MS-DOS, PC-BIOS et E/S de Fichiers Chapitre 13

Un système PC classique se compose de beaucoup de composants à côté du CPU 80x86 et de la mémoire. MS-DOS et le BIOS du PC fournissent une interface logicielle entre votre programme d'application et le matériel sous-jacent. Bien qu'il soit parfois nécessaire de programmer le matériel directement vous-même, le plus souvent il vaut mieux laisser le logiciel système (MS-DOS et le BIOS) le manipuler ceci à votre place. En outre, il vous est beaucoup plus facile d'appeler simplement une routine intégrée dans votre système que d'écrire la routine vous-même.

Vous pouvez accéder au matériel système du PC IBM à un de trois niveaux généraux depuis l'assembleur. Vous pouvez programmer le matériel directement, vous pouvez employer des routines de la ROM BIOS pour accéder au matériel à votre place, ou vous pouvez faire appel à MS-DOS pour accéder au matériel. Chaque niveau d'accès au système a son propre ensemble d'avantages et d'inconvénients.

La programmation directe du matériel offre deux avantages sur les autres options: contrôle et efficacité. Si vous contrôlez les modes matériels, vous pouvez tirer cette dernière goutte de performance du système en tirant profit de trucs spéciaux sur le matériel ou d'autres détails qu'une routine généraliste ne peut pas utiliser. Pour quelques programmes, comme les éditeurs d'écran (qui doivent avoir accès à grande vitesse à l'affichage vidéo), l'accès direct au matériel est la seule manière de réaliser des niveaux de performances raisonnables.

D'autre part, la programmation directe du matériel a aussi ses inconvénients. L'éditeur d'écran qui accède directement à la mémoire vidéo peut ne pas fonctionner si un nouveau type de carte graphique est installé. De multiples drivers d'affichage peuvent être nécessaires pour un tel programme, augmentant la quantité de travail pour créer et maintenir le programme. En outre, si vous avez écrit des programmes qui accèdent à la mémoire vidéo directement et que sortent de nouveaux adaptateurs vidéo, incompatibles, vous devrez récrire tous vos programmes pour travailler avec la nouvelle carte graphique.

Votre charge de travail serait réduite énormément si les fabricants de cartes graphiques fournissait, dans un endroit fixe, connu, quelques routines qui faisaient toutes les opérations d'E/S de l'écran à votre place. Vos programmes appelleraient toutes ces routines. Quand un fabricant présente un nouvel adapteur graphique, il fournit un nouvel ensemble de routines d'affichage vidéo avec la carte graphique. Ces nouvelles routines patcheraient les anciennes (les remplaçant ou les augmentant) de sorte que les appels aux vieilles routines appelleraient maintenant les nouvelles routines. Si l'interface du programme est la même entre les deux ensembles de routines, vos programmes marcheraient avec les nouvelles routines.

IBM a mis en application un tel mécanisme dans le firmware système du PC. Tout en haut de l'espace mémoire de un méga-octet du PC se trouvent quelques adresses consacrées au stockage de données de la ROM. Ces cellules de mémoire ROM contiennent le logiciel spécial appelé Basic Input Outputt System du PC, ou BIOS. Les routines du BIOS fournissent une interface indépendante du matériel pour les divers périphériques dans le système du PC d'IBM. Par exemple, un des services du BIOS est un pilote d'affichage vidéo. En faisant divers appels aux routines vidéo du BIOS, votre logiciel pourra écrire des caractères à l'écran indépendamment de la carte graphique réelle installée.

Au niveau supérieur, se trouve MS-DOS. Tandis que le BIOS vous permet de manoeuvrer des périphériques à un niveau très bas, MS-DOS fournit une interface de niveau élevé à beaucoup de périphériques. Par exemple, une des routines du BIOS vous permet d'accéder au lecteur de disquettes. Avec cette routine du BIOS vous pouvez lire ou écrire des blocs sur la disquette. Malheureusement, le BIOS ne connaît rien de choses comme les fichiers et les répertoires. Il connait seulement des blocs. Si vous voulez accéder à un fichier sur le lecteur de disquettes en utilisant un appel du BIOS, vous devrez savoir exactement où ce fichier apparaît sur la surface de la disquette. Par contre, les appels à MS-DOS vous permettent de traiter des noms de fichier plutôt que des adresses de fichier sur le disque. MS-DOS garde une trace de l'endroit où les fichiers se trouvent sur la face du disque et fait appel à la ROM BIOS pour lire les blocs appropriés. Cette interface à niveau élevé réduit considérablement le travail que votre logiciel doit fournir afin d'accéder à des données sur l'unité de disques.

Le but de ce chapitre est de fournir une brève introduction aux divers services du BIOS et du DOS disponibles pour vous. Ce chapitre n'essaye pas de commencer à décrire toutes les routines ou options disponibles de chaque routine. Il existe plusieurs autres textes de la taille de celui-ci qui essayent de discuter *uniquement* le BIOS ou *uniquement* MS-DOS. En outre, essayer de couvrir complètement MS-DOS ou le BIOS dans un texte unique est voué à l'échec dès le début - tous les deux forment une cible mobile avec des caractéristiques changeant avec chaque nouvelle version. Aussi plutôt que d'essayer d'expliquer tout, ce chapitre essayera simplement d'en donner un aperçu. Voyez la bibliographie pour des textes ayant affaire directement avec le BIOS ou MS-DOS.

13.0 Vue d'ensemble du chapitre

Ce chapitre présente des notions qui sont spécifiques au PC. Cette information sur le BIOS du PC et MS-DOS n'est pas nécessaire pour apprendre la programmation en assembleur; cependant, c'est une information importante pour écrire des programmes en assembleur qui fonctionnent sous MS-DOS sur une machine compatible PC. En conséquence, la majeure partie de l'information de ce chapitre est facultative pour ceux qui veulent apprendre la programmation générique en assembleur 80x86. D'autre part, cette information est utile pour ceux qui veulent écrire des applications en assembleur sur un PC.

Les sections ci-dessous qui ont un préfixe "●" sont essentielles. Les sections avec un "❑" discutent de matières avançées que vous pouvez remettre à plus tard.

- Le BIOS du PC IBM
- ❑ Impression d'écran.
- Services vidéo.
- ❑ Équipement installé.
- ❑ Mémoire disponible.
- ❑ Service de disque de bas niveau.
- E/S série.
- ❑ Services divers.
- Services de clavier.
- Services d'imprimante.
- ❑ Exécution du BASIC.
- ❑ Réinitialisation de l'ordinateur.
- ❑ Horloge temps réel .
- Séquence d'appel de MS-DOS.
- Fonctions de caractères de MS-DOS
- ❑ Commande de lecteur de MS-DOS.
- ❑ Fonctions de date et d'heure de MS-DOS.
- ❑ Fonctions de gestion de la mémoire de MS-DOS.
- ❑ Fonctions de contrôle du programme de MS-DOS.
- "Nouveaux" appels de fichiers de MS_DOS.
- Ouvrir un fichier .
- Créer un fichier.
- Fermer un fichier .
- Lire à partir d'un fichier.
- Écrire sur un fichier.
- ❑ Rechercher.
- ❑ Positionner l'adresse de transfert de disque.
- ❑ Trouver le premier fichier.
- ❑ Trouver le fichier suivant.
- Détruire un fichier.
- Renommer un fichier.
- ❑ Changer/obtenir les attributs d'un fichier.
- ❑ Changer/ obtenir la date et l'heure d'un fichier.
- ❑ Autres appels DOS
- Exemples d'E/S de fichier.
- E/S de fichier en bloc.
- ❑ Le préfixe de segment de programme.
- ❑ L'accès aux paramètres de la ligne de commande.
- ❑ ARGC et ARGV.
- Les routines d'E/S de fichier de l'UCR Standard Library .
- FOPEN.
- FCREATE.
- FCLOSE.
- FFLUSH.
- FGETC.

- FREAD.
- FPUTC.
- FWRITE.

❑ Redirection des E/S par les routines d'E/S de fichier de la STDLIB.

13.1 Le BIOS du PC d'IBM

Plutôt que placer les routines du BIOS à des emplacement fixes de la mémoire dans la ROM, IBM a employé une approche beaucoup plus souple dans la conception du BIOS. Pour appeler une routine du BIOS, vous employez une des instructions d'interruption logicielle **int** du 80x86. L'instruction **int** emploie la syntaxe suivante:

int *valeur*

Valeur est un nombre dans la plage 0..255. L'exécution de l'instruction **int** demandera au 80x86 de transférer le contrôle à un des 256 gestionnaires d'interruption différents. La table des vecteur d'interruption, commençant à l'emplacement de la mémoire physique 0:0, contient les adresses de ces gestionnaires d'interruption. Chaque adresse est une adresse segmentée complète, occupant quatre bytes, si bien qu'il y a 400h bytes dans la table des vecteur d'interruption -- une adresse segmentée pour chacune des 256 interruptions logicielles possibles. Par exemple, int 0 transfère le contrôle à la routine dont l'adresse est à l'emplacement 0:0, int 1 transfère le contrôle à la routine dont l'adresse est à 0:4, int 2 via 0:8, int 3 via 0:C, et int 4 via 0:10.

Quand le PC redémarre, une des premières opérations qu'il fait, est d'initialiser plusieurs de ces vecteurs d'interruption pour qu'ils pointent sur des routines de service du BIOS. Par la suite, quand vous exécutez une instruction **int** appropriée, le contrôle est transféré au code approprié du BIOS.

Si vous vous contentez d'appeler des routines du BIOS (par opposition à les écrire), vous pouvez considérer l'instruction **int** comme rien d'autre qu'une instruction **call** spéciale.

13.2 Une Introduction aux Services du BIOS

Le BIOS du PC d'IBM utilise les interruptions logicielles 5 et 10h..1Ah pour accomplir diverses opérations. Par conséquent, les instructions **int 5**, et **int 10h.. int 1ah** fournissent l'interface au BIOS. La table suivante récapitule les services de BIOS:

INT	Fonction
5	Opération d'Impression d'écran.
10h	Services d'affichage vidéo.
11h	Détermination de l'équipement.
12h	Détermination de la capacité de la mémoire.
13h	Services de disquette et de disque dur.
14h	Services d'E/S série.
15h	Services divers.
16h	Services du clavier.
17h	Services de l'imprimante.
18h	BASIC.
19h	Réinitialisation.
1Ah	Services de l'horloge temps réel.

La plupart de ces routines exigent divers paramètres dans les registres du 80x86. Certains exigent des paramètres additionnels dans certains emplacements de mémoire. Les sections suivantes décrivent l'opération exacte de nombreuses routines du BIOS.

13.2. INT 5- Impression d'écran

Instruction: int 5h
Opération du BIOS: Impression de l'écran de texte courant.
Paramètres: Aucun

Si vous exécutez l'instruction **int 5h**, le PC enverra une copie de l'image d'écran à l'imprimante exactement comme si vous aviez appuyé sur la touche PrtSc du clavier. En fait, le BIOS émet une instruction **int 5** quand vous appuyez sur PrtSc, ainsi les deux opérations sont absolument identiques (sinon que l''une est sous contrôle logiciel et l'autre sous contrôle manuel). Notez que les 80286 et plus récents emploient également **int 5** pour la capture de BOUNDS.

13.2.2 INT 10h- Services Vidéo

Instruction: int 10h
Opération du BIOS: Services d'E/S Vidéo.
Paramètres: Plusieurs, passés dans les registres **ax**, **bx**, **cx**, **dx** et **es:bp**.

L'instruction int 10h exécute plusieurs fonctions en relation avec l'affichage vidéo. Vous pouvez l'employer pour initialiser l'affichage vidéo, déterminer la taille du curseur et sa position, lire la position du curseur, manipuler un crayon lumineux, lire ou écrire la page courante d'affichage, faire défiler les données dans l'écran vers le haut ou vers le bas, lire et écrire des caractères, lire et écrire des pixels dans un mode d'affichage graphique, et écrire des chaînes à l'écran. Vous choisissez la fonction particulière à exécuter en passant une valeur dans le registre **ah**.

Les services vidéos représentent un des plus grand ensembles d'appels du BIOS disponibles. Il y a beaucoup de cartes d'affichage vidéo différentes construites pour les PCs, chacune avec des variations mineures et souvent chacune avec son propre ensemble de fonctions de BIOS uniques. La référence du BIOS dans les annexes énumère certaines des fonctions plus communes disponibles, mais comme précisé précédemment, cette liste est tout à fait incomplète et démodée vu l'évolution rapide de la technologie.

Le service vidéo le plus généralement utilisé est probablement la routine d'affichage de caractère:

Nom: Écrire un caractère à l'écran en mode téléscripteur
Paramètres ah = 0Eh, al = code ASCII (en mode graphique, bl = numéro de page)

Cette routine écrit un caractère unique à l'écran. MS-DOS appelle cette routine pour afficher des caractères sur l'écran. La Bibliothèque Standard UCR fournit également un appel qui vous permet d'écrire des caractères directement à l'affichage en utilisant des appels au BIOS.

La plupart des routines d'affichage vidéo du BIOS sont mal écrites. Il n'y a pas grand chose d'autre à dire à leur sujet. Elles sont extrêmement lentes et ne fournissent pas beaucoup fonctionnalités. Pour cette raison, la plupart des programmeurs (qui ont besoin d'un pilote d'affichage vidéo à haut rendement) finissent par écrire leur propre code d'affichage. Ceci fournit la vitesse aux dépens de la portabilité. Malheureusement, il y a rarement d'autre choix. Si vous avez besoin de fonctionnalité plutôt que de vitesse, vous devriez envisager d'employer le pilote d'affichage ANSI.SYS fourni avec MS-DOS. Ce pilote d'affichage fournit toutes sortes de services utiles tels que l'effacement jusqu'à la fin de la ligne, l'effacement jusqu'à la fin de l'écran, etc... Pour plus d'information, consultez votre manuel DOS.

Tableau 49: Fonctions Vidéo du BIOS (Liste Partielle)

AH	Paramètres d' Entrée	Paramètres de Sortie	Description
0	al = mode		Détermine le mode d'affichage vidéo
1	ch =ligne de départ cl =ligne de de fin		Détermine la forme du curseur. Les valeurs ligne sont dans la plage 0..15. Vous pouvez faire disparaître le curseur en chargeant **ch** avec 20h.
2	bh = page dl = x dh = y		Positionne le curseur à l'emplacement (x,y) sur l'écran. Généralement, vous indiquerez la page zéro. Le BIOS maintient un curseur séparé pour chaque page.
3	bh = page	ch =ligne de départ cl =ligne de fin dl =coordonnée x dh = coordonnée y	Obtient la position et la forme du curseur.
4			Obsolète (Obtient la position du Crayon Lumineux).
5	al =page d'affichage		Détermine la page d'affichage. Commute la page d'affichage du texte au numéro de page

			indiqué. La page zéro est la page standard de texte. La plupart des adapteurs couleur supportent jusqu'à huit pages de texte (0..7).
6	al = nombre de lignes à faire défiler bh = attribut d'écran pour la zone effacée cl = coordonnée x Haut/Gauche ch = coordonnée y Haut/Gauche dl = coordonnée x Bas/Droite dh = coordonnée y Bas/Droite		Effacer ou faire défiler vers le haut. Si **al** contient zéro, cette fonction efface la partie rectangulaire de l'écran spécifiée par **cl/ch** (le coin supérieur gauche) et **dl/dh** (le coin inférieur droit). Si **al** contient n'importe quelle autre valeur, ce service fera défiler cette fenêtre rectangulaire vers le haut du nombre de lignes spécifié dans **al**.
7	al = nombre de lignes à faire défiler bh = attribut d'écran pour la zone effacée cl = coordonnée x Haut/Gauche ch = coordonnée y Haut/Gauche dl = coordonnée x Bas/Droite dh = coordonnée y Bas/Droite		Effacer ou faire défiler vers le bas. Si **al** contient zéro, cette fonction efface la partie rectangulaire de l'écran spécifiée par **cl/ch** (le coin supérieur gauche) et **dl/dh** (le coin inférieur droit). Si **al** contient n'importe quelle autre valeur, ce service fera défiler cette fenêtre rectangulaire vers le bas du nombre de lignes spécifié dans **al**.
8	bh =page d'affichage	al =caractère lu ah =attribut du caractère	Lit le code ASCII du caractère et son byte d'attribut depuis la position actuelle à l'écran.
9	al = caractère bh = page bl = attribut cx = # de repétitions du caractère		Cet appel écrit **cx** copies du caractère et de l'attribut dans **al/bl** en commençant à la position actuelle du curseur sur l'écran. Il ne change pas la position du curseur.
0Ah	al = caractère bh = page		Écrit le caractère dans **al** à la position actuelle d'écran en utilisant l'attribut existant. Ne change pas la position du curseur.
0Bh	bh = 0 bl = couleur		Détermine la couleur de la bordure pour l'affichage des textes.
0Eh	al = caractère bh = page		Écrit un caractère à l'écran. Utilise l'attribut existant et replace le curseur après écriture.

0Fh		ah = # colonnes
		al = mode d'affichage
		bh = page
			Obtient le mode vidéo

Notez qu'il y a beaucoup d'autres sous-functions de l'INT BIOS 10h. La plupart du temps, ces autres fonctions traitent des modes de graphiques (le BIOS est trop lent pour manipuler les graphiques, aussi vous ne devriez pas utiliser ces appels) et des fonctions étendues pour certaines cartes d'affichage vidéo. Plus d'information sur ces derniers appels, prenez un texte sur le BIOS du PC.

13.2.3 INT 11h- Matériel Installé

Instruction: int 11h
Opération du BIOS: Renvoie une liste de matériel.
Paramètres: En entrée: Aucun, en sortie: AX contient la liste de matériel

Au retour de int 11h, le registre AX contient une liste de matériel codée par bit avec les valeurs suivantes:

Bit 0 Unité de disquettes installée
Bit 1 Coprocessor mathématique installé
Bits 2, 3 RAM de carte système installée (obsolète)
Bits 4, 5 Mode vidéo initial
- 00- aucun
- 01- 40x25 couleur
- 10- 80x25 couleur
- 11- 80x25 n/b

Bits 6, 7	Nombre de disques durs
Bit 8	DMA présente
Bits 9, 10, 11	Nombre de cartes RS-232 série installées
Bit 12	Carte d'E/S de jeu installée
Bit 13	Imprimante série reliée
Bits 14, 15	Nombre d'imprimantes reliées.

Notez que ce service BIOS a été conçu autour du PC d'IBM original avec des possibilités très limitées d'expansion de matériel. Les bits retournés par cet appel sont presque sans signification aujourd'hui.

13.2.4 INT 12h- Mémoire Disponible

Instruction:	int 12h
Opération du BIOS:	Détermine la taille de la mémoire.
Paramètres:	Taille de la mémoire retournée dans AX

A l'époque où les PCS d'IBM sortaient avec jusqu'à 64K de mémoire installée sur la carte mère, cet appel avait quelque intérêt. Cependant, les PCs d'aujourd'hui peuvent manipuler jusqu'à 64 méga-octets ou plus. Évidemment cet appel du BIOS est un peu démodé. Quelques PCs emploient cet appel pour des buts différents, mais vous ne pouvez pas espérer que de tels appels fonctionnent sur toutes les machines.

13.2.5 INT 13h- Services de Disque de Bas Niveau

Instruction:	int 13h
Opération du BIOS:	Services de disquette.
Paramètres:	**ax**, **es:bx, cx**, **dx** (voir ci-dessous)

La fonction int 13h fournit plusieurs services de disque de bas niveau aux programmes du PC: Réinitialiser le système de la disquette, obtenir le statut de la disquette, lire les secteurs de la disquette, écrire sur les secteurs de la disquette, vérifier les secteurs de la disquette et formater une piste et d'autres encore. C'est un autre exemple d'une routine BIOS qui a changé au cours des années. Quand cette routine a été développée la première fois, un disque dur de 10 méga-octets a été considéré grand. Aujourd'hui, un jeu aux performances élevées exige 20 à 30 méga-octets de stockage.

Tableau 50: Quelques appels communs du BIOS pour le sous-système de disque

AH	Paramètres d' Entrée	Paramètres de Sortie	Description
0	**dl** = lecteur (0..7fh disquette, 80h..ffh disque dur)	**ah**- statut (0 et carry nul si aucune erreur, code d'erreur si erreur).	Réinitialise l'unité de disques indiquée. La réinitialisation d'un disque dur réinitialise également les lecteurs de disquette.
1	**dl** = lecteur (comme ci-dessus)	**ah**- 0 **al**- statut de l'opération de disque précédente .	Cet appel renvoie les valeurs de statut suivantes dans **al**: 0- pas d'erreur 1- commande non valide 2- marque d'adresse introuvable 3- disque protégé en écriture 4- secteur non trouvé 5- erreur de réinitiaklisation 6- média enlevé 7- mauvaise table de paramètres 8- débordement de DMA 9- une opération de DMA a franchi la frontière de 64K 10- drapeau de secteur illégal 11- drapeau de piste illégale 12- média illégal 13- # de secteur non valide 14- marque d'adresse de contrôle de

			données rencontrée 15- erreur de DMA 16- erreur de CRC de données 17- erreur de données corrigées par ECC 32- échec du contrôleur de disque 64- erreur de recherche 128- erreur de dépassement de temps 170- lecteur pas prêt l'erreur 187- erreur indéterminée 204- erreur d'écriture 224- erreur de statut 255. --échec du contact
2	**al**- # de secteurs à lire **es:bx**- adresse du tampon **cl**- bits 0..5: # de secteur **cl**- bits 6/7: bits 8 & 9 de la piste **ch**- bits 0..7 de la piste **dl**- lecteur (comme ci-dessus) **dh**- bits 0..5: # de tête **dh**- bits 6/7: bits 10 & 11 de la piste	**ah**- statut de retour **al**- erreur de lecture en rafale carry - 0: succès - 1: erreur	Lit le nombre indiqué de secteurs de512 bytes à partir du disque. Les données lues doivent être de 64 K bytes ou moins.
3	pareils que (2) ci-dessus	pareils que (2) ci-dessus	Écrit le nombre indiqué de secteurs de512 bytes sur le disque. Les données écrites ne doivent pas excéder 64 K bytes de longueur.
4	pareils que (2) ci-dessus sauf qu'on n'a pas besoin de tampon.	pareils que (2) ci-dessus	Vérifie les données dans le nombre indiqué de secteurs de 512 bytes sur le disque.
0Ch	pareils que (4) ci-dessus sauf qu'on n'a pas besoin de # de secteur	pareils que (2) ci-dessus	Envoie la tête du disque sur la piste du disque indiquée.
0Dh	**dl** = # lecteur (80h ou 81h)	**ah**- statut de retour (carry =0 : aucune erreur, 1: erreur).	Réinitialise le contrôleur de disque dur.

Note: voir la documentation appropriée du BIOS pour des informations additionnelles sur le support du BIOS pour le sous-système de disque.

13.2.6 INT 14h- E/S série

Instruction: int 14h
Opération du BIOS: Accède au port de communication série.
Paramètres: **ax**, **dx**

Le BIOS d'IBM supporte jusqu'à quatre ports de communications série différents (le matériel en supporte jusqu'à huit). En général, la plupart des PCS ont un ou deux ports série (COM1: et COM2:) installé. int 14h supporte quatre sousbfonctions- initialiser, transmettre un caractère, recevoir un caractère et statut. Pour chacun des quatre services, le numéro de port série (une valeur dans la plage 0..3) est dans le registre **dx** (0=COM1:, 1=COM2:, etc.). int 14h attend et renvoie d'autres données dans le registre **al** ou **ax**.

13.2.6.1 AH=0: Initialisation du Port Série

La sous-fonction zéro initialise un port série. Cet appel vous permet de spécifier le nombre de bauds, choisir le mode de parité, choisir le nombre de bits d'arrêt et le nombre de bit transmis sur la ligne série. Ces paramètres tous sont spécifiés par la valeur dans le registre **al** en utilisant les codages de bits suivants:

Bits	Fonction
5..7	Selectionner le nombre de bauds 000- 110 bauds

001- 150
010- 300
011- 600
100 1200
101- 2400
110- 4800
111- 9600

3..4 Selectionner la parité
00- Pas de parité
10- Impair
10- Pas de parité
11- Pair

2 Bits d'arrêt
0- Un bit d'arrêt
1-Deux bits d'arrêt

0..1 Taille de caractère
10- 7 bits
11- 8

Bien que le matériel standard des ports série du PC supporte 19.200 bauds, quelques bios peuvent ne pas supporter cette vitesse.

Exemple: Initialiser COM1: à 2400 bauds, pas de parité, données sur huit bits et deux bits d'arrêt

```
mov    ah, 0              ;Opcode d'initialisation
mov    al, 10100111b             ;Paramètre de données.
mov    dx, 0              ;Port COM1:.
int    14h
```

Après l'appel au code d'initialisation, le statut du port série est retourné dans **ax** (voir Statut du Port Série, ah=3, ci-dessous).

13.2.6.2 AH=1: Transmettre un caractère au port série

Cette fonction