Déterminer tous les handles

Cette fonction permet de charger dans un tableau tous les handles activés ainsi que le nombre de pages respectivement allouées à chacun.

Entrée : AH = 4Dh
ES:DI = Pointeur sur le tableau

Sortie : AH = 0, o.k, auquel cas :
BX = Nombre de pages logiques allouées
AH > 0, erreur, cf. Fonction 40h

Remarques :
- Comme toutes les autres fonctions de l'EMM, cette fonction ne doit être appelée qu'après s'être assuré qu'une mémoire EMM de la version 3.0 ou supérieure est effectivement installée. Voyez aussi le chapitre 12 à ce propos.
- Si la fonction est exécutée avec succès, la zone de mémoire dont l'adresse avait été transmise dans la paire de registres ES:DI contient deux mots pour chaque handle activé. Le premier mot représente le handle, le mot suivant le nombre de pages allouées à ce handle. Le nombre d'entrées de ce tableau figure après appel de la fonction dans le registre BX.
- Comme l'EMM attribue au maximum 256 handles, le tableau ne peut jamais occuper plus de 1024 octets (soit 1 Ko).

Interruption 67h, Fonction 4Eh, Sous-fonction 00h LIM/EMS (à partir de 3.2)
Sauvegarder Table des pages

Cette fonction permet à un programme de sauvegarder la table des pages décrivant la concordance entre les pages logiques et physiques.

Entrée : AH = 4Eh
AL = 00h
ES:DI = Pointeur sur la tableau devant recevoir la table des pages

Sortie : AH = 0, Tout va bien
AH >0, Erreur, cf. Fonction 40h

Remarques :

- Comme toutes les autres fonctions de l'EMM, cette fonction ne doit être appelée qu'après s'être assuré qu'une mémoire EMM de la version 3.2 ou supérieure est effectivement installée. Voyez aussi le chapitre 12 à ce propos.
- Contrairement à la fonction similaire 47h, ici un handle EMM s'avère inutile.
- Vous pouvez prendre connaissance de la taille du buffer à l'aide de la sous-fonction 03h.
- La codification des informations entrées dans le buffer varie en fonction des constructeurs de cartes et drivers EMS. Il faut donc éviter de le manipuler ou l'interpréter.
- Après avoir modifié la concordance entre les pages EMS physiques et logiques, vous pouvez la rétablir avec la sous-fonction 01h mais en prenant soin de la sauvegarder au préalable avec la sous-fonction 00h.

Interruption 67h, Fonction 4Eh, Sous-fonction 01h LIM/EMS (à partir de 3.2)
Rétablir la table des pages

L'appel de cette fonction rétablit la concordance entre les pages EMS physiques et logiques préalablement définie avec la sous-fonction 00h de cette fonction.

Entrée : AH = 4Eh
AL = 01h
ES:DI = Pointeur sur la tableau contenant la table des pages

Sortie : AH = 0, Tout va bien
AH > 0, Erreur, cf. Fonction 40h

Remarques :

- Comme toutes les autres fonctions de l'EMM, cette fonction ne doit être appelée qu'après s'être assuré qu'une mémoire EMM de la version 3.2 ou supérieure est effectivement installée. Voyez aussi le chapitre 12 à ce propos.
- Contrairement à la fonction 48h de même nature, ici un handle EMM s'avère inutile.

Interruption 67h, Fonction 4Eh, Sous-fonction 02h LIM/EMS (à partir de 3.2)
Sauvegarder et rétablir simultanément la table des pages

Cette fonction permet de sauvegarder simultanément la table des pages en cours et de le charger avec un autre qui a été préalablement sauvegardé.

Entrée : AH = E4h
AL = 02h
DS:SI = Pointeur sur la tableau contenant déjà la table des pages à restaurer
ES:DI = Pointeur sur la tableau devant recevoir la table des pages en cours

Sortie : AH = 0, Tout va bien
AH > 0, Erreur, cf. Fonction 40h

Remarques :

- Comme toutes les autres fonctions de l'EMM, cette fonction ne doit être appelée qu'après s'être assuré qu'une mémoire EMM de la version 3.2 ou supérieure est effectivement installée. Voyez aussi le chapitre 12 à ce propos.
- Le buffer contenant la table des pages à installer doit d'abord être chargé à l'aide d'un appel de la sous-fonction 00h ou 02h de cette fonction.
- Vous pouvez prendre connaissance de la taille du buffer devant recevoir la table des pages en cours à l'aide de la sous-fonction 03h de cette fonction.
- Contrairement aux fonctions 47h et 48h de même nature, ici un handle EMM s'avère inutile.

Interruption 67h, Fonction 4Eh, Sous-fonction 03h LIM/EMS (à partir de 3.2)
Déterminer la taille du Table des pages

Avant l'appel de la sous-fonction 00h ou 02h, vous pouvez utiliser cette fonction pour déterminer la place mémoire nécessaire pour stocker la table des pages.

Entrée : AH = 4Eh
AL = 03h

Sortie : AH = 0, Tout va bien, dans ce cas
AL = Taille du Table des pages en octets
AH > 0, Erreur, cf. Fonction 40h

Remarques :
- Comme toutes les autres fonctions de l'EMM, cette fonction ne doit être appelée qu'après s'être assuré qu'une mémoire EMM de la version 3.2 ou supérieure est effectivement installée. Voyez aussi le chapitre 12 à ce propos.

Interruption 67h, Fonction 4Fh, Sous-fonction 00h LIM/EMS (à partir de 4.0)
Sauvegarder une partie du Table des pages

Cette fonction permet de sauvegarder séparément la structure de pages physiques bien précises et leurs pages logiques correspondantes.

Entrée : AH = 4Fh
AL = 00h
DS:SI = Pointeur sur la liste des informations à propos des pages physiques à construire
ES:DI = Pointeur sur le buffer devant stocker les entrées sélectionnées parmi la table des pages.

Sortie : AH = 0, Tout va bien, dans ce cas
AH > 0, Erreur, cf. Fonction 40h

Remarques :
- Comme toutes les autres fonctions de l'EMM, cette fonction ne doit être appelée qu'après s'être assuré qu'une mémoire EMM de la version 4.0 ou supérieure est effectivement installée. Voyez aussi le chapitre 12 à ce propos.
- La liste des pages physiques à construire doit contenir le nombre des pages concernées dans le premier mot. Suivent ensuite les adresses de segment de ces pages occupant chacune un mot. La taille de cette liste se calcule comme suit :

  2 + (Nombre de pages à construire * 2) octets
- La taille du buffer nécessaire pour recevoir les entrées sélectionnées parmi la table des pages peut être obtenue par la sous-fonction 02h de cette fonction.

Interruption 67h, Fonction 4Fh, Sous-fonction 01h LIM/EMS (à partir de 4.0)
Rétablir une table des pages sauvée partiellement

Cette fonction rétablit une table des pages sauvegardée préalablement à l'aide de la sous-fonction 00h.

Entrée : AH = 4Fh
AL = 01h
DS:SI = Pointeur sur le buffer contenant la table sauvegardée

Sortie : AH = 0, Tout va bien, dans ce cas
AH > 0, Erreur, cf. Fonction 40h

Remarques :
- Comme toutes les autres fonctions de l'EMM, cette fonction ne doit être appelée qu'après s'être assuré qu'une mémoire EMM de la version 4.0 ou supérieure est effectivement installée. Voyez aussi le chapitre 12 à ce propos.
- Le buffer spécifié doit être chargé au préalable avec la sous-fonction 00h de la fonction 4Fh.

Interruption 67h, Fonction 4Fh, Sous-fonction 02h LIM/EMS (à partir de 4.0)
Déterminer la taille du buffer pour recevoir une table des pages partielle

Si vous devez sauvegarder une partie de la table des pages avec la sous-fonction 00h, cette sous-fonction peut vous aider à déterminer d'abord la taille du buffer nécessaire à cet effet.

Entrée : AH = 4Fh
AL = 02h
BX = Nombre de pages physiques à construire lors de la sauvegarde

Sortie : AH = 0, Tout va bien, dans ce cas
AL = Taille du buffer nécessaire en octets
AH > 0, Erreur, cf. Fonction 40h

Remarques :
- Comme toutes les autres fonctions de l'EMM, cette fonction ne doit être appelée qu'après s'être assuré qu'une mémoire EMM de la version 4.0 ou supérieure est effectivement installée. Voyez aussi le chapitre 12 à ce propos.

Interruption 67h, Fonction 50h, Sous-fonction 00h LIM/EMS (à partir de 4.0)
Construire des pages logiques sur des pages physiques en fonction de leurs numéros

En appelant une seule fois cette fonction, un programme peut construire des pages logiques multiples sur un nombre quelconque de pages physiques adressées au moyen de leur numéro respectif.

Entrée : AH = 50h
AL = 00h
CX = Nombre de pages logiques à construire
DX = Handle sur lequel les pages sont à allouer
DS:SI = Pointeur sur le buffer contenant les informations sur la structure

Sortie : AH = 0, Tout va bien, dans ce cas
AH > 0, Erreur, cf. Fonction 40h

Remarques :
- Comme toutes les autres fonctions de l'EMM, cette fonction ne doit être appelée qu'après s'être assuré qu'une mémoire EMM de la version 4.0 ou supérieure est effectivement installée. Voyez aussi le chapitre 12 à ce propos.
- Le buffer spécifié doit contenir deux mots consécutifs pour toutes les structures à élaborer. Le premier mot doit renfermer le numéro de la page logique, le second le numéro de la page physique.

Si le numéro de la page logique vaut -1, la page physique correspondante sera alors désactivée. Dans ce cas, elle devient interdite à la lecture et à l'écriture.

Interruption 67h, Fonction 50h, Sous-fonction 01h LIM/EMS (à partir de 4.0)
Construire des pages logiques sur des pages physiques en fonction de leurs adresses

Tout comme la sous-fonction 00h, celle-ci permet de construire des pages logiques multipes sur un nombre quelconque de pages physiques à l'aide d'un seul appel de la fonction. Ici, les pages physiques ne sont pas adressées avec leur numéros, mais avec leur adresse de segment.

Entrée : AH = 50h
AL = 01h
CX = Nombre de pages logiques à construire
DX = Handle sur lequel les pages sont à allouer
DS:SI = Pointeur sur le buffer contenant les informations sur la structure

Sortie : AH = 0, Tout va bien, dans ce cas
AH > 0, Erreur, cf. Fonction 40h

Remarques :

- Comme toutes les autres fonctions de l'EMM, cette fonction ne doit être appelée qu'après s'être assuré qu'une mémoire EMM de la version 4.0 ou supérieure est effectivement installée. Voyez aussi le chapitre 12 à ce propos.
- Le buffer spécifié doit contenir deux mots consécutifs pour toutes les structures à élaborer. Le premier mot doit renfermer le numéro de la page logique, le second l'adresse de segment de la page physique.
- Si le numéro de la page logique vaut -1, la page physique correspondante sera alors désactivée. Dans ce cas, elle devient interdite à la lecture et à l'écriture.
- Les adresses de segment des pages physiques disponibles peuvent être obtenues à l'aide de la sous-fonction 00h de la fonction 58h.

Interruption 67h, Fonction 51h LIM/EMS (à partir de 4.0)
Modifier le nombre de pages logiques allouées

Le nombre de pages logiques appartenant à un handle alloué préalablement à l'aide de la fonction 43h peut être augmenté ou diminué librement avec cette fonction.

Entrée : AH = 48h
DX = Handle sur lequel les pages ont été allouées
BX = Nouveau nombre des pages rendues disponibles par ce handle

Sortie : AH = 0, Tout va bien, dans ce cas
BX = Nombre de pages logiques dont dispose maintenant le handle
AH > 0, Erreur, cf. Fonction 40h

Remarques :

- Comme toutes les autres fonctions de l'EMM, cette fonction ne doit être appelée qu'après s'être assuré qu'une mémoire EMM de la version 4.0 ou supérieure est effectivement installée. Voyez aussi le chapitre 12 à ce propos.
- La réduction du nombre de pages affectées est toujours possible, mais l'augmentation dépend de la taille de la mémoire EMS non encore allouée. Par conséquent, après avoir appelé la fonction, n'oubliez jamais de vérifier l'état d'erreur dans le registre AH.
- Si vous réduisez le nombre de pages affectées à l'aide de cette fonction, un nombre correspondant de pages est soustrait de l'extrémité supérieure des pages actuellement allouées. Leur contenu est perdu définitivement.

- Après appel de cette fonction, le handle reste toujours valable même si le nombre de pages atteint 0.

Interruption 67h, Fonction 52h, Sous-fonction 00h LIM/EMS (à partir de 4.0)
Déterminer l'attribut d'un handle

Cette fonction permet de constater si le contenu des pages EMS affectées à un handle reste conservé à travers un démarrage à chaud de l'ordinateur. Dans ce cas, elles sont désignées comme des pages "non volatiles".

Entrée : AH = 52h
AL = 00h
DX = Handle à consulter

Sortie : AH = 0, Tout va bien, dans ce cas
AL = Attribut des pages
0 : le contenu des pages est perdu (volatile)
1 : le contenu des pages reste conservé (non volatile)
AH > 0, Erreur, cf. Fonction 40h

Remarques :
- Comme toutes les autres fonctions de l'EMM, cette fonction ne doit être appelée qu'après s'être assuré qu'une mémoire EMM de la version 4.0 ou supérieure est effectivement installée. Voyez aussi le chapitre 12 à ce propos.

Interruption 67h, Fonction 52h, Sous-fonction 01h LIM/EMS (à partir de 4.0)
Fixer l'attribut d'un handle

Si une carte EMS est capable de protéger les pages EMS contre un démarrage à chaud de l'ordinateur, l'attribut correspondant peut alors être défini à l'aide de cette fonction.

Entrée : AH = 52h
AL = 01h
BL = Attribut pour les pages du handle
0 : le contenu des pages est perdu (volatile)
1 : le contenu des pages reste conservé (non volatile)
DX = Handle à consulter

Sortie : AH = 0, Tout va bien
AH > 0, Erreur, cf. Fonction 40h

Remarques :
- Comme toutes les autres fonctions de l'EMM, cette fonction ne doit être appelée qu'après s'être assuré qu'une mémoire EMM de la

version 4.0 ou supérieure est effectivement installée. Voyez aussi le chapitre 12 à ce propos.

- Attention ! Très peu de cartes EMS sont capables de protéger les pages EMS contre l'écriture lors d'un démarrage à chaud de l'ordinateur. En ce qui concerne les émulateurs EMS, cette possibilité est généralement à exclure. N'oubliez donc pas de vérifier au préalable si les pages EMS non volatiles disposent de cette technique au moyen de la sous-fonction 02h.

Interruption 67h, Fonction 52h, Sous-fonction 02h LIM/EMS (à partir de 4.0)
Vérifier la faculté des pages EMS non volatiles

Cette fonction permet de vérifier si une carte EMS offre la possibilité de protéger les pages EMS lors d'un démarrage à chaud de l'ordinateur. Les fonctions 52h/00h et 52h/01h ne peuvent être utilisées que dans ce cas.

Entrée : AH = 52h
AL = 02h

Sortie : AH = 0, Tout va bien, dans ce cas
AL = 0 : Les pages non volatiles ne sont pas reconnues
1 : Les pages volatiles sont reconnues
AH > 0, Erreur, cf. Fonction 40h

Remarques :
- Comme toutes les autres fonctions de l'EMM, cette fonction ne doit être appelée qu'après s'être assuré qu'une mémoire EMM de la version 4.0 ou supérieure est effectivement installée. Voyez aussi le chapitre 12 à ce propos.

Interruption 67h, Fonction 53h, Sous-fonction 00h LIM/EMS (à partir de 4.0)
Déterminer le nom d'un handle

Le nom d'un handle est transmis dans un buffer du programme d'appel suite à l'appel de cette fonction.

Entrée : AH = 53h
AL = 00h
DX = Handle
ES:DI = Pointeur sur le buffer devant recevoir le nom

Sortie : AH = 0, Tout va bien
AH > 0, Erreur, cf. Fonction 40h

Remarques :

- Comme toutes les autres fonctions de l'EMM, cette fonction ne doit être appelée qu'après s'être assuré qu'une mémoire EMM de la version 4.0 ou supérieure est effectivement installée. Voyez aussi le chapitre 12 à ce propos.
- La longueur du nom d'un handle est toujours de 8 caractères si bien que le buffer spécifié doit contenir au moins 8 octets.
- Lorsqu'un handle est alloué avec la fonction 43h, le nom est d'abord vide, c'est-à-dire qu'il ne contient que les caractères ayant le code 0. A l'aide de la sous-fonction 01h de la fonction 53h, un handle peut toutefois obtenir un nom qui est transmis par un appel de la sous-fonction 00h.

Interruption 67h, Fonction 53h, Sous-fonction 01h LIM/EMS (à partir de 4.0)
Définir le nom d'un handle

Cette sous-fonction affecte un nom à un handle préalablement alloué servant ensuite à l'appeler.

Entrée : AH = 53h
AL = 01h
DX = Handle
ES:DI = Pointeur sur le buffer contenant le nom du handle

Sortie : AH = 0, Tout va bien
AH > 0, Erreur, cf. Fonction 40h

Remarques :

- Comme toutes les autres fonctions de l'EMM, cette fonction ne doit être appelée qu'après s'être assuré qu'une mémoire EMM de la version 4.0 ou supérieure est effectivement installée. Voyez aussi le chapitre 12 à ce propos.
- La longueur du nom d'un handle est toujours de 8 caractères si bien que le buffer spécifié doit contenir au moins 8 octets.

Interruption 67h, Fonction 54h, Sous-fonction 00h LIM/EMS (à partir de 4.0)
Déterminer les noms de tous les handles

Les noms de tous les handles sont transmis dans un buffer du programme d'appel une fois cette fonction appelée.

Entrée : AH = 54h
AL = 00h
ES:DI = Pointeur sur le buffer recevant les noms des handles

Sortie : AH = 0, Tout va bien, dans ce cas
AL = Nombre de handles actifs
AH > 0, Erreur, cf. Fonction 40h

Remarques :
- Comme toutes les autres fonctions de l'EMM, cette fonction ne doit être appelée qu'après s'être assuré qu'une mémoire EMM de la version 4.0 ou supérieure est effectivement installée. Voyez aussi le chapitre 12 à ce propos.
- Le buffer doit disposer d'une place équivalente à 2550 octets parce qu'un maximum de 255 handles est géré et chaque entrée de handle nécessite 10 octets de ce buffer. Les deux premiers octets reçoivent le véritable handle, puis viennent les huit octets qui contiennent le nom du handle.

Interruption 67h, Fonction 54h, Sous-fonction 01h LIM/EMS (à partir de 4.0)
Déterminer un handle avec un nom précis

Cette fonction renvoie le handle dont le nom lui a été transmis.

Entrée : AH = 54h
AL = 01h
DS:SI = Pointeur sur le buffer contenant le nom du handle à rechercher

Sortie : AH = 0, Tout va bien, dans ce cas
DX = Le handle recherché
AH > 0, Erreur, cf. Fonction 40h

Remarques :
- Comme toutes les autres fonctions de l'EMM, cette fonction ne doit être appelée qu'après s'être assuré qu'une mémoire EMM de la version 4.0 ou supérieure est effectivement installée. Voyez aussi le chapitre 12 à ce propos.

Interruption 67h, Fonction 55h, Sous-fonction 00h LIM/EMS (à partir de 4.0)
Construire des pages et sauter vers le mémoire étendue

D'une part, cette fonction définit la structure entre les pages logiques et physiques. D'autre part, elle transfère l'exécution du programme vers l'une des pages construites à l'aide d'un FAR-Jump.

Entrée : AH = 55h
AL = 00h
DX = Handle

DS:SI = Pointeur sur le buffer contenant des informations complémentaires

Sortie : AH = 0, Tout va bien
AH > 0, Erreur, cf. Fonction 40h

Remarques :
- Comme toutes les autres fonctions de l'EMM, cette fonction ne doit être appelée qu'après s'être assuré qu'une mémoire EMM de la version 4.0 ou supérieure est effectivement installée. Voyez aussi le chapitre 12 à ce propos.
- Le buffer spécifié doit présenter la structure suivante :

Structure du tampon		
Adresse	**Contenu**	**Type**
+00h	Pointeur sur l'adresse à laquelle le programme doit continuer	1 PTR
+05h	Nombre des pages mémoire à afficher avant le saut	1 BYTE
+06h	Pointeur sur la liste d'affichage	1 PTR
Longueur : 9 octets		

La structure de la liste d'affichage est identique à celle de cette liste dans la structure de la fonction 50h, sous-fonction 00h. A chaque appel, nous obtenons donc le numéro d'une page logique et celui de la page physique correspondante.

Interruption 67h, Fonction 55h, Sous-fonction 01h LIM/EMS (à partir de 4.0)
Afficher pages et sauter en mémoire étendue

Cette sous-fonction est similaire à la 00h, mais la liste d'affichage contient les adresses de segment au lieu des numéros des pages physiques.

Veuillez vous reporter à la description de la sous-fonction 00h.

Interruption 67h, Fonction 56h, Sous-fonction 00h LIM/EMS (à partir de 4.0)
Afficher pages et appeler routine en mémoire étendue

Cette fonction définit la structure entre les pages logiques et physiques et elle provoque la poursuite de l'exécution du programme avec une ro utine ens mémoire étendue par un appel FAR. A la fin de cette routine, une nouvelle structure est construite et on revient à l'appelant de la fonction d'interruption.

Entrée : AH = 55h
AL = 00h
DX = identificateur
DS:SI = pointeur sur tampon avec informations complémentaires

Sortie : AH = 0, ok
AH > 0, erreur, voir fonction 40h

L'appel de cette fonction doit être précédé de la vérification présence d'un EMM ("Expanded Memory Manager") de la version 4.0. Voir également le chapitre 12 à ce sujet.

Le tampon indiqué doit avoir la structure suivante :

Structure du tampon		
Adresse	Contenu	Type
+00h	Pointeur sur le sous-programme par lequel le programme doit continuer	1 PTR
+05h	Nombre de pages à afficher avant l'appel du sous-programme	1 BYTE
+06h	Pointeur sur la liste d'affichage pour l'affichage avant l'affichage avant l'appel du sous-programme	1 PTR
+0Ah	Nombre de pages à afficher avant l'appel du sous-programme	1 BYTE
+0Bh	Pointeur sur la liste d'affichage après l'appel du sous-programme	1 PTR
Longueur : 14 octets		

La structure de la liste d'affichage est identique à celle de cette liste dans l'appel de la fonction 50h, sous-fonction 00h : elle renvoie le numéro d'une page logique et d'une page physique.

L'exécution de cette fonction exige un certain espace libre de la pile. La taille requise est renvoyée par la sous-fonction 02h de cette fonction.

Interruption 67h, fonction 56h, sous-fonction 01h LIM/EMS (à partir de 4.0)
Afficher des pages et appeler une routine de la mémoire étendue

Cette sous-fonction est similaire à la sous-fonction 00h mais elle renvoie les adresses de ses segments et non les numéros des pages physiques.

Veuillez vous reporter à la description de la sous-fonction 00h pour informations complémentaires.

Interruption 67h, fonction 56h, sous-fonction 02h LIM/EMS (à partir de 4.0)

Cette fonction renvoie l'espace minimal requis sur la pile pour permettre l'exécution des sous-fonctions 00h et 01h.

Entrée : AH = 56h
AL = 02h

Sortie : AH = 0, ok, dans ce cas :
BX = espace pile demandé
AH > 0, erreur, voir fonction 40h

L'appel de cette fonction doit être précédé de la vérification de la présence d'un EMM ("Expanded Memory Manager") de la version 4.0. Voir également le chapitre 12 à ce sujet.

La valeur renvoyée correspond exclusivement aux besoins de l'EMM. L'espace de la pile requis pour la routine à appeler dans la mémoire étendue vient s'y ajouter.

Interruption 67h, fonction 57h, sous-fonction 00h LIM/EMS (à partir de 4.0)
Copier un bloc de mémoire

Cette fonction permet de copier tout bloc de mémoire entre la mémoire paginée et la mémoire étendue ou à l'intérieur de chacune de ces mémoires.

Entrée : AH = 57h
AL = 00h
DS:SI = Pointeur sur le tampon contenant les informations sur les blocs à copier

Sortie : AH = 0, ok
AH > 0, erreur, voir fonction 40h

L'appel de cette fonction doit être précédé de la vérification de la présence d'un EMM ("Expanded Memory Manager") de la version 4.0. Voir également le chapitre 12 à ce sujet.

Le tampon contenant les informations sur les blocs à copier doit avoir le format suivant :

Structure du tampon		
Adresse	**Contenu**	**Type**
+00h	Taille (en octets) du bloc à copier	1 DWORD
+04h	Type du bloc source (0 = mémoire conventionnelle, 1 = EMS)	1 BYTE
+05h	Identificateur du bloc source (EMS seulement)	1 WORD
+07h	Décalage du début du bloc source	1 WORD
+09h	Adresse de segment du bloc source (mémoire conventionnelle) ou numéro de la page physique (EMS)	1 WORD
+0Bh	Type du bloc cible (0 = mémoire conventionnelle, 1 = EMS)	1 BYTE
+0Ch	Identificateur du bloc cible (EMS seulement)	1 WORD
+0Eh	Décalage du début du bloc source	1 WORD
+10h	Adresse segment du bloc cible (mémoire conventionnelle) ou numéro de la page physique (EMS)	1 WORD
Longueur : 18 octets		

Interruption 67h, fonction 57h, sous-fonction 01h LIM/EMS (à partir de 4.0)
Echanger des blocs de mémoire

Cette fonction équivaut à la sous-fonction 00h, mais elle échange le contenu des blocs de mémoire au lieu de copier le contenu de l'un dans l'autre.

Entrée : AH = 57h
AL = 01h
DS:SI = Pointeur sur le tampon contenant les informations sur les blocs à copier

Sortie : AH = 0, ok
AH > 0, erreur, voir fonction 40h

L'appel de cette fonction doit être précédé de la vérification de la présence d'un EMM ("Expanded Memory Manager") de la version 4.0. Voir également le chapitre 12 à ce sujet.

Le tampon contenant les informations sur les blocs à copier doit avoir le format décrit pour la sous-fonction 00h.

Cette fonction peut échanger au maximum 1 Mo de mémoire par appel.

Si des pages d'EMS sont concernées par le déplacement et si le bloc défini sort de la page EMS définie, les pages EMS suivantes sont également déplacées.

Les blocs source et cible ne doivent pas se chevaucher !

Interruption 67h, fonction 58h, sous-fonction 00h LIM/EMS (à partir de 4.0)
Renvoie les adresses des pages EMS physiques et leurs numéros

Cette fonction retourne, dans un tampon établi par l'appelant, les adresses des segments de toutes les pages physiques EMS et de leurs numéros.

Entrée : AH = 58h
AL = 00h
DS:SI = Pointeur sur le tampon destiné à recevoir les informations

Sortie : AH = 0, ok, dans ce cas :
CX = nombre des entrées retournées dans le tampon
AH > 0, erreur, voir fonction 40h

L'appel de cette fonction doit être précédé de la vérification de la présence d'un EMM ("Expanded Memory Manager") de la version 4.0. Voir également le chapitre 12 à ce sujet.

Vous pouvez demander à la sous-fonction 00h la taille requise pour le tampon.

Dans le tampon indiqué, l'EMM contient les informations demandées sous forme d'entrées de 2 mots chacune. Le premier contient l'adresse du segment d'une page EMS, le deuxième contient son numéro.

Interruption 67h, fonction 58h, sous-fonction 01h LIM/EMS (à partir de 4.0)
Renvoie le nombre de pages EMS

Cette fonction retourne le nombre des pages EMS physiques, ce qui permet de calculer la taille du tampon qui sera nécessaire pour l'appel de la sous-fonction 00h de cette fonction.

Entrée : AH = 58h
AL = 01h

Sortie : AH = 0, ok, dans ce cas :
CX = nombre des entrées retournées dans le tampon
AH > 0, erreur, voir fonction 40h

L'appel de cette fonction doit être précédé de la vérification de la présence d'un EMM ("Expanded Memory Manager") de la version 4.0. Voir également le chapitre 12 à ce sujet.

La taille requise pour le tampon est obtenue en multipliant le nombre des entrées (registre CX) par quatre.

Interruption 67h, fonction 59h, sous-fonction 00h LIM/EMS (à partir de 4.0)
Renvoie des informations sur le matériel EMS

Cette fonction permet de recueillir certaines informations sur le matériel EMS, présentant pour la plupart de l'intérêt pour un système d'exploitation intégrant la mémoire EMS à sa gestion de la mémoire.

Entrée : AH = 59h
AL = 00h

Sortie : AH = 0, ok, dans ce cas :
CX = taille (en octets) du tableau des pages
AH > 0, erreur, voir fonction 40h

L'appel de cette fonction doit être précédé de la vérification de la présence d'un EMM ("Expanded Memory Manager") de la version 4.0. Voir également le chapitre 12 à ce sujet.

La taille du tampon doit être au moins égale à 10 octets. Après l'appel de la fonction, le tampon contient les informations suivantes :

Structure du tampon

Adresse	Contenu	Type
+00h	Taille des pages EMS non standardisées en paragraphes	1 WORD
+02h	Nombre des jeux alternatifs de registres EMS	1 WORD
+04h	Taille de la table d'affichage et de sauvegarde en octets	1 WORD
+06h	Nombre de jeux de registres pour le transfert en DMA	1 WORD
+08h	Mode d'opération DMA	1 WORD

Longueur : 10 octets

Les modes d'opérations DMA sont au nombre de deux :

0 : le DMA est également utilisable avec des jeux alternatifs de registres
1 : Un seul jeu de registres DMA disponible

Interruption 67h, fonction 59h, sous-fonction 01h LIM/EMS (à partir de 4.0)
Renvoie le nombre de pages EMS non standardisées

Cette fonction retourne le nombre total de pages EMS dont la taille diffère des 16 Ko standard et le nombre des pages encore libres parmi celles-ci.

Entrée : AH = 59h
AL = 01h

Sortie : AH = 0, ok, dans ce cas :
DX = nombre total des pages non standardisées
BX = nombre des pages non encore allouées
AH > 0, erreur, voir fonction 40h

L'appel de cette fonction doit être précédé de la vérification de la présence d'un EMM ("Expanded Memory Manager") de la version 4.0. Voir également le chapitre 12 à ce sujet.

Cette forme particulière de pages EMS n'est pas gérée par toutes les cartes EMS. Ceci se traduit par la valeur 0 dans le registre DX.

Interruption 67h, fonction 5Ah, sous-fonction 00h LIM/EMS (à partir de 4.0)
Alloue des pages EMS

Cette fonction alloue des pages EMS et ressemble donc à la 43h. Cependant, contrairement à cette dernière, la fonction 5Ah permet l'allocation de 0 page EMS sans que cela ne se traduise par une erreur.

Entrée : AH = 5Ah
Al = 0h
BX = nombre des pages EMS allouées

Sortie : AH = 0, ok. Dans ce cas :
DX = identificateur permettant l'accès aux pages allouées
AH > 0, erreur, voir fonction 40h

L'appel de cette fonction doit être précédé de la vérification de la présence d'un EMM ("Expanded Memory Manager") de la version 4.0. Voir également le chapitre 12 à ce sujet.

Vous pouvez allouer 0 page à l'aide de cette fonction sans causer d'erreur, et ainsi obtenir un identificateur.

Interruption 67h, fonction 5Ah, sous-fonction 01h LIM/EMS (à partir de 4.0)
Alloue des pages EMS non standardisées

Cette fonction permet d'allouer des pages EMS dont la taille diffère des 16 Ko standard.

Entrée : AH = 5Ah
AL = 01h

Sortie : AH = 0, ok, dans ce cas :
DX = identificateur permettant d'accéder aux pages allouées
AH > 0, erreur, voir fonction 40h

L'appel de cette fonction doit être précédé de la vérification de la présence d'un EMM ("Expanded Memory Manager") de la version 4.0. Voir également le chapitre 12 à ce sujet.

Cette forme particulière de pages EMS n'est pas gérée par toutes les cartes EMS. Vérifiez soigneusement l'état de la fonction dans le registre AH après l'appel.

Cette fonction permet d'allouer zéro page pour obtenir simplement un identificateur.

Interruption 67h, fonction 5Bh LIM/EMS (à partir de 4.0)
Utile dans les transferts DMA

Cette fonction possède neuf sous-fonctions que nous ne décrirons pas ici car leur intérêt est strictement limité au développement de systèmes d'exploitation, et elles ne sont en outre gérées que par quelques cartes et pilotes EMS. En d'autres termes, ces sous-fonctions ne sont pas utilisées dans le cadre du développement normal sur PC.

Interruption 67h, fonction 5Ch LIM/EMS (à partir de 4.0)
Prépare l'EMS à un redémarrage à chaud

Entrée : AH = 5Ch

Sortie : AH = 0, ok
AH > 0, erreur, voir fonction 40h

L'appel de cette fonction doit être précédé de la vérification de la présence d'un EMM ("Expanded Memory Manager") de la version 4.0. Voir également le chapitre 12 à ce sujet.

Voir également les sous-fonctions de la fonctions 52h.

Interruption 67h, fonction 5Dh LIM/EMS (à partir de 4.0)
Utile pour inactiver le système d'exploitation

Cette fonction possède trois sous-fonctions que nous ne décrirons pas ici car leur intérêt est strictement limité au développement de systèmes d'exploitation, et elles ne sont en outre gérées que par quelques cartes et pilotes EMS. En d'autres termes, ces sous-fonctions ne sont pas utilisées dans le cadre du développement normal sur PC.

G. Description des fonctions de la spécification XMS

Le standard XMS (Extended Memory Specification) a été introduit par Microsoft en collaboration avec d'autres sociétés afin de mieux coordonner l'accès à la mémoire étendue au-delà de la limite des 1 Mo.

Contrairement aux nombreuses interfaces de fonction, l'appel des fonctions du standard XMS ne s'effectue pas à travers une interruption particulière mais à l'aide d'une instruction FAR CALL. Dans ce cas, il faut d'abord transmettre l'adresse d'un gestionnaire XMS. Reportez-vous au chapitre XXX pour obtenir de plus amples informations à ce sujet.

Fonction 00h XMS
Déterminer le numéro de version XMS

Cette fonction retourne le numéro de version du driver XMS et son numéro de révision interne.

Entrée : AH = 00h

Sortie : AX = Numéro de version XMS
BX = Numéro de révision interne
DX = Etat de la HMA
0 : Accès à HMA impossible
1 : HMA disponible

Remarques :
- Comme toutes les fonctions du driver XMS, celle-ci ne doit également être appelée qu'après avoir vérifié l'existence d'un tel driver à l'aide de l'interruption 2Fh et obtenu le point d'entrée du driver.

Fonction 01h XMS
Réserver une HMA (High Memory Area ou zone de mémoire haute)

Cette fonction permet à un programme d'allouer totalement ou partiellement la HMA pour ses besoins personnels.

Entrée : AH = 01h
DX = Place mémoire demandée en octets, FFFFh pour une application normale ayant un besoin réel en programmes TSR

Sortie : AX = 0001h Fonction exécutée
0000h Erreur, dans ce cas
BL = Numéro d'erreur
80h Fonction spécifiée non connue
81h VDISK-RAMDISK découvert
82h Erreur sur l'adresse de ligne A20
8Eh Erreur de driver générale
8Fh Erreur impossible à éliminer
90h HMA inexistante
91h HMA déjà occupée
92h Taille de mémoire spécifiée en DX inférieure à la valeur du paramètre /HmaMin=, donc trop petite
93h HMA non allouée
94h La ligne est encore en fonction
A0h Plus de mémoire étendue disponible
A1h Tous les handles XMS occupés
A2h Handle incorrect
A3h Handle source irrecevable
A4h Offset source irrecevable
A5h Handle cible incorrect
A6h Offset cible incorrect
A7h Longueur incorrecte pour la fonction Move
A8h Superposition interdite avec la fonction Move
A9h Erreur de parité
AAh UMB non verrouillé
ABh UMB encore verrouillé
ACh Débordement du compteur de verrouillage UMB
ADh UMB ne peut pas être verrouillé
B0h UMB plus petit disponible
B1h Plus de UMB disponible
B2h Adresse de segment de l'UMB incorrecte

Remarques :

- Comme toutes les fonctions du driver XMS, celle-ci ne doit également être appelée qu'après avoir vérifié l'existence d'un tel driver à l'aide de l'interruption 2Fh et obtenu le point d'entrée du driver.
- Après un appel réussi de cette fonction, l'accès à la HMA ne peut s'effectuer que si l'adresse de ligne A20 a été correctement activée. Cela est possible grâce aux fonctions 03h et 05h.
- Une erreur peut survenir dans le déroulement de la fonction si un programme TSR ne dispose pas de la totalité de la HMA et si l'insuffisance de la mémoire se situe au-dessous de la valeur spécifiée dans le paramètre /HMAMIN lors de l'installation du driver. Cela empêche d'octroyer un droit d'accès exclusif à la HMA à un programme TSR ne nécessitant que peu de Ko et de laisser ainsi la place à un autre qui peut avoir besoin d'une place mémoire plus importante.

- A la fin d'un programme, si la HMA reste toujours en sa possession, elle doit être libérée avec la fonction 02h avant de terminer le programme.

Fonction 02h XMS
Libérer la HMA

Un programme occupant la HMA à l'aide de la fonction 01h peut libérer cette zone de mémoire avec la fonction 02h.

Entrée : AH = 02h

Sortie : AX = 0001h : Fonction exécutée
0000h : Erreur, dans ce cas
BL = Numéro d'erreur (cf. Fonction 01h)

Remarques :
- Comme toutes les fonctions du driver XMS, celle-ci ne doit également être appelée qu'après avoir vérifié l'existence d'un tel driver à l'aide de l'interruption 2Fh et obtenu le point d'entrée du driver.
- Sans appeler cette fonction, une HMA préalablement allouée reste réservée lorsqu'un programme est terminé. Elle ne peut donc pas être attribuée au programme suivant.
- Le contenu de la HMA est détruit définitivement à la suite d'un appel de cette fonction.

Fonction 03h XMA
Activation globale de l'adresse de ligne A20

Cette fonction permet d'activer l'adresse A20 pour qu'un accès direct à la HMA puisse également avoir lieu à partir du mode réel du processeur.

Entrée : AH = 03h

Sortie : AX = 0001h : Fonction exécutée
0000h : Erreur, dans ce cas
BL = Numéro d'erreur (cf. Fonction 01h)

Remarques :
- Comme toutes les fonctions du driver XMS, celle-ci ne doit également être appelée qu'après avoir vérifié l'existence d'un tel driver à l'aide de l'interruption 2Fh et obtenu le point d'entrée du driver.

- Cette fonction doit être appelée avant ou après la demande portant sur la HMA pour s'assurer qu'il est également possible d'accéder à la HMA à partir du mode réel via l'adresse de segment FFFFh.
- Avant de terminer un programme, la ligne d'adresse A20 doit être clôturée avec la fonction 04h pour éviter tout risque de débordement de segment dans le programme suivant.
- L'activation de la ligne d'adresse A20 constitue un processus relativement long selon le système informatique dont vous disposez. Vous devez donc éviter d'appeler trop souvent cette fonction.

Fonction 04h XMS
Fermeture globale de la ligne d'adresse A20

Contrairement à la fonction 03h, cette fonction a pour rôle de refermer la ligne d'adresse A20. Il devient alors impossible d'accéder directement à la HMA à partir du mode réel.

Entrée : AH = 04h

Sortie : AX = 0001h : Fonction exécutée
0000h : Erreur, dans ce cas
BL = Numéro d'erreur (cf. Fonction 01h)

Remarques :
- Comme toutes les fonctions du driver XMS, celle-ci ne doit également être appelée qu'après avoir vérifié l'existence d'un tel driver à l'aide de l'interruption 2Fh et obtenu le point d'entrée du driver.
- Selon le système informatique, le verrouillage de la ligne d'adresse A20 constitue un processus relativement long. Vous devez donc éviter d'appeler trop souvent cette fonction.

Fonction 05h XMS
Libération locale de la ligne d'adresse A20

Contrairement à la libération globale réalisée avec la fonction 03h, ici la libération ne s'effectue que localement. Un compteur comptabilise l'appel de cette fonction et de la fonction 06h. La mise en service de la ligne ne s'effectue que si elle n'a pas été déjà activée par un appel antérieur.

Entrée : AH = 05h

Sortie : AX = 0001h : Fonction exécutée
0000h : Erreur, dans ce cas
BL = Numéro d'erreur (cf. Fonction 01h)

Remarques : ■ Voir Fonction 03h.

Fonction 06h XMS
Verrouillage local de la ligne d'adresse A20

Tout comme la fonction 05h, celle-ci ne prend effet que si l'adresse n'a pas été encore verrouillée à l'aide d'un appel antérieur.

Entrée : AH = 06

Sortie : AX = 0001h : Fonction exécutée
0000h : Erreur, dans ce cas
BL = Numéro d'erreur (cf. Fonction 01h)

Remarques :
- Les appels des fonctions 05h et 06h sont exprimés sur le plan interne à l'aide d'un compteur qui augmente à chaque appel de la fonction 05h et diminue à chaque appel de la fonction 06h. L'adresse n'est réellement verrouillée que lorsque le compteur atteint la valeur 0. Les appels des deux fonctions doivent donc être équivalents.
- Reportez-vous à la fonction 05h pour de plus amples informations.

Fonction 07h XMS
Obtenir l'état de la ligne d'adresse A20

Cette fonction permet de constater si la ligne d'adresse A20 est déjà en service, si un appel des fonctions 03h ou 05h est en cours.

Entrée : AH = 07h

Sortie : AX = 0001h : Ligne d'adresse A20 libre
0000h : Adresse A20 verrouillée

Remarques :
- Comme toutes les fonctions du driver XMS, celle-ci ne doit également être appelée qu'après avoir vérifié l'existence d'un tel driver à l'aide de l'interruption 2Fh et obtenu le point d'entrée du driver.
- La demande formulée pour la ligne d'adresse A20 s'effectue indépendamment de l'électronique pour que l'appel de cette fonction ne dure pas aussi longtemps que par exemple le verrouillage ou la libération de la ligne d'adresse A20.

Fonction 08h XMS
Déterminer la taille de la mémoire étendue disponible

Cette fonction retourne le volume total disponible dans la mémoire étendue et la taille du plus grand bloc encore libre.

Entrée : AH = 08h

Sortie : AX = Taille du plus grand bloc de mémoire étendue (EMB) en octet
DX = Volume total disponible dans la mémoire étendue en Ko

Remarques :
- Comme toutes les fonctions du driver XMS, celle-ci ne doit également être appelée qu'après avoir vérifié l'existence d'un tel driver à l'aide de l'interruption 2Fh et obtenu le point d'entrée du driver.
- Attention ! La HMA n'est pas prise en compte lors de l'évaluation des deux indications même si elle n'a pas été encore allouée. Par conséquent, les indications s'élèvent à 64 Ko parce que l'allocation de la mémoire étendue commence toujours après la HMA. Reportez-vous au chapitre 12.

Fonction 09h XMS
Allouer un bloc de mémoire étendue (EMB)

Un programme peut se servir de cette fonction pour réserver un bloc de mémoire étendue.

Entrée : AH = 09h
DX = Taille de la zone sollicitée en Ko

Sortie : AX = 0001h : Fonction exécutée
0000h : Erreur, dans ce cas
BL = Numéro d'erreur (cf. Fonction 01h)
DX = Handle sur le prochain accès à l'EMB si un tel bloc peut être alloué

Remarques :
- Comme toutes les fonctions du driver XMS, celle-ci ne doit également être appelée qu'après avoir vérifié l'existence d'un tel driver à l'aide de l'interruption 2Fh et obtenu le point d'entrée du driver.
- Pour tous les appels ultérieurs, le handle retourné sert à accéder à l'EMB et doit être sauvegardé dans le programme. S'il est perdu, il faut réinitialiser l'ordinateur ou effectuer un démarrage à chaud pour libérer l'EMB concerné.

- Le XMS ne peut répondre aux besoins d'un EMB que s'il existe un bloc de mémoire suffisamment grand et un handle. Mais comme le nombre de handles est limité (32 en règle générale), il faut toujours essayer d'allouer des EMB suffisamment grands pour éviter d'appeler trop souvent la fonction et d'utiliser trop de handles.
- Avant de terminer un programme, il faut libérer l'EMB avec la fonction 0Ah pour qu'il ne soit pas perdu lors de sa transmission à d'autres applications.

Fonction 0Ah XMS
Libérer un bloc de mémoire étendue (EMB)

Un bloc de mémoire alloué préalablement par la fonction 09h peut être libéré à l'aide de cette fonction.

Entrée : AH = 0Ah
DX = Handle

Sortie : AX = 0001h : Fonction exécutée
0000h : Erreur, dans ce cas
BL = Numéro d'erreur (cf. Fonction 01h)

Remarques :
- Comme toutes les fonctions du driver XMS, celle-ci ne doit également être appelée qu'après avoir vérifié l'existence d'un tel driver à l'aide de l'interruption 2Fh et obtenu le point d'entrée du driver.
- Le contenu de l'EMB est définitivement perdu à la suite d'un appel de cette fonction et le handle devient non valable.

Fonction 0Bh XMS
Copier la mémoire

Cette fonction sert à transférer la mémoire entre la RAM conventionnelle et la mémoire étendue ou à copier les blocs de mémoire à l'intérieur de la mémoire conventionnelle ou la mémoire étendue.

Entrée : AH = 0Bh
DS:SI = Pointeur sur la structure suivante déterminant la zone de mémoire à copier et la destination.

Structure de l'EMB à déplacer		
Adr.	Contenu	Type
00h	Taille du bloc en octets	1DWORD
04h	Handle du bloc source	1WORD
06h	Offset du bloc source à partir duquel la copie doit avoir lieu	1DWORD
0Ah	Handle du bloc cible	1WORD
0Ch	Offset du bloc cible à partir duquel la copie doit avoir lieu	1DWORD
Taille : 16 octets		

Sortie : AX = 0001h : Fonction exécutée
0000h : Erreur, dans ce cas
BL = Numéro d'erreur (cf. Fonction 01h)

Remarques :

- Comme toutes les fonctions du driver XMS, celle-ci ne doit également être appelée qu'après avoir vérifié l'existence d'un tel driver à l'aide de l'interruption 2Fh et obtenu le point d'entrée du driver.
- Si un EMB doit être appelé, il faut spécifier également le handle retourné dans la fonction 09h lors de l'allocation du EMB. L'adresse d'offset représente alors l'offset relativement au début du bloc.
- Au contraire, si une zone doit être appelée dans la RAM conventionnelle, il faut spécifier la valeur 0 en guise de handle et les adresses de segment et d'offset du bloc de départ selon le format habituel (d'abord l'adresse d'offset, puis l'adresse de segment) en guise d'offset.
- En ce qui concerne les handles spécifiés, il ne faut pas qu'ils correspondent aux handles dont les blocs correspondants ont été verrouillés par la fonction 0Ch.
- Si les zones spécifiées se superposent, la zone source doit figurer avant la zone cible pour garantir une exécution dépourvue d'erreurs de la fonction.
- Pour assurer une duplication rapide de la zone source, il est souhaitable que les deux zones commencent par une adresse de mémoire paire dans un AT et par une adresse de mémoire divisible par quatre dans un 80386.

Fonction 0Ch XMS
Protège un bloc de mémoire étendue (EMB) contre un déplacement

Pour que des trous n'apparaissent pas dans la mémoire étendue après sa libération, la XMM se voit parfois contrainte de déplacer les divers EMB dans la mémoire. Cette fonction empêche de déplacer un EMB bien précis.

Entrée : AH = 0Ch
DX = Handle de l'EMB

Sortie : AX = 0001h : Fonction exécutée
0000h : Erreur, dans ce cas
BL = Numéro d'erreur (cf. Fonction 01h)
DX:BX = Adresse 32 bits linéaire de l'EMB dans la mémoire

Remarques :

- Comme toutes les fonctions du driver XMS, celle-ci ne doit également être appelée qu'après avoir vérifié l'existence d'un tel driver à l'aide de l'interruption 2Fh et obtenu le point d'entrée du driver.
- La fonction retourne l'adresse de l'EMB qui ne reste valable que tant que l'EMB n'est pas à nouveau verrouillé. Les blocs verrouillés doivent être libérés aussi vite que possible avec la fonction 0Dh pour que la gestion de la mémoire XMM ne subisse pas trop de conséquences.
- L'état de verrouillage des différents EMB est fourni distinctement dans un compteur qui augmente à chaque appel de cette fonction et diminue à chaque appel de la fonction 0Dh. Le compteur commence avec la valeur 0. Lorsque le compteur indique à nouveau la valeur 0, cela signifie que le EMB concerné est déverrouillé.

Fonction 0Dh XMS
Déverrouiller le bloc de mémoire étendue (EMB)

Un EMB préalablement verrouillé peut être déverrouillé à l'aide de cette fonction. Il pourra donc être déplacé dans la mémoire étendue.

Entrée : AH = 0Dh
DX = Handle de l'EMB

Sortie : AX = 0001h : Fonction exécutée
0000h : Erreur, dans ce cas
BL = Numéro d'erreur (cf. Fonction 01h)

Remarques :

- Comme toutes les fonctions du driver XMS, celle-ci ne doit également être appelée qu'après avoir vérifié l'existence d'un tel driver à l'aide de l'interruption 2Fh et obtenu le point d'entrée du driver.
- L'adresse de l'EMB retournée par l'appel de la fonction 0Ch devient caduque à la suite de cet appel.
- Reportez-vous à la fonction 0Ch à propos du "compteur de verrouillage".

Fonction 0Eh XMS
Lire les informations sur un EMB

Cette fonction permet de prendre connaissance des diverses informations concernant un EMB. Outre la taille du bloc, vous pouvez connaître son compteur de verrouillage ainsi que le nombre de handles encore disponibles.

Entrée : AH = 0Eh
DX = Handle de l'EMB

Sortie : AX = 0001h : Fonction exécutée, dans ce cas
BH = Compteur de verrouillage du bloc
BL = Nombre de handles EMB disponibles
DX = Taille de l'EMB en Ko
0000h : Erreur, dans ce cas
BL = Numéro d'erreur (cf. Fonction 01h)

Remarques :

- Comme toutes les fonctions du driver XMS, celle-ci ne doit également être appelée qu'après avoir vérifié l'existence d'un tel driver à l'aide de l'interruption 2Fh et obtenu le point d'entrée du driver.
- En ce qui concerne le compteur de verrouillage, il s'agit d'un compteur réglé individuellement pour chaque EMB. Il augmente à chaque appel de la fonction de verrouillage (0Ch) et diminue en conséquence à chaque appel de la fonction 0Dh. L'EMB n'est effectivement verrouillé que lorsque le compteur atteint la valeur 0. Une valeur différente de zéro indique le nombre d'appels de la fonction 0Dh à effectuer pour verrouiller à nouveau l'EMB.

Fonction 0Fh XMS
Agrandir ou diminuer le bloc de mémoire étendue (EMB) alloué

Cette fonction permet de modifier la taille d'un EMB.

Entrée : AH = 0Fh
BX = Nouvelle taille de l'EMB en Ko
DX = Handle de l'EMB

Sortie : AX = 0001h : Fonction exécutée
0000h : Erreur, dans ce cas
BL = Numéro d'erreur (cf. Fonction 01h)

Remarques :
- Comme toutes les fonctions du driver XMS, celle-ci ne doit également être appelée qu'après avoir vérifié l'existence d'un tel driver à l'aide de l'interruption 2Fh et obtenu le point d'entrée du driver.
- L'EMB spécifié doit être verrouillé pour pouvoir être agrandi ou diminué.
- Le contenu du bloc se perd définitivement si la taille de l'EMB est réduite.

Fonction 10h XMS
Allouer un bloc de mémoire supérieure (UMB)

Compte tenu du matériel installé, certains drivers XMS permettent d'utiliser la mémoire conventionnelle située entre la limite des 640 Ko et le début de la mémoire étendue. Un tel bloc de mémoire peut être alloué à l'aide de cette fonction.

Entrée : AH = 10h
DX = Taille du bloc de mémoire sollicité en paragraphes

Sortie : AX = 0001h : Fonction exécutée, dans ce cas
BX = Adresse de segment de l'UMB
0000h : Erreur, dans ce cas
BL = Numéro d'erreur (cf. Fonction 01h)
DX = Taille maximale d'un UMB à allouer en paragraphes

Remarques :
- Comme toutes les fonctions du driver XMS, celle-ci ne doit également être appelée qu'après avoir vérifié l'existence d'un tel driver à l'aide de l'interruption 2Fh et obtenu le point d'entrée du driver.
- Attention ! Cette fonction dépend considérablement du matériel et n'est pas disponible sur tous les drivers XMS. Après son appel, il est donc indispensable de vérifier l'état d'erreur pour s'assurer qu'un UMB peut effectivement être alloué.
- L'adresse de segment retournée par la fonction permet également d'accéder directement à l'UMB en mode réel. Dans ce cas, l'adresse

d'offset en octets résultant du calcul de la taille du bloc n'est pas ignorée.

- Cette fonction peut également être utilisée pour obtenir le plus grand bloc UMB disponible en spécifiant tout simplement une valeur non réelle pour la taille du bloc à allouer (exemple, FFFFh). La fonction n'alloue en aucun cas un UMB mais retourne la taille du plus grand bloc UMB disponible.

Fonction 11h XMS
Libérer le bloc de mémoire supérieure (UMB) alloué

Un UMB préalablement alloué par la fonction 10h est libéré par l'appel de cette fonction et devient ainsi disponible pour d'autres programmes.

Entrée : AH = 11h
DX = Adresse de segment de l'UMB

Sortie : AX = 0001h : Fonction exécutée, dans ce cas
0000h : Erreur, dans ce cas
BL = Numéro d'erreur
(cf. Fonction 01h)

Remarques :

- Comme toutes les fonctions du driver XMS, celle-ci ne doit également être appelée qu'après avoir vérifié l'existence d'un tel driver à l'aide de l'interruption 2Fh et obtenu le point d'entrée du driver.
- Le contenu de l'UMB se perd définitivement après l'appel de cette fonction. Il devient alors impossible d'accéder à la mémoire au moyen de l'adresse de segment du bloc.
- Avant de terminer un programme, il convient de libérer tous les blocs UMB alloués à l'aide de cette fonction, sinon ils ne peuvent pas être accédés par les applications lancées ultérieurement.

H. Les fonctions du driver de souris

Dès le départ, Microsoft a équipé sa souris d'une interface logicielle implémentée sous la forme du driver de périphérique MOUSE.SYS ou du programme MOUSE.COM. Cette interface est devenue par la suite un standard adopté par tous les constructeurs de souris afin de garantir une compatibilité totale avec la souris Microsoft. Les diverses fonctions de l'interface de souris sont appelées alors par l'intermédiaire de l'interruption 33h où une valeur 16 bits tient lieu de numéro de fonction. Celle-ci doit être transmise dans le registre AX.

La version 8.0 du driver de souris reconnaît en tout 53 fonctions variées avec des numéros allant de 0000h à 00034h. Toutefois, les fonctions 11h et 12h ainsi que 0002Eh sont considérées comme non documentées et ne s'utilisent qu'à l'intérieur du driver de souris.

Le gestionnaire de souris a entraîné des révisions et augmentations dans plus d'un programme de la société Microsoft. D'innombrables variantes ont apparu entre les versions 1.0 et 8.0. Les versions dont le numéro est inférieur à 6.26 n'entrent plus en ligne de compte parce qu'elles n'utilisent pas les cartes vidéo modernes.

Malheureusement, Microsoft ne fournit la date de mise en service des diverses fonctions du gestionnaire de souris à partir de la fonction 25h seulement, celle-ci étant introduite avec la version 6.26 citée précédemment. De nombreuses fonctions existent déjà depuis la version 1.0, mais peu d'entre elles ont été insérées au cours du développement intervenu entre les versions 1.0 et 6.26. Etant donné que nous ne disposons pas d'informations précises, nous supposons ici que ces fonctions sont disponibles depuis la version 1.0. Même si cette conception ne s'applique pas à tous les cas de figure, vous pouvez considérer lors de la programmation que ces fonctions sont reconnues par tous les drivers de souris utilisés actuellement.

La plupart des fonctions de souris travaillent avec les registres AX, BX, CX et DX pour traiter les informations provenant du programme d'appel et lui retourner les résultats nécessaires dans ces mêmes registres. Les registres ES, SI et DS ne s'avèrent utiles que dans des cas exceptionnels, notamment lorsqu'il s'agit de transmettre l'adresse des buffers.

En règle générale, l'appel d'une fonction a pour conséquence de changer le contenu du registre adressé pour retourner les résultats de la fonction. Le contenu des autres registres reste inchangé. Dans la plupart des fonctions, le registre AX sert cependant à retourner l'état de la fonction qui indique un échec de l'opération en cas d'erreur.

Interruption 33h, Fonction 00h — Souris (à partir de 1.0)
Réinitialisation du driver de souris

Initialise le driver de souris et enclenche ainsi le test de la souris.

Entrée : AX = 0000h

Sortie : AX = FFFFh : dans ce cas, un driver de souris est installé
BX = Nombre de boutons de la Souris (à partir de 1.0)
AX = 0000h : erreur, aucun driver de souris n'est installé

Remarques :
- L'appel de cette fonction agit sur toute une série de paramètres de la souris qui peuvent être fixés à l'aide des différentes fonctions de la souris :
  - Le curseur de la souris est placé au centre de l'écran puis caché. Il apparaîtra désormais en mode graphique sous forme d'une flèche, alors qu'il sera représenté en mode de texte sous forme d'une case de texte en inversion vidéo. Le curseur de la souris est systématiquement affiché dans la page 0 de l'écran, indépendamment du mode. La zone de déplacement sélectionnée est la totalité de l'écran.
  - Les gestionnaires d'événements installés par un programme. Ils sont désactivés.
  - L'émulation du crayon optique est également désactivée.
  - La vitesse de la souris. Pour la vitesse horizontale de la souris, un rapport de 8 mickeys pour 8 points est défini. Pour la vitesse verticale, ce rapport est de 16 pour 8 (Mickey : chapitre 14).
  - Le seuil de doublement de la vitesse de la souris. Il est fixé à 64 mickeys par seconde.

Interruption 33h, Fonction 01h — Souris (à partir de 1.0)
Afficher le curseur de la souris sur l'écran

Après appel de cette fonction, le curseur de la souris devient visible sur l'écran et suit désormais les déplacements de la souris sur l'écran.

Entrée : AX = 0001h

Sortie : Aucune

Remarques :
- L'appel de cette fonction entraîne l'incrémentation d'un compteur interne, en fonction duquel le curseur de la souris est ou non affiché

sur l'écran. Si ce compteur contient la valeur 0, le curseur de la souris sera affiché sur l'écran, alors que la valeur -1 a pour effet de le faire disparaître de l'écran. Lorsque la fonction 00h (Reset) est appelée, ce compteur est tout d'abord fixé sur -1, pour prendre ensuite la valeur 0 lors du premier appel de cette fonction, ce qui se traduit par la réapparition du curseur sur l'écran.

- Le driver de la souris suit les déplacements de la souris même si le curseur de la souris n'est pas affiché sur l'écran. A la suite de l'appel de cette fonction, le curseur de la souris ne réapparaîtra donc pas nécessairement au même endroit que là où il se trouvait lorsqu'il avait été caché.

Interruption 33h, Fonction 02h Souris (à partir de 1.0)
Masquer le curseur de la souris

Cette fonction élimine le curseur de la souris de l'écran.

Entrée : AX = 0002h

Sortie : Aucune

Remarques :

- L'appel de cette fonction entraîne la décrémentation d'un compteur interne, en fonction duquel le curseur de la souris est ou non affiché sur l'écran. Si ce compteur contient la valeur 0, le curseur de la souris sera affiché sur l'écran, alors que la valeur -1 a pour effet de le faire disparaître de l'écran.
- Le driver de la souris suit les déplacements de la souris même après appel de cette fonction. A la suite de l'appel de la fonction 01h (Afficher le curseur de la souris), le curseur de la souris ne réapparaîtra donc pas nécessairement au même endroit que là où il se trouvait lorsqu'il avait été caché.

Interruption 33h, Fonction 03h Souris (à partir de 1.0)
Déterminer la position de la souris et l'état des boutons de la souris

L'appel de cette fonction fournit la position actuelle de la souris et l'état des différents boutons de la souris.

Entrée : AX = 0003h

Sortie : BX = Etat des boutons de la Souris (à partir de 1.0)

Bit	Signification
0	1 = bouton gauche de la souris appuyé
1	1 = bouton droit de la souris appuyé
2	1 = bouton central de la souris appuyé
3-15	Aucune (0)

CX = Position horizontale de la Souris (à partir de 1.0)
DX = Position verticale de la souris

Remarques :
- Les ordonnées renvoyées dans les registres CX et DX ne se rapportent pas à l'écran physique mais aux positions en points écran dans l'écran virtuel de la souris.
- Si la souris ne possède que deux boutons, les informations fournies sur le bouton central sont dépourvues de signification.

Interruption 33h, Fonction 04h — Souris (à partir de 1.0)
Déplacement du curseur de la souris

Déplace le curseur de la souris vers une position déterminée de l'écran, à moins qu'il n'ait été caché avec la fonction 02h.

Entrée : AX = 0004h
CX = Position horizontale de la Souris (à partir de 1.0)
DX = Position verticale de la souris

Sortie : Aucune

Remarques :
- Les ordonnées renvoyées dans les registres CX et DX ne se rapportent pas à l'écran physique mais aux positions en points écran dans l'écran virtuel de la souris.
- Si la position spécifiée se situe en dehors de la zone de déplacement fixée avec les fonctions 07h et 08h, les coordonnées sont adaptées pour que le curseur de la souris ne sorte pas de cette zone.
- Le curseur de la souris est bien déplacé vers le nouvelle position même s'il est caché, mais ce déplacement n'apparaîtra naturellement que lorsque le curseur sera à nouveau incrusté.

Interruption 33h, Fonction 05h — Souris (à partir de 1.0)
Déterminer combien de fois un bouton de la souris a été enfoncé

Indique au programme d'appel combien de fois un bouton de la souris a été appuyé depuis le dernier appel de cette fonction et où le curseur de la souris se trouvait la dernière fois que ce bouton avait été actionné.

Entrée : AX = 0005h
BX = Bouton de la souris appelé :
0 = bouton gauche de la Souris (à partir de 1.0)
1 = bouton droit de la Souris (à partir de 1.0)
2 = bouton central de la souris

Sortie : AX = Etat de tous les boutons de la Souris (à partir de 1.0)

Bit	Signification
0	1 = bouton gauche de la souris appuyé
1	1 = bouton droit de la souris appuyé
2	1 = bouton central de la souris appuyé
3-15	Aucune signification (0)

BX = Nombre de fois que le bouton de la souris appelé a été actionné depuis le dernier appel de cette fonction
CX = Position horizontale de la souris la dernière fois que le bouton a été actionné
DX = Position verticale de la souris la dernière fois que le bouton a été actionné

Remarques :
- Les ordonnées transmises dans les registres CX et DX ne se rapportent pas à l'écran physique mais aux positions en points écran dans l'écran virtuel de la souris.
- Le compteur du nombre de fois que le bouton de la souris appelé a été appuyé est à nouveau fixé sur 0 lorsque cette fonction est appelée.

Interruption 33h, Fonction 06h — Souris (à partir de 1.0)
Combien de fois un bouton de la souris a-t-il été relâché ?

Indique au programme d'appel combien de fois un bouton de la souris a été relâché depuis le dernier appel de cette fonction et où figurait le curseur de la souris lors du dernier de ces événements.

Entrée : AX = 0006h
BX = Bouton de la souris appelé :
0 = bouton gauche de la Souris (à partir de 1.0)
1 = bouton droit de la Souris (à partir de 1.0)
2 = bouton central de la souris

Sortie : AX = Etat de tous les boutons de la Souris (à partir de 1.0)

Bit	Signification
0	1 = bouton gauche de la souris appuyé
1	1 = bouton droit de la souris appuyé
2	1 = bouton central de la souris appuyé
3-15	Aucune signification (0)

BX = Nombre de fois que le bouton de la souris a été relâché
CX = Position horizontale de la souris la dernière fois que le bouton a été relâché
DX = Position verticale de la souris la dernière fois que le bouton a été relâché

Remarques :
- Les ordonnées transmises dans les registres CX et DX ne se rapportent pas à l'écran physique mais aux positions en points écran dans l'écran virtuel de la souris.
- Le compteur du nombre de fois que le bouton de la souris appelé a été relâché est à nouveau fixé sur 0 lorsque cette fonction est appelée.

Interruption 33h, Fonction 07h Souris (à partir de 1.0)
Fixer la zone de déplacement horizontale pour le curseur de la souris

Définit la zone horizontale à l'intérieur de laquelle le curseur de la souris peut se déplacer. L'utilisateur n'aura aucune possibilité de faire sortir le curseur de la souris de la zone ainsi fixée.

Entrée : AX = 0007h
CX = position horizontale minimale de la souris
DX = position horizontale maximale de la souris

Sortie : Aucune

Remarques :
- Les ordonnées transmises dans les registres CX et DX ne se rapportent pas à l'écran physique mais aux positions en points écran dans l'écran virtuel de la souris.
- Si le curseur de la souris se trouve en dehors de la zone spécifiée, il y est automatiquement ramené par le driver de la souris.
- Si la valeur spécifiée dans le registre DX est inférieure à celle dans CX, les deux paramètres sont échangés.

Interruption 33h, Fonction 08h Souris (à partir de 1.0)
Fixer la zone de déplacement verticale pour le curseur de la souris

Définit la zone verticale à l'intérieur de laquelle le curseur de la souris peut se déplacer. L'utilisateur n'aura aucune possibilité de faire sortir le curseur de la souris de la zone ainsi fixée.

Entrée : AX = 0008h
CX = position verticale minimale de la souris
DX = position verticale maximale de la souris

Sortie : Aucune

Remarques :
- Les ordonnées transmises dans les registres CX et DX ne se rapportent pas à l'écran physique mais aux positions en points écran dans l'écran virtuel de la souris.
- Si le curseur de la souris se trouve en dehors de la zone spécifiée, il y est automatiquement ramené par le driver de la souris.
- Si la valeur spécifiée dans le registre DX est inférieure à celle dans CX, les deux paramètres sont échangés.

Interruption 33h, Fonction 09h Souris (à partir de 1.0)
Définit le curseur de la souris en mode graphique

En mode graphique, un tableau de bits définit l'apparence du curseur de la souris, en fixant de quelle manière les points sous le curseur de la souris et les points du curseur de la souris doivent être combinés. Cette fonction sert à définir ce tableau de bits, et donc l'apparence du curseur de la souris.

Entrée : AX = 0009h
BX = Distance du point de référence au bord gauche du tableau de bits
CX = Distance du point de référence au bord supérieur du tableau de bits
ES:DX = Adresses de segment et d'offset du tableau de bits dans la mémoire

Sortie : Aucune

Remarques :
- Le tableau de bits comporte 64 octets, dont les 32 premiers définissent une combinaison ET et les 32 autres une combinaison OU avec le motif de points actuel.

Interruption 33h, Fonction 0Ah Souris (à partir de 1.0)
Définit le curseur de la souris en mode de texte

Fixe le motif de bits définissant l'apparence du curseur de la souris en mode de texte.

Entrée : AX = 000Ah
BX = Type de curseur de la Souris (à partir de 1.0)
0 = curseur logiciel
1 = curseur électronique
CX = Masque ET pour le curseur logiciel ou la ligne de départ du curseur électronique
DX = Masque XOR pour le curseur logiciel ou la ligne finale du curseur électronique

Sortie : Aucune

Remarques :
- Si le curseur logiciel est sélectionné, le code du caractère sous le curseur de la souris et l'octet d'attribut correspondant sont tout d'abord combiné par un ET binaire avec le masque dans le registre CX, avant qu'une combinaison OU EXCLUSIF (XOR) soit ensuite opérée avec la valeur dans le registre DX. L'octet d'attribut est combiné avec les octets de plus fort poids des deux valeurs (CH et DH), alors que le code de caractère est combiné avec les octets de plus faible poids (CL et DL).
- Le curseur électronique est le curseur clignotant de l'écran. Les lignes de départ et de fin doivent être comprises, en mode monochrome, entre 0 et 13, et en mode couleur, entre 0 et 7.
- Le chapitre 14 vous fournit de plus amples informations sur le travail avec cette fonction.

Interruption 33h, Fonction 0Bh Souris (à partir de 1.0)
Déterminer les distances de déplacement

Détermine la distance entre la position actuelle de la souris et la position de la souris lors du dernier appel de cette fonction.

Entrée : AX = 000Bh

Sortie : CX = Distance horizontale
DX = Distance verticale

Remarques :
- Les valeurs doivent être interprétées comme des nombres signés, les valeurs positives indiquant un déplacement vers le bas ou la droite

de l'écran, alors que les valeurs négatives indiquent un déplacement vers le haut ou la gauche de l'écran.

- Les valeurs ne sont pas fournies en points mais en mickeys.(1 mickey équivant à 1/200 de pouce).

Interruption 33h, Fonction 0Ch — Souris (à partir de 1.0)
Installer le gestionnaire d'événements

Cette fonction permet à un programme d'installer un gestionnaire d'événements, qui sera appelé lorsqu'interviendra un événement déterminé concernant la souris.

Entrée : AX = 000Ch
CX = Evénements devant déclencher l'appel du gestionnaire d'événements (masque d'événements)

Bit	Signification
0	La souris a été déplacée
1	Bouton gauche de la souris appuyé
2	Bouton gauche de la souris relâché
3	Bouton droit de la souris appuyé
4	Bouton droit de la souris relâché
5	Bouton central de la souris appuyé
6	Bouton central de la souris relâché
7-15	Aucune signification

ES:DX = Adresses de segment et d'offset du gestionnaire

Sortie : Aucune

Remarques :

- Le gestionnaire d'événements est appelé par le driver de la souris à travers une instruction assembleur FAR CALL et il doit donc de ce fait se terminer également par une instruction FAR RETurn. Comme pour un gestionnaire d'interruption, aucun des registres du processeur ne doit être restitué à l'utilisateur avec un contenu modifié.
- Le driver de la souris fournit les informations suivantes au gestionnaire d'événements lorsqu'il est appelé :

  AX = Masque d'événements. Les différents bits correspondent aux différents événements tels qu'ils ont été indiqués dans le registre CX lors de l'installation du gestionnaire d'événements. D'autres bits peuvent également être fixés car cette valeur reflète l'état actuel du driver de la souris, sans se limiter aux événements sélectionnés.

BX = Etat des boutons de la Souris (à partir de 1.0)

Bit	Signification
0	Bouton gauche de la souris a été appuyé
1	Bouton droit de la souris a été appuyé
2	Bouton central de la souris a été appuyé

CX = Position horizontale de la souris.
DX = Position verticale de la souris.
SI = Longueur du dernier déplacement horizontal de la souris.
DI = Longueur du dernier déplacement vertical de la souris.
DS = Segment de données du driver de la souris

- Les ordonnées transmises dans les registres CX et DX ne se rapportent pas à l'écran physique mais aux positions en points écran dans l'écran virtuel de la souris.
- Les distances spécifiées dans les registres SI et DI ne sont pas exprimées en points écran, mais en mickeys (1/200ème de pouce). Les valeurs transmises doivent être interprétées comme des nombres signés, les valeurs négatives indiquant un déplacement vers le haut ou la gauche de l'écran, alors que les valeurs positives indiquent un déplacement vers le bas ou la droite de l'écran.

Interruption 33h, Fonction 0Dh — Souris (à partir de 1.0)
Activer l'émulation du crayon optique

Active l'émulation du crayon optique et simule ainsi un crayon optique inexistant.

Entrée : AX = 000Dh

Sortie : Aucune

Remarques :
- L'émulation d'un crayon optique n'est intéressante qu'en liaison avec des programmes soutenant le crayon optique ou fournissant des mécanismes de test du crayon optique. C'est le cas par exemple du BASIC IBM avec son instruction PEN.
- Sous le mode d'émulation, la position de la souris est associée à la position du crayon optique et le fait d'actionner simultanément les boutons gauche et droit de la souris est interprété comme le fait d'appuyer sur le bouton du crayon optique.

Interruption 33h, Fonction 0Eh Souris (à partir de 1.0)
Désactiver l'émulation du crayon optique

Désactive à nouveau l'émulation du crayon optique qui avait été auparavant activée avec la fonction 0Dh.

Entrée : AX = 000Eh

Sortie : Aucune

Remarques :
- Voyez la fonction 0Dh.

Interruption 33h, Fonction 0Fh Souris (à partir de 1.0)
Définir le rapport entre les mickeys et les points

Cette fonction définit le rapport entre les mickeys et les points, dont dépend la sensibilité de la souris du point de vue de l'utilisateur, c'est-à-dire la vitesse avec laquelle le curseur de la souris peut être déplacé sur l'écran.

Entrée : AX = 000Fh
CX = Nombre de mickeys horizontaux représentant huit points
DX = Nombre de mickeys verticaux représentant huit points

Sortie : Aucune

Remarques :
- Peuvent être spécifiées des valeurs comprises entre 1 et 32767.
- Le réglage par défaut est de 8 mickeys horizontalement et de 16 mickeys verticalement, de sorte que le curseur de la souris se déplace deux fois plus "vite" horizontalement que verticalement.
- Les valeurs par défaut peuvent être à nouveau fixées en appelant la fonction 00h (Réinitialisation du driver de la souris).

Interruption 33h, Fonction 10h Souris (à partir de 1.0)
Définir la zone d'exclusion

Cette fonction permet à un programme de définir une zone quelconque de l'écran comme zone d'exclusion (une sorte de trou noir), qui aura pour effet de faire disparaître le curseur de la souris de l'écran dès qu'il pénétrera dans cette zone.

Entrée : AX = 0010h
CX = Ordonnée X du coin supérieur gauche de la zone d'exclusion

DX = Ordonnée Y du coin supérieur gauche de la zone d'exclusion
SI = Ordonnée X du coin inférieur droit de la zone d'exclusion
DI = Ordonnée Y du coin inférieur droit de la zone d'exclusion

Sortie : Aucune

Remarques :
- Les ordonnées transmises dans les registres CX, DX, SI et DI ne se rapportent pas à l'écran physique mais aux positions en points écran dans l'écran virtuel de la souris.
- La zone d'exclusion peut être à nouveau annulée en appelant la fonction 00h (réinitialisation du driver de souris) ou 01h (affichage du curseur de la souris).

Interruption 33h, Fonction 11h — Souris (à partir de 1.0)
Non documentée

Cette fonction n'intervient qu'à l'intérieur du gestionnaire de la souris. Elle n'est pas documentée et ne peut pas être appelée par un programme utilisateur.

Interruption 33h, Fonction 12h — Souris (à partir de 1.0)
Non documentée

Cette fonction n'intervient qu'à l'intérieur du gestionnaire de la souris. Elle n'est pas documentée et ne peut pas être appelée par un programme utilisateur.

Interruption 33h, Fonction 13h — Souris (à partir de 1.0)
Fixer le seuil pour le doublement de la vitesse de la souris

Si la vitesse de déplacement de la souris dépasse un seuil déterminé, le driver de la souris double le rapport entre les points et les mickeys, de façon à doubler la vitesse de la souris. Cette fonction permet de fixer le seuil au-delà duquel ce doublement intervient.

Entrée : AX = 0013h
DX = Seuil en mickeys par seconde

Sortie : Aucune

Remarques :
- 1 mickey équivaut à une longueur d'1/200 ou d'1/400 de pouce selon la souris.

- Pour empêcher tout doublement de la vitesse de la souris, il suffit de fixer le seuil suffisamment haut. Des vitesses de 5000 mickeys par seconde ne peuvent, par exemple, déjà plus être atteintes en pratique.

Interruption 33h, Fonction 14h — Souris (à partir de 1.0)
Echange des gestionnaires d'événements

Cette fonction permet à un programme d'installer un nouveau gestionnaire d'événements pour certains événements souris déterminés, tout en obtenant en même temps l'adresse de l'ancien gestionnaire d'événements.

Entrée : AX = 0014h
CX = Evénements devant entraîner l'appel du gestionnaire d'événements

Bit	Signification
0	La souris a été déplacée
1	Bouton gauche de la souris appuyé
2	Bouton gauche de la souris relâché
3	Bouton droit de la souris appuyé
4	Bouton droit de la souris relâché
5	Bouton central de la souris appuyé
6	Bouton central de la souris relâché
7-15	Aucune signification

ES:DX = Adresses de segment et d'offset du gestionnaire

Sortie : CX = Masque d'événements du gestionnaire d'événements installé jusqu'ici
ES:DX = Adresses de segment et d'offset du gestionnaire d'événements installé jusqu'ici

Remarques :
- Les informations nécessaires sur l'appel du gestionnaire d'événements sont fournies dans la description de la fonction 0Ch (Installer le gestionnaire d'événements).

Interruption 33h, Fonction 15h — Souris (à partir de 1.0)
Déterminer la taille du buffer d'état de la souris

Cette fonction fournit la taille du buffer d'état de la souris dans lequel un programme peut sauvegarder l'état complet du driver de la souris.

Entrée : AX = 0015h

Sortie : BX = Taille du buffer d'état de la souris en octets

Remarques :
- L'état de la souris peut être sauvegardé à l'aide de la fonction 16h.

Interruption 33h, Fonction 16h — Souris (à partir de 1.0)
Sauvegarder état de la souris

Lorsque cette fonction est appelée, le driver de la souris copie toutes les informations d'état gérées dans un buffer du programme d'appel.

Entrée : AX = 0016h
ES:DX = Adresses de segment et d'offset du buffer

Sortie : Aucune

Remarques :
- Le programme d'appel doit veiller à ce que le buffer soit suffisamment grand pour recevoir toutes les informations d'état. Avant d'appeler cette fonction, il ferait donc bien de déterminer à l'aide de la fonction 15h la taille du buffer d'état de la souris.
- L'appel de cette fonction peut par exemple être utile avant de faire exécuter un programme à l'aide de la fonction EXEC, pour que tous les paramètres de la souris puissent être à nouveau fixés sur leurs valeurs antérieures après exécution de ce programme, quelles que soient les modifications intervenues à l'intérieur du programme appelé.

Interruption 33h, Fonction 17h — Souris (à partir de 1.0)
Restaurer l'état de la souris

Lit tous les paramètres de la souris dans un buffer dans lequel ils avaient été sauvegardés auparavant avec la fonction 16h.

Entrée : AX = 0017h
ES:DX = Adresses de segment et d'offset du buffer d'état de la souris

Sortie : Aucune

Interruption 33h, Fonction 18h Souris (à partir de 1.0)
Installer un gestionnaire d'événements alternatif

Cette fonction permet à un programme d'installer au maximum trois gestionnaires d'événements différents, qui devront être appelés par le driver de la souris lorsqu'interviendront certains événements liés à la souris et au clavier.

Entrée : AX = 0018h
CX = Evénements devant entraîner l'appel du gestionnaire d'événements

Bit	Signification
0	La souris a été déplacée
1	Bouton gauche de la souris appuyé
2	Bouton gauche de la souris relâché
3	Bouton droit de la souris appuyé
4	Bouton droit de la souris relâché
5	Touche Shift actionnée pendant qu'un bouton de la souris était appuyé ou relâché
6	Touche Ctrl actionnée pendant qu'un bouton de la souris était appuyé ou relâché
7	Touche Alt actionnée pendant qu'un bouton de la souris était appuyé ou relâché
8-15	Aucune signification

ES:DX = Adresses de segment et d'offset du gestionnaire

Sortie : AX = 0018h: Le gestionnaire d'événements a été installé
FFFFh: Le gestionnaire d'événements n'a pu être installé

Remarques :

- Dans le masque d'événements du registre CX doit être fixé au moins l'un des bits 5 à 7 pour que l'événement soit combiné au fait d'actionner au moins une touche de contrôle. Si les touches de contrôle ne doivent pas être prises en compte, il est préférable d'utiliser les fonctions 0Ch ou 14h.
- Une erreur peut se produire si trois gestionnaires d'événements alternatifs ont déjà été installés ou bien si un gestionnaire d'événements a déjà été enregistré avec la même masque d'événements.
- Le gestionnaire d'événements est appelé par le driver de la souris à travers une instruction assembleur FAR CALL et il doit donc de ce fait se terminer également par une instruction FAR RET. Comme pour un gestionnaire d'interruption, aucun des registres du processeur ne doit être restitué à l'utilisateur avec un contenu modifié.
- Le driver de la souris fournit les informations suivantes au gestionnaire d'événements lorsqu'il est appelé :

AX = Masque d'événements. Les différents bits correspondent aux différents événements tels qu'ils ont été indiqués dans le registre CX lors de l'installation du gestionnaire d'événements. D'autres bits peuvent également être fixés car cette valeur reflète l'état actuel du driver de la souris, sans se limiter aux événements sélectionnés.
BX = Etat des boutons de la Souris (à partir de 1.0)

Bit	Signification
0	Bouton gauche de la souris a été appuyé
1	Bouton droit de la souris a été appuyé
2	Bouton central de la souris a été appuyé

CX = Position horizontale de la souris.
DX = Position verticale de la souris.
SI = Longueur du dernier déplacement horizontal de la souris.
DI = Longueur du dernier déplacement vertical de la souris.
DS = Segment de données du driver de la souris

- Les ordonnées transmises dans les registres CX et DX ne se rapportent pas à l'écran physique mais aux positions en points écran dans l'écran virtuel de la souris.
- Les distances spécifiées dans les registres SI et DI ne sont pas exprimées en points écran, mais en mickeys (1/200ème de pouce). Les valeurs transmises doivent être interprétées comme des nombres signés, les valeurs négatives indiquant un déplacement vers le haut ou la gauche de l'écran, alors que les valeurs positives indiquent un déplacement vers le bas ou la droite de l'écran.

Interruption 33h, Fonction 19h — Souris (à partir de 1.0)
Déterminer l'adresse d'un gestionnaire d'événements alternatif

Renvoie au programme d'appel l'adresse d'un gestionnaire d'événements alternatif.

Entrée : AX = 0019h
CX = Masque d'événements du gestionnaire d'événements appelé

Sortie : AX = 0000h : OK, dans ce cas :
CX = Masque d'événements
ES:DX = Adresses de segment et d'offset du gestionnaire

Remarques :
- Vous trouverez dans la description de la fonction 18h de plus amples informations sur la signification des différents bits du masque d'événements.

- L'appel de fonction échoue si aucune gestionnaire d'événements alternatif n'a encore été installé avec le masque d'événements spécifié.

Interruption 33h, Fonction 1Ah Souris (à partir de 1.0)
Fixer la sensibilité de la souris

Cette fonction est une combinaison des fonctions 0Fh et 13h, qui permettent respectivement de fixer le rapport entre déplacement de la souris et déplacement du curseur de la souris, d'une part, et de fixer le seuil de doublement de la vitesse de la souris, d'autre part.

Entrée : AX = 001Ah
BX = Nombre de mickeys représentant huit points horizontalement
CX = Nombre de mickeys représentant huit points verticalement
DX = Seuil pour le doublement de la vitesse de la souris

Sortie : Aucune

Remarques :
- Pour les paramètres dans les registres CX et DX sont autorisées des valeurs comprises entre 1 et 32767.
- Le réglage par défaut est de 8 mickeys horizontalement et de 16 mickeys verticalement, de sorte que le curseur de la souris se déplace deux fois plus "vite" horizontalement que verticalement.
- Pour empêcher tout doublement de la vitesse de la souris, il suffit de fixer le seuil suffisamment haut. Des vitesses de 5000 mickeys par seconde ne peuvent, par exemple, déjà plus être atteintes en pratique.
- Les valeurs par défaut peuvent être à nouveau fixées en appelant la fonction 00h (Réinitialisation du driver de la souris).

Interruption 33h, Fonction 1Bh Souris (à partir de 1.0)
Déterminer la sensibilité de la souris

Renvoie les paramètres qui ont été fixés à travers la fonction 1Ah ou à travers les fonctions 0Fh et 13h.

Entrée : AX = 001Bh

Sortie : BX = Nombre de mickeys représentant huit points horizontalement
CX = Nombre de mickeys représentant huit points verticalement
DX = Seuil pour le doublement de la vitesse de la souris

Interruption 33h, Fonction 1Ch Souris (à partir de 1.0)
Fixer la fréquence d'interruption de l'électronique de la souris

Détermine à quelle fréquence l'électronique de la souris lit la position actuelle de la souris et l'état des boutons de la souris pour les retransmettre au driver de la souris.

Entrée : AX = 001C
BX = Fréquence d'interruption

Bit	Signification
0	Pas d'interruptions
1	30 interruptions par seconde
2	50 interruptions par seconde
3	100 interruptions par seconde
4	200 interruptions par seconde
5-15	Aucune signification (0)

Sortie : Aucune

Remarques :
- Cette fonction n'est disponible que pour la souris InPort.
- Si plus d'un bit est fixé dans le registre BX, c'est le bit fixé de plus faible poids qui compte.
- La précision de la souris est proportionnelle à la fréquence des interruptions, mais la vitesse du programme de premier plan est au contraire inversement proportionnelle à cette fréquence, puisqu'il est interrompu plus souvent par le driver de souris

Interruption 33h, Fonction 1Dh Souris (à partir de 1.0)
Fixer une page écran pour le curseur de la souris

Fixe la page écran dans laquelle le driver de la souris doit afficher le curseur de la souris.

Entrée : AX = 001Dh
BX = Numéro de la page écran

Sortie : Aucune

Remarques :
- La valeur par défaut est la page écran 0.
- L'appel de cette fonction ne présente naturellement d'intérêt que si votre programme travaille avec plusieurs pages écran, telles qu'elles sont offertes par les cartes CGA, EGA et VGA.

Interruption 33h, Fonction 1Eh — Souris (à partir de 1.0)
Déterminer page écran du curseur de la souris

Détermine la page écran dans laquelle le curseur de la souris est représenté par le driver de la souris.

Entrée : AX = 001Eh

Sortie : BX = Numéro de la page écran

Interruption 33h, Fonction 1Fh — Souris (à partir de 1.0)
Désactiver le driver de la souris

Désactive le driver actuel de la souris et fournit l'adresse du gestionnaire d'interruption précédent pour l'interruption 33h.

Entrée : AX = 001Fh

Sortie : AX = FFFFh : Erreur
001Fh : Tout va bien, auquel cas
ES:BX = Adresses de segment et d'offset du driver de souris antérieur

Remarques :
- Lorsque cette fonction est appelée, le driver de souris libère à nouveau toutes les routines d'interruption qu'il avaient installées lors de son activation. Seul fait exception le gestionnaire de l'interruption 33h, mais le programme d'appel peut charger à nouveau sa valeur d'origine dans ce vecteur d'interruption également, car son adresse est renvoyée dans la paire de registres ES:BX.

Interruption 33h, Fonction 20h — Souris (à partir de 1.0)
Activer le driver de la souris

Active un driver de souris désactivé auparavant avec la fonction 1Fh.

Entrée : AX = 0020h

Sortie : Aucune

Interruption 33h, Fonction 21h — Souris (à partir de 1.0)
Réinitialiser le driver de la souris

Cette fonction effectue une réinitialisation du driver de la souris, en cachant le curseur de la souris et en désactivant les gestionnaires d'événements déjà installés.

Entrée : AX = 0021h

Sortie : AX = FFFFh : Erreur
0021h : Tout va bien, dans ce cas :
BX = nombre de boutons

Remarque :
- Cette fonction, contrairement à la fonction 00h ne réinitialise pas le hard de la souris.

Interruption 33h, Fonction 22h — Définir la langue étrangère pour les messages
Souris (à partir de 1.0)

La version internationale du driver de la souris reconnaît plusieurs langues étrangères et peut retourner les messages dans ces langues. Cette fonction détermine la langue étrangère souhaitée.

Entrée : AX = 0022h
BX = Numéro de code de la langue étrangère (voir plus bas)

Sortie : aucune

Remarques :
- Les codes suivants sont utilisés pour désigner les langues étrangères :

  0 Anglais
  1 Français
  2 Néerlandais
  3 Allemand
  4 Suédois
  5 Finlandais
  6 Espagnol
  7 Portugais
  8 Italien

- Vu que cette fonction ne retourne pas un état d'erreur, il n'est pas toujours possible de déterminer si une version internationale du driver de la souris existe réellement et de définir l'option souhaitée. La fonction 23h permet cependant de contrôler facilement cet aspect.

Interruption 33h, Fonction 23h — Souris (à partir de 1.0)
Vérifier l'existence d'une langue étrangère pour les messages

La version internationale du driver de la souris reconnaît plusieurs langues étrangères et peut retourner les messages dans ces langues. Cette fonction retourne le numéro de code de la langue étrangère actuellement définie.

Entrée : AX = 0023h

Sortie : BX = Numéro de code de la langue étrangère choisie

Remarques :
- Reportez-vous à la fonction 22h pour obtenir les codes des langues étrangères.
- Si vous n'êtes pas en présence d'une version internationale du driver de la souris, la fonction retourne la valeur 0.

Interruption 33h, Fonction 24h — Souris (à partir de 1.0)
Fournit le type de souris

Fournit le type de souris ainsi que son numéro de version.

Entrée : AX = 0024h

Sortie :
- BH = Partie entière du numéro de version
- BL = Partie décimale du numéro de version
- CH = Type de souris
  - 1 = Souris bus
  - 2 = Souris sérielle
  - 3 = Souris InPort
  - 4 = Souris PS/2
  - 5 = Souris HP
- CL = Numéro IRQ
  - 0 = PS/2
  - 2, 3, 4, 5 ou 7 = Numéro PC

Remarque :
- Un driver de souris de version 6.24 aura 6 dans le registre BH et 24h dans le registre BL.

Interruption 33h, Fonction 25h Souris (à partir de 6.26)
Obtenir des informations générales

Cette fonction retourne des informations générales sur le driver de souris, par exemple le type du driver et l'état du curseur de la souris.

Entrée : AX = 0025h

Sortie : AX = Informations générales (voir plus bas)
BX,
CX,
DX = Informations d'état concernant OS/2 uniquement

Remarques :
- Les informations générales sont transmises dans les divers tableaux de bits du registre AX. Il reproduit les informations suivantes :

  Bit 15 : Type du driver
  - 0b = Le driver se trouve dans un fichier COM
  - 1b = Le driver est sauvegardé en tant qu'un driver de périphérique. Il a été inséré par un fichier CONFIG.SYS.

  Bits 12 & 13 : Informations sur le curseur de la souris
  - 00b = Affichage d'un curseur texte logiciel
  - 01b = Affichage d'un curseur texte électronique
  - 10b, 11b = Affichage d'un curseur graphique
  - Bits 8 à 11 = Valeur binaire représentant la fréquence d'interruption de l'électronique de la souris

- Les arguments obtenus dans les registres BX, CX et DX ne sont retournés que par un driver de souris sous OS/2. Ils ne jouent aucun rôle dans la programmation DOS.

Interruption 33h, Fonction 26h Souris (à partir de 6.26)
Obtenir l'extension de l'écran de souris virtuel

Cette fonction retourne l'extension absolue de l'écran de souris virtuel et indique si le driver de souris est actuellement activé ou désactivé.

Entrée : AX = 0026h

Sortie : BX = Etat du driver de souris
CX = Ordonnée X maximale
DX = Ordonnée Y maximale

Remarques :

- En ce qui concerne l'état du driver de souris, 0 représente l'état désactivé et les autres valeurs l'état activé. Cet état peut être modifié à l'aide des fonctions 1Fh et 20h.
- Les valeurs retournées dans les registres CX et DX concernent la taille absolue de l'écran de souris virtuel et non les valeurs spécifiées à l'aide des fonctions 07h et 08h. Elles peuvent être obtenues au moyen de la fonction 31h.

Interruption 33h, Fonction 27h — Souris (à partir de 7.01)
Obtenir les masques de bits du curseur d'écran

Cette fonction permet de prendre connaissance des masques de bits en cours pour la création du curseur logiciel ou pour la ligne de départ et la ligne finale du curseur électronique. En outre, cette fonction donne des informations sur le déplacement de la souris depuis sa dernière utilisation.

Entrée : AX = 0027h

Sortie :
- AX = Masque ET du curseur logiciel ou ligne de départ du curseur électronique
- BX = Masque XOR du curseur logiciel ou ligne finale du curseur électronique
- CX = Longueur du déplacement horizontal en mickeys
- DX = Longueur du déplacement vertical en mickeys

Remarques :

- Si un curseur électronique se trouve activé au moment de l'appel de la fonction, celle-ci retourne la ligne de départ et la ligne finale (en lignes Scan). Au contraire, si le curseur logiciel est actif, la fonction retourne les masques de bits en cours.
- Bien que cette fonction soit reconnue depuis la version 7.01 du driver de souris, les informations concernant les lignes de départ et finales du curseur électronique ne sont transmises qu'à partir de la version 7.02. Cela élimine un risque d'erreur dans la version 7.01 qui ne fonctionne librement qu'avec le curseur logiciel.
- Les distances de déplacement retournées dans les registres CX et DX ne sont pas modifiées à cause des différents paramètres logiciels tels que le seuil du doublement de la vitesse de la souris ou la table d'accélération. Elles représentent donc les valeurs transmises directement au driver de souris par le hard.

Interruption 33h, Fonction 28h Souris (à partir de 7.0)
Fixer le mode vidéo

Cette fonction permet de spécifier un mode vidéo à condition qu'il soit reconnu par l'équipement vidéo en cours.

Entrée : AX = 0028h
CX = Numéro de code du mode vidéo
DX = Taille de la fonte écran

Sortie : CX = Etat de la fonction

Remarques :
- La liste des modes vidéo disponibles et leurs numéros de code peut être obtenue par la fonction 29h.
- La valeur 0 est retournée dans le registre CX si le mode vidéo spécifié est reconnu par le matériel en cours. Dans ce cas, il peut être défini sinon le numéro de code est renvoyé par le registre AX.
- Certains modes vidéo utilisant les polices à tailles variables attendent le paramètre d'appel en DX. La taille de la police le long de l'axe X doit alors être codifiée dans l'octet de poids fort de DX et l'étirement le long de l'axe Y dans l'octet de poids faible.

Interruption 33h, Fonction 29h Souris (à partir de 7.0)
Obtenir la liste des modes vidéo disponibles

Cette fonction permet à un programme utilisateur de demander la liste des modes vidéo reconnus par le matériel vidéo en cours.

Entrée : AX = 0029h
CX = Demander le premier mode vidéo ou le suivant

Sortie : BX = Adresse de segment d'une chaîne
CX = Numéro de code du mode vidéo
DX = Adresse d'offset d'une chaîne

Remarques :
- Pour demander la liste complète des modes vidéo disponibles, il convient d'exécuter un appel multiple. Lors du premier appel, la fonction doit retourner la valeur 0 pour le paramètre de CX et une valeur différente de 0 lors des appels suivants.

  Chaque appel décrit un mode vidéo différent jusqu'à ce que la valeur 0 obtenue dans le registre CX indique que tous les modes vidéo ont été sollicités.

- Outre le numéro de code du mode vidéo, cette fonction ne fournit pas toujours le type du mode vidéo (texte ou graphique), sa résolution et le nombre de couleurs à afficher. Généralement, cette information figure dans une chaîne ASCII dont l'adresse est retournée dans la paire des registres BX:DX.

  Dans la mesure où une telle chaîne est disponible et les deux registres cités ne contiennent pas la valeur 0, elle se termine par un caractère dollar et un octet nul.

Interruption 33h, Fonction 2Ah — Souris (à partir de 7.02)
Obtenir des informations sur le curseur de la souris

Cette fonction retourne diverses informations concernant le curseur et l'électronique de la souris.

Entrée : AX = 002Ah

Sortie : AX = Compteur de curseur interne
BX = Ordonnée X du Hot-Spot
CX = Ordonnée Y du Hot-Spot
DX = Type de souris

Remarques :

- Le compteur de souris interne indique si le curseur est actuellement visible ou non. Vous pouvez agir indirectement sur lui en appelant les fonctions 01h et 02h. La valeur 0 signale alors que le curseur de souris est pour l'instant invisible. Toute autre valeur indique que le curseur de souris se trouve sur l'écran.
- En ce qui concerne le curseur graphique, le Hot-Spot reproduit le point à l'intérieur du masque de bits du curseur retourné à la suite d'une demande de sa position. Pour exprimer sa suppression du coin supérieur gauche du masque de bits, une valeur positive signée comprise entre -128 et 127 est retournée dans les registres BX ou CX.
- Dans le registre DX, la fonction retourne un code de type selon la table suivante :

  0 = pas de souris
  1 = souris bus
  2 = souris sérielle
  3 = souris InPort
  4 = souris PS/2
  5 = souris HP

Interruption 33h, Fonction 2Bh — Souris (à partir de 7.0)
Déterminer les courbes d'accélération

Les quatre courbes d'accélération gérées par le driver de souris sur le plan interne peuvent être chargées à l'aide de cette fonction. L'une d'entre elles peut être sélectionnée comme la courbe en cours.

Entrée : AX = FFFFh : Erreur
0021h : Tout va bien, dans ce cas
BX = Nombre d'appuis sur la souris

Remarques :
- Cette fonction permet de commuter entre les courbes d'accélération prédéfinies dans le driver, la valeur -1 représentant le numéro de la courbe en cours. Il n'est pas indispensable de transmettre un buffer avec les données-clé des courbes d'accélération.
- Les quatre courbes d'accélération sont décrites à l'aide d'une structure de données définie dans la mémoire par le programme d'appel. Elle doit être transmise au moyen du registre ES:SI. Reportez-vous au chapitre XXX pour obtenir de plus amples informations à ce sujet.

Interruption 33h, Fonction 2Ch — Souris (à partir de 7.0)
Lire la courbe d'accélération en cours

Cette fonction sert à lire la courbe d'accélération en cours.

Entrée : AX = 002Ch

Sortie :
AX = Etat de la fonction
BX = Numéro de la courbe d'accélération en cours (0 à 3)
ES = Adresse de segment du buffer
SI = Adresse d'offset du buffer

Remarques :
- Après appel de cette fonction, il faut obligatoirement demander l'état de la fonction dans le registre AX. La courbe d'accélération ne peut être lue que lorsque la valeur 0 a été retournée dans le registre.
- Après appel de la fonction, le contenu des registres ES:SI indique le buffer renfermant les données-clé de la courbe d'accélération en cours. La structure de ces données correspond au format identique à celui attendu lors de la définition d'une courbe d'accélération à l'aide de la fonction 2Bh.

Interruption 33h, Fonction 2Dh — Souris (à partir de 7.0)
Définir/Obtenir la courbe d'accélération en cours

Cette fonction a pour rôle d'activer l'une des quatre courbes d'accélération ou de retourner la courbe en cours.

Entrée :
- AX = 002Dh
- BX = -1 : Retourner la courbe d'accélération en cours
  = 1 - 4 : Activer cette courbe

Sortie :
- AX = Etat de la fonction
- BX = Numéro de la courbe d'accélération en cours (1-4)
- ES = Adresse de segment du buffer avec le nom de la courbe d'accélération
- SI = Adresse d'offset du buffer avec le nom de la courbe d'accélération

Remarques :

- Si la valeur -1 est transmise dans le registre BX lors de l'appel de la fonction, celle-ci retourne alors des informations sur la courbe d'accélération en cours dans les registres BX, ES et SI. En revanche, si l'une des quatre courbes d'accélération est activée lors de l'appel, une valeur comprise entre 1 et 4 doit être chargée dans le registre BX avant l'appel de la fonction. Dans ce cas, seul l'état de la fonction est retourné en BX en guise de résultat.
- Après appel de la fonction, l'état contient la valeur 0 si la fonction s'est exécutée correctement, sinon une valeur quelconque.
- Lorsqu'une demande est formulée pour connaître la courbe d'accélération en cours, les registres ES:SI indiquent la chaîne ainsi que les autres données-clé concernant la courbe qui ont été définies lors de l'appel de la fonction 2Bh. La chaîne comprend 16 octets et ne se termine pas par un caractère spécial ($ ou octet nul). L'évaluation de cette chaîne permet de connaître le nom symbolique de la courbe d'accélération.

Interruption 33h, Fonction 2Eh — Souris (à partir de 6.26)
Non documentée

Cette fonction n'intervient qu'à l'intérieur du driver de souris. Elle n'est pas documentée et ne peut donc pas être appelée par un programme utilisateur.

Interruption 33h, Fonction 2Fh — Souris (à partir de 7.02)
Réinitialisation de l'électronique de la souris

Cette fonction réinitialise l'électronique de la souris sans agir sur les paramètres logiciels tels que le masque de bits du curseur de la souris, le seuil du doublement de la vitesse de la souris ou les courbes d'accélération.

Entrée : AX = 002Fh

Sortie : AX = FFFFh : Tout va bien
0 = Erreur

Remarques :
- Cette fonction représente le contraire de la fonction 21h qui ne réinitialise que les paramètres logiciels.

Interruption 33h, Fonction 30h — Souris (à partir de 7.04)
Définir/Obtenir les paramètres de la souris Ballpoint

Cette fonction est conçue spécialement pour les besoins de la souris Ballpoint. Elle permet de définir ou lire les paramètres spécifiques à cette souris. Il s'agit notamment de connaître l'orientation de la souris Ballpoint et les trois boutons considérés comme activés parmi les quatre qui sont disponibles.

Entrée : AX = 0030h
BX = Angle de rotation
CX = Code de commande

Sortie : AX = Etat de la fonction
BX = Angle de rotation
CX = Boutons activés

Remarques :
- Après appel de la fonction, il faut obligatoirement vérifier l'état dans le registre AX. La valeur -1 indique en effet qu'aucune souris Ballpoint n'est installée. Toute autre valeur signale en revanche l'existence d'une souris Ballpoint et reproduit l'état des divers boutons de la souris. Les différents boutons sont représentés par les bits suivants à l'intérieur du registre AX :

  Bit 2 = Bouton 4
  Bit 3 = Bouton 2
  Bit 4 = Bouton 3
  Bit 5 = Bouton 1

  les autres bits contiennent la valeur 0.

- Si cette fonction est utilisée pour demander l'angle de rotation en cours et les boutons activés, il faut préalablement charger le registre CX avec la valeur 0 avant l'appel. Un angle de rotation en BX ne s'avère alors pas indispensable.

  Comme résultat de la fonction, on obtient l'angle de rotation en BX avec une valeur comprise entre 0 et 360 (degrés). En outre, les deux boutons activés peuvent être lus dans l'octet de poids fort de CX alors que les boutons désactivés sont codifiés dans l'octet de poids faible de ce registre. A l'intérieur des deux octets, les bits suivants sont disponibles pour les différents boutons :

  Bit 2 = Bouton 4
  Bit 3 = Bouton 2
  Bit 4 = Bouton 3
  Bit 5 = Bouton 1

  les autres bits contiennent la valeur 0.

- Si l'angle de rotation actuel de la souris Ballpoint est communiqué au driver de souris à l'aide de cette fonction et les deux boutons activés sont sélectionnés, les boutons activés et désactivés sont alors à codifier en CX. La codification s'effectue selon le même principe que le renvoi des boutons activés ou désactivés à la suite d'un appel (voir plus haut). Comme angle de rotation, la fonction attend une valeur comprise entre 0 et 360 en BX.

Interruption 33h, Fonction 31h — Souris (à partir de 7.05)
Déterminer l'étirement de l'écran virtuel

Cette fonction offre à un programme la possibilité de demander l'étirement de l'écran de souris virtuel dans le mode vidéo actuel.

Entrée : AX = 0031h

Sortie : AX = Ordonnée X minimale
BX = Ordonnée Y minimale
CX = Ordonnée X maximale
DX = Ordonnée Y maximale

Remarques :

- La taille de l'écran de souris virtuel peut également être modifiée par les fonctions 07h et 08h qui réduisent le champ de déplacement du curseur.

Interruption 33h, Fonction 32h — Souris (à partir de 7.05)
Déterminer les fonctions disponibles

Cette fonction permet tout simplement de connaître toutes les fonctions reconnues depuis la fonction 25h.

Entrée : AX = 0032h

Sortie : AX = Fonctions reconnues

Remarques :

- Le résultat de la fonction retourné en AX doit être considéré comme un tableau de bits où chaque bit représente une fonction. S'il est réglé, cela signifie que la fonction est reconnue.

  Le bit 15 représente la fonction 25h, le bit 14 la fonction 26h, le bit 13 la fonction 27h, etc.

  Bit 15 = Fonction 25h
  Bit 45 = Fonction 26h
  Bit 13 = Fonction 27h
  Bit 12 = Fonction 28h
  Bit 11 = Fonction 29h
  Bit 10 = Fonction 2Ah
  Bit 9 = Fonction 2Bh
  Bit 8 = Fonction 2Ch
  Bit 7 = Fonction 2Dh
  Bit 6 = Fonction 2Eh
  Bit 5 = Fonction 2Fh
  Bit 4 = Fonction 30h
  Bit 3 = Fonction 31h
  Bit 1 = Fonction 32h
  Bit 0 = Fonction 33h

Interruption 33h, Fonction 33h — Souris (à partir de 7.05)
Définir les paramètres

Tous les paramètres spécifiés par le logiciel ou le matériel à l'intérieur du driver de souris peuvent être lus dans un buffer à l'aide de cette fonction. La structure de ce buffer n'étant pas encore documentée avec précision, l'utilisation de cette fonction semble peu intéressante.

Entrée : AX = 0033h
CX = Taille du buffer
ES = Adresse de segment du buffer
DX = Adresse d'offset du buffer

Sortie : AX = 0
CX = Nombre d'octets dans le buffer
ES = Adresse de segment du buffer
DX = Adresse d'offset du buffer

Interruption 33h, Fonction 34h Souris (à partir de 8.0)
Déterminer l'état du fichier MOUSE.INI

Cette fonction est conçue pour la programmation de la souris sous Windows. Elle retourne la désignation exacte du chemin du fichier d'initialisation MOUSE.INI sous la forme d'une chaîne ASCII dans un buffer.

Entrée : AX = 0034h

Sortie : AX = 0
ES = Adresse de segment du buffer
DX = Adresse d'offset du buffer

Remarques :
- La chaîne retournée se termine par un octet nul.
- La désignation de chemin du fichier MOUSE.INI provient de la variable d'environnement MOUSE. Si cette variable n'est pas définie, MOUSE.INI est recherché dans le répertoire du driver de souris et la fonction retourne une chaîne correspondante.

I. Interruptions matérielles

Cette annexe décrit les interruptions matérielles générées par les différentes cartes d'extensions et circuits auxiliaires d'un PC. Elle comporte aussi les "exceptions", c'est-à-dire les interruptions matérielles déclenchées à l'initiative du processeur lui-même.

Interruption 0	Division par 0.

Le 8088 dispose des instructions machine DIV et IDIV qui effectuent une division entière. Les règles arithmétiques usuelles interdisent toute division par zéro. C'est pourquoi le processeur déclenche dans ce cas une interruption 0. Lors du lancement du système, le vecteur associé est dirigé par DOS sur une routine qui affiche un message du genre "Division par zéro", puis revient par IRET au programme interrompu.

Interrupton 1	Pas à pas

Cette interruption est appelée par le processeur à la suite de chaque instruction lorsque le bit TRAP du registre des indicateurs est à 1. Les débogueurs exploitent ce mode pour pouvoir suivre l'exécution d'un programme. Ils fournissent alors un gestionnaire pour cette interruption.

Pour éviter qu'elle ne se déclenche à l'intérieur même de la routine d'interruption (ce qui conduirait à des appels récursifs en nombre infini et au débordement de la pile), le processeur annule le bit TRAP au début du gestionnaire. Auparavant comme c'est le cas pour toutes les interruptions, le registre des indicateurs et notamment le bit TRAP aura été sauvegardé sur la pile.

Dès que la routine d'interruption se termine par une instruction IRET, le processeur rétablit automatiquement l'ancien contenu du bit TRAP en récupérant sur la pile le registre des indicateurs. De ce fait, l'interruption 1 sera à nouveau appelée dès que la prochaine instruction du programme à surveiller aura été exécutée.

Si le programmeur a recueilli toutes les informations nécessaires sur le programme, le mode TRAP (autrement dit le bit TRAP) peut être désactivé. Mais comment faire, alors que le programme suivi ne sait pas qu'il est surveillé et ne peut donc pas contenir d'instruction propre à mettre à zéro le bit TRAP du registre des indicateurs ?

La solution de ce problème se trouve encore dans la routine de l'interruption 1. C'est à elle d'annuler le bit TRAP mais ce n'est pas si simple car ce bit été systématiquement mis à 0 à l'entrée dans le programme et il sera rempli à la sortie, lors de la récupération du registre des indicateurs, par l'ancienne valeur 1. Il faut donc avant de sortir de la

routine d'interruption reprendre le registre des indicateurs sur la pile, y annuler le bit TRAP et remettre ensuite sur la pile, à son emplacement d'origine, le registre des indicateurs. Au moment de l'instruction IRET, le processeur récupère le registre des indicateurs comme si de rien n'était, mais le bit TRAP étant cette fois ci à 0, le programme continue son exécution dans les conditions normales.

Ces possibilités ne sont guère exploitées dans les applications ordinaires. C'est pourquoi le BIOS fait pointer le vecteur de l'interruption 1 sur un simple IRET, de sorte qu'il ne se passera rien de spécial si un programme met par hasard le bit TRAP à 1. Le seul effet sera un ralentissement de l'exécution puisqu'après chaque instruction l'interruption 1 sera systématiquement invoquée. L'interruption 1 par contre est très utile dans les programmes de test du genre DEBUG car elle permet de suivre à la trace des programmes en inspectant une à une les instructions et leurs conséquences.

Interruption 2	NMI

Cette interruption est désignée par le signe NMI qui veut dire "Non Maskable Interrupt". En conséquence, son exécution ne peut pas être inhibée par l'instruction CLI qui est en principe chargée de couper court à toute interruption. Dans un PC, l'interruption NMI est chargée d'attirer l'attention sur les erreurs en mémoire vive qui laissent présager un défaut de circuit. Il y urgence car une erreur de ce type peut altérer gravement le fonctionnement du système, et même l'endommager. C'est pourquoi le système déclenche l'interruption NMI même si toutes les autres sont inhibées

Au moment de l'initialisation du système, le vecteur d'interruption correspondant reçoit la référence d'une routine du BIOS charger d'afficher un message d'erreur et d'arrêter le système.

Interruption 3	Point d'arrêt

Cette interruption est aussi exploitée par les débogueurs. Par rapport à toutes les autres interruptions, elle présente la particularité de pouvoir être appelée à l'aide d'une instruction spéciale qui n'appartient qu'à elle seule. Les autres interruptions sont en effet appelées par une instruction machine sur 2 octets (premier octet = CDh, second octet = numéro de l'interruption) alors que l'interruption 3 se déclenche par une instruction particulière sur un seul octet seulement (le code CCh). Cette interruption convient ainsi parfaitement pour tester un programme, lorsqu'on désire l'exécuter jusqu'à un endroit déterminé où il devra s'interrompre et révéler le contenu courant des registres. Dans un débogueur on obtient ce résultat en insérant un appel à l'interruption 3 à l'endroit du programme où celui-ci devra être interrompu. Derrière cette interruption se cache en général un gestionnaire du débogueur qui affiche les contenus des registres et tout ce qu'il faut.

Vous pourriez objecter qu'à la place de l'interruption 3 toute autre interruption ferait l'affaire, pourvu que dans le gestionnaire associé on installe une routine d'affichage des registres. Cette remarque est certes judicieuse mais il ne faut pas oublier que l'interruption 3 présente l'avantage de tenir sur un seul octet. Pour comprendre la portée de cet avantage prenons un programme fictif.

Ce programme contient une instruction RET. Cette instruction prend un octet et constitue la fin d'un sous-programme. Une autre sous-programme se trouve à la suite qui commence par une instruction machine quelconque. L'utilisateur souhaite examiner les registres à la fin du premier sous-programme, il disposera donc un point d'arrêt (c'est-à-dire un appel à l'interruption 3) à la place de l'instruction RET. Il est tout à fait avantageux que cet appel ne prenne qu'un seul octet. S'il occupait davantage de place, par exemple 2 ou 3 octets, il n'écraserait pas seulement l'instruction RET mais également tout ou partie de la première instruction du sous-programme qui suit. Si au cours de l'exécution le second sous-programme est invoqué, il se produirait une erreur capable de planter le système. Mais en limitant le point d'arrêt à un seul octet, cet incident ne se produit pas car le point d'arrêt ne peut pas écraser plus d'une seule instruction.

Jusqu'à son exploitation par un programme de mise au point, cette interruption n'a généralement pas d'application. Le système DOS l'aiguille sur une routine simplement constituée de l'instruction IRET Interrupt Return) qui rend la main au programme interrompu sans rien faire.

Interruption 4	Débordement

Cette interruption peut être appelée par une certaine instruction liée à une condition. Il s'agit de l'instruction machine INTO (INTerruption on Overflow) qui ne déclenche l'interruption 4 que si le bit de débordement (Overflow) du registre des indicateurs est à 1 au moment de son exécution. Ce sera notamment le cas à la suite d'opérations arithmétiques (par exemple après une multiplication avec l'instruction MUL) si le résultat de l'opération ne peut plus être représenté avec un nombre déterminé de bits. Cette interruption peut naturellement également être appelée à l'aide de l'instruction INT normale mais dans ce cas l'appel ne dépendra pas de l'état d'armement du bit Overflow. En fait cette interruption est très peu utilisée et DOS la dirige directement sur une instruction de retour IRET.

Interruption 5	Copie d'écran

Bien que classée dans le groupe des interruptions matérielles, l'interruption 5 est exploitée par le BIOS. Elle est déclenchée par le BIOS chaque fois que la touche "PrtScr" (sur un clavier américain) ou "Impr" (sur un clavier français) est actionnée. Sa tâche consiste à envoyer sur l'imprimante le contenu de l'écran, ce qu'on appelle une copie

d'écran ou hard copy. C'est pourquoi le vecteur de cette interruption est initialisé par le BIOS dans la table des vecteurs d'interruption de façon à pointer sur la routine de copie d'écran du BIOS qui figure en ROM . Les programmes en assembleur ou en langage évolué peuvent également recourir à cette interruption, en l'appelant par INT, pour envoyer au cours de leur exécution une copie d'écran sur l'imprimante.

Interruptions 6,7	non utilisées

Ces interruptions sont pour l'instant réservées à un usage ultérieur par IBM. Elles peuvent donc être utilisées pour des besoins de programmation individuels.

Interruption 8	Timer

Le timer du PC (un circuit 8253) reçoit 1 193 180 signaux par seconde du coeur du système constitué par un oscillateur à quartz. Au bout de 65536 de ces signaux, soit environ 18,2 fois par seconde, il génère un appel à l'interruption 8 qui est transmis au processeur par l'intermédiaire du 8259 . Comme la fréquence d'appel de cette interruption ne dépend pas de la fréquence d'horloge (variable) du système, cette interruption convient parfaitement à la mesure du temps puisqu'on sait qu'une seconde s'écoule tous les 18,2 appels. C'est dans cet esprit que l'interruption est exploitée dans un PC. Le BIOS dirige le vecteur de l'interruption sur une routine qui est donc appelée 18,2 fois par seconde. Cette routine est chargée de faire avancer un compteur de temps et d'arrêter le moteur du lecteur de disquette si aucun accès à la disquette ne s'est produit depuis un certain délai. Une fois ces tâches exécutées, la routine appelle l'interruption 1Ch qui peut être détournée par un utilisateur qui aurait besoin d'une exploitation dépendant d'un signal périodique.

Interruption 9	Clavier

Le clavier comporte un processeur Intel portant l'appellation 8048 (ou 8042 sur l'AT). Ce processeur surveille le clavier et enregistre toute pression ou tout relâchement d'une touche, ou encore tout enfoncement prolongé . Il doit renseigner le processeur central pour que le système puisse réagir de façon appropriée en lisant le code de la touche frappée. Il envoie donc un signal au 8259 qui, lorsqu'aucune interruption de priorité supérieure n'est à prendre en compte, demande au processeur central de déclencher l'interruption 9. Cette interruption appelle alors une routine du BIOS qui lit le caractère frappé au clavier et le place le buffer du clavier.

Interruptions 10 à 12	Dépendent des périphériques rattachés

Interruption 13h	Disque dur

Lorsqu'un disque dur est connecté au système, il fait appel à l'interruption 13h. Ce sera par exemple le cas lorsqu'une opération de lecture ou d'écriture sera terminée et que le BIOS doit en être averti.

Interruption 14	Disquette

Cette interruption est appelée par le contrôleur des lecteurs de disquette en coordination avec le 8259 lorsque l'attention du processeur doit être attirée. Derrière cette interruption se dissimule une routine du BIOS qui communique avec le contrôleur de disquette au niveau le plus bas. Lorsqu'elle est appelée, le contrôleur lui transmet des informations d'état destinées à informer le BIOS qu'une opération de lecture ou d'enregistrement est terminée.

Interruption 15	Imprimante

Cette interruption est déclenchée par une imprimante connectée en parallèle en coordination avec le 8259 lorsque cette imprimante désire attirer l'attention du processeur.

Grâce à son deuxième contrôleur d'interruption, l'AT dispose de plus d'interruptions que ses prédecesseurs. Les huit interruptions numérotées 70h à 77h deviennent ainsi des interruptions matérielles parfaitement accessibles, alors qu'elles étaient antérieurement mises à la disposition des programmes d'application, une liberté désormais révoquée.

Selon le même principe que pour les huit périphériques reliés au premier contrôleur d'interruption, le périphérique correspondant au bit 0 du registre de masque d'interruption du deuxième contrôleur déclenche l'interruption 70h. Le périphérique associé au bit 1 déclenche l'interruption 71h, et ainsi de suite.

En réalité, seules les interruptions 70h et 75h sont appelées par le contrôleur car seuls les bits 0 et 5 du registre de masque d'interruption sont reliés à des périphériques. Mais les vecteurs des interruptions 71h à 74h, ainsi que les numéros 75h et 77h ne doivent pas être oubliés pour autant.

Interruption 70h — Horloge en temps réel

Une horloge en temps réel a beaucoup de raisons d'interrompre un programme en cours d'exécution. Il peut s'agir d'une alarme au bout d'un temps déterminé, de la mise à jour de la date et de l'heure, ou simplement d'un appel cyclique à l'initiative d'un programme. Cette interruption est normalement desservie par une routine du BIOS qui détermine d'abord la cause de l'appel puis réagit en conséquence.

Interruption 75h — Coprocesseur arithmétique

L'interruption 75h permet au coprocesseur arithmétique (80287) de requérir l'attention du processeur central, par exemple en raison d'une erreur.

Interruption 76h — Disque dur de l'AT

Cette interruption est déclenchée par le contrôleur de disque dur de l'AT lorsqu'il a terminé un accès au disque.

J. Systèmes numériques

Au cours de cet ouvrage, nous avons souvent rencontré des nombres qui n'étaient pas exprimés dans le système décimal mais en binaire ou hexadécimal. Si vous n'êtes pas encore très familiarisés avec ces deux systèmes numériques, ce chapitre a pour but de vous fournir une petite introduction au problème des systèmes numériques.

Avant de décrire ces deux systèmes numériques nouveaux pour vous, il nous faut examiner précisément les bases de notre système décimal.

Lorsque vous écrivez par exemple le nombre décimal 1988, ce nombre pourrait aussi être formulé de la façon suivante : 1*1000+9*100+8*10+8*1. Cela vous montre déjà que si nous numérotons les chiffres de droite à gauche, le premier chiffre (celui des unités) est associé à un facteur 1, le second (les dizaines) à un facteur 10, le troisième (les centaines) à un facteur 100 et le dernier enfin (les milliers) à un facteur 1000. Vous voyez donc que le prochain chiffre sur la gauche est toujours associé à un facteur 10 fois plus élevé.

Tous les systèmes numériques ont ceci en commun que le premier chiffre (en partant de la droite) est toujours associé à un facteur de 1. La progression de ce facteur de chiffre en chiffre est par contre différente pour chaque système numérique, dont elle est la caractéristique. Cette progression est un facteur égal au nombre de chiffres utilisés par le système. Pour le système décimal, ce facteur de progression est de 10 puisqu'il y a dix chiffres (0 à 9).

Essayons maintenant de préciser ce principe en décrivant les systèmes numériques binaire et hexadécimal que nous avons déjà évoqués.

Le système binaire est le système de base de votre ordinateur car, comme vous le savez, l'électronique d'un ordinateur ne distingue au fond que deux états, et donc deux nombres, 0 et 1. Comme nous ne disposons plus dans ce système des chiffres 0 à 9 mais seulement des deux chiffres 0 et 1, le facteur associé à chaque chiffre ne sera plus décuplé mais seulement doublé en progressant vers la gauche.

Concrètement, le premier chiffre sera donc associé à un facteur 1, le second à un facteur 2, le troisième à un facteur 4 et le quatrième à un facteur 8. Après 8, viennent 16, 32, 64, 128 et ainsi de suite.

Le nombre binaire 11001 correspondra donc en système décimal à 1*16+1*8+0*4+0*2+1*1, soit 25.

Contrairement au système binaire, le système hexadécimal fonctionne avec plus de chiffres que le système décimal. Dans ce système, on ne compte pas de 0 à 9, encore moins de 0 à 1, mais de 0 à F. La base de ce système est en effet 16 et il nous faut donc 16 chiffres. Comme les chiffres arabes ne vont que jusqu'à 9, on a recours aux lettres A à F pour représenter les valeurs 10 à 15.

A représente donc 10, B 11, C 12, D 13, E 14 et enfin F 15.

Comme nous l'avons indiqué, le facteur de progression d'un chiffre à l'autre n'est ici ni 2 ni 10 mais 16.

Le premier chiffre est toujours associé à un facteur 1, mais le second est associé au facteur 16, le troisième au facteur 256 et le quatrième au facteur 4096.

Prenons un exemple pour illustrer ce principe : le nombre hexadécimal FB3 correspond en décimal à 15*256+11*16+3*1, soit 4019.

Le système hexadécimal est toutefois en relation étroite avec le système binaire puisque chaque chiffre d'un nombre hexadécimal peut être exprimé par un groupe de 4 chiffres binaires. C'est d'ailleurs la raison qui explique le rôle si important du système hexadécimal pour la programmation en assembleur. Il constitue en effet une sorte de pont avec le système binaire, système de base de l'ordinateur mais illisible pour les humains. C'est ainsi, par exemple, qu'un octet (une valeur codée sur 8 bits) pourra être représentée très simplement à l'aide de deux chiffres hexadécimaux.

Pour éviter tout risque de confusion entre les divers systèmes numériques, les nombres binaires sont désignés dans cet ouvrage par b et les nombres hexadécimaux par h.

Si vous n'avez pas complètement assimilé certaines des explications de ce chapitre, les figures suivantes vous aideront.

Position	5	4	3	2	1
Décimal :	10000	1000	100	10	1
Binaire :	16	8	4	2	1
Hexadecimal :	65536	4096	256	16	1

Table d'équivalence entre les trois systèmes numériques pour quelques valeurs choisies

Comparaison des sytèmes numériques

DECIMAL	BINAIRE	HEXADECIMAL
0	0	0
1	1	1
2	10	2
3	11	3
4	100	4
5	101	5
6	110	6
7	111	7
8	1000	8
9	1001	9
10	1010	A
11	1011	B
12	1100	C
128	10000000	80
129	10000001	81
256	100000000	100
1024	10000000000	400
4096	1000000000000	1000
65535	1111111111111111	FFFF

Facteur associé aux différents chiffres d'un nombre dans les trois systèmes numériques les plus utilisés en informatique

K. Index des programmes

L. La disquette du livre

Cet ouvrage est accompagné d'une disquette 3,5" 720 Ko ou de deux disquettes 5,25" 360 Ko. Elles contiennent sous forme compactée tous les fichiers source des exemples importants cités ou étudiés au fil des pages.

Vous trouverez sur la disquette 3,5" ou les disquettes 5,25" quatre fichiers EXE auto-décompactables nommés :

PAS.EXE	Fichiers Pascal
C.EXE	Fichiers C
ASM.EXE	Fichiers Assembleur
BAS.EXE	Fichiers Basic

Ces quatre fichiers ont été compactés avec le programme de compactage LHA (Copyright (C) Haruyasu Yoshizaki, 1988-91).

Pour installer et décompacter ces groupes de fichiers sur votre disque dur (par exemple C:), vous pouvez suivre les étapes décrites ci-dessous :

① Créer un répertoire de travail sur votre disque dur, par exemple C:\BIBLE, en entrant les commandes suivantes :

```
C:\
MD BIBLE
CD BIBLE
```

② Activez le lecteur de disquette dans lequel vous avez placé la ou les disquettes d'accompagnement fournies avec cet ouvrage :

```
A: ou
B:
```

③ Lorsque vous exécutez l'un des fichiers EXE fournis sur la disquette, les fichiers source correspondants sont automatiquement extraits et placés dans un sous-répertoire approprié.

Par exemple, pour décompacter tous les exemples écrits en Pascal, entrez :

`A:`	Active le lecteur A
`PAS.EXE C:\BIBLE`	Extrait de PAS.EXE tous les fichiers nécessaires aux programmes Pascal (PAS, ASM, OBJ) et les place dans le répertoire C:\BIBLE\PAS

De même :

`C.EXE C:\BIBLE`	Extrait de C.EXE tous les fichiers nécessaires aux programmes C (C, ASM, OBJ) et les place dans le répertoire C:\BIBLE\C
`BAS.EXE C:\BIBLE`	Extrait de BAS.EXE tous les fichiers Basic et les place dans le répertoire C:\BIBLE\BAS
`ASM.EXE C:\BIBLE`	Extrait de ASM.EXE tous les fichiers Assembleur et les place dans le répertoire C:\BIBLE\ASM

M. Bibliographie

Abrash, Michael	Power Graphics Programming Que 1989
Althaus, Martin	PC Profibuch, Extended Edition Sybex 1990
Angermeyer, John/Jaeger, Kevin:	MS-DOS Developer's Guide The Waite Group 1986
Byres, T.J.	IBM PC/AT McGraw Hill 1986
Crawford, John H.	Programming the 80386 Sybex 1987
DPMI Committee	DOS Protected Mode Interface (DPMI) Specification Intel 1991
Duncan, Ray (Hrsg.)	Extending DOS Addison Wesley, 1990
Duncan, Ray	Advanced MS-DOS Microsoft Press 1986
IBM	Technical Reference Manual AT IBM 1984
IBM	Technical Reference Manual PC IBM 1980
IBM	Technical Reference Manual XT IBM 1981
Kernighan, B.W./Ritchie, D.M.	Programmieren in C Hanser 1983
Michael, Manfred	Enhanced Graphics Adapter Markt & Technik 1987
Microsoft	Mouse Programmer's Reference Microsoft Press 1991

Microsoft	MS-DOS Programmer's Reference Microsoft Press 1991
Murray, Pappas	80386/80286 Assembly Language Programming Mc Graw-Hill 1986
Nance, Barry	Network-Programming in C Que 1990
Norton, Peter	Programmierhandbuch für den IBM PC Vieweg 1986
Phoenix Technologies	System BIOS for IBM PC/XT/AT Computers and Compatibles Addison Wesley 1989
Rosch, Winn L.	The Winn Rosch Hardware Bible Brady 1989
Sargent/Schumaker/Stelzer	Assemblersprache und Hardware des IBM PC Addison/Wesley 1984
Schäpers, Arne	DOS 5 für Programmierer Addison Wesley 1991
Schulman, Andrew (Hrsg.)	DOS undocumented Addison Wesley, 1990
Schwaderer, David W.	C Programmer's Guide to NetBIOS Howard W.Sams 1988
Sutty, George	Advanced Programmer's Guide to SuperVGAs Brady 1990
VCPI Committee	Virtual Control Program Interface Phar Lap Software Inc. 1989
Wilton, Richard	Programmer's Guide to PC & PS/2 Video-Systems Micorsoft Press 1987
Young, Michael J.	Performance Programming under MS-DOS Sybex 1987

N. Table des caractères ASCII

Décimal	Hexadécimal	Caractère	Décimal	Hexadécimal	Caractère	Décimal	Hexadécimal	Caractère	Décimal	Hexadécimal	Caractère
0	00		32	20		64	40	@	96	60	`
1	01	☺	33	21	!	65	41	A	97	61	a
2	02	☻	34	22	"	66	42	B	98	62	b
3	03	♥	35	23	#	67	43	C	99	63	c
4	04	♦	36	24	$	68	44	D	100	64	d
5	05	♣	37	25	%	69	45	E	101	65	e
6	06	♠	38	26	&	70	46	F	102	66	f
7	07	•	39	27	'	71	47	G	103	67	g
8	08	◘	40	28	(	72	48	H	104	68	h
9	09	○	41	29	)	73	49	I	105	69	i
10	0A	◙	42	2A	*	74	4A	J	106	6A	j
11	0B	♂	43	2B	+	75	4B	K	107	6B	k
12	0C	♀	44	2C	,	76	4C	L	108	6C	l
13	0D	♪	45	2D	-	77	4D	M	109	6D	m
14	0E	♫	46	2E	.	78	4E	N	110	6E	n
15	0F	☼	47	2F	/	79	4F	O	111	6F	o
16	10	►	48	30	0	80	50	P	112	70	p
17	11	◄	49	31	1	81	51	Q	113	71	q
18	12	↕	50	32	2	82	52	R	114	72	r
19	13	‼	51	33	3	83	53	S	115	73	s
20	14	¶	52	34	4	84	54	T	116	74	t
21	15	§	53	35	5	85	55	U	117	75	u
22	16	▬	54	36	6	86	56	V	118	76	v
23	17	↨	55	37	7	87	57	W	119	77	w
24	18	↑	56	38	8	88	58	X	120	78	x
25	19	↓	57	39	9	89	59	Y	121	79	y
26	1A	→	58	3A	:	90	5A	Z	122	7A	z
27	1B	←	59	3B	;	91	5B	[	123	7B	{
28	1C	∟	60	3C	<	92	5C	\	124	7C	|
29	1D	↔	61	3D	=	93	5D	]	125	7D	}
30	1E	▲	62	3E	>	94	5E	^	126	7E	~
31	1F	▼	63	3F	?	95	5F	_	127	7F	⌂

Décimal	Hexadécimal	Caractère	Décimal	Hexadécimal	Caractère	Décimal	Hexadécimal	Caractère	Décimal	Hexadécimal	Caractère
128	80	Ç	160	A0	á	192	C0	└	224	E0	α
129	81	ü	161	A1	í	193	C1	┴	225	E1	ß
130	82	é	162	A2	ó	194	C2	┬	226	E2	Γ
131	83	â	163	A3	ú	195	C3	├	227	E3	π
132	84	ä	164	A4	ñ	196	C4	─	228	E4	Σ
133	85	à	165	A5	Ñ	197	C5	┼	229	E5	σ
134	86	å	166	A6	ª	198	C6	╞	230	E6	µ
135	87	ç	167	A7	º	199	C7	╟	231	E7	τ
136	88	ê	168	A8	¿	200	C8	╚	232	E8	Φ
137	89	ë	169	A9	⌐	201	C9	╔	233	E9	Θ
138	8A	è	170	AA	¬	202	CA	╩	234	EA	Ω
139	8B	ï	171	AB	½	203	CB	╦	235	EB	δ
140	8C	î	172	AC	¼	204	CC	╠	236	EC	∞
141	8D	ì	173	AD	¡	205	CD	═	237	ED	ø
142	8E	Ä	174	AE	«	206	CE	╬	238	EE	∈
143	8F	Å	175	AF	»	207	CF	╧	239	EF	∩
144	90	É	176	B0	▒	208	D0	╨	240	F0	≡
145	91	æ	177	B1	▓	209	D1	╤	241	F1	±
146	92	Æ	178	B2	▓	210	D2	╥	242	F2	≥
147	93	ô	179	B3	│	211	D3	╙	243	F3	≤
148	94	ö	180	B4	┤	212	D4	╘	244	F4	⌠
149	95	ò	181	B5	╡	213	D5	╒	245	F5	⌡
150	96	û	182	B6	╢	214	D6	╓	246	F6	÷
151	97	ù	183	B7	╖	215	D7	╫	247	F7	≈
152	98	ÿ	184	B8	╕	216	D8	╪	248	F8	°
153	99	Ö	185	B9	╣	217	D9	┘	249	F9	∙
154	9A	Ü	186	BA	║	218	DA	┌	250	FA	·
155	9B	¢	187	BB	╗	219	DB	█	251	FB	√
156	9C	£	188	BC	╝	220	DC	▄	252	FC	n
157	9D	¥	189	BD	╜	221	DD	▌	253	FD	z
158	9E	₧	190	BE	╛	222	DE	▐	254	FE	■
159	9F	ƒ	191	BF	┐	223	DF	▀	255	FF	

Index

!

A

B

C

D

E

F

G

H

I

J

L

M

N

O

P

Q

R

S

T

U

V

Z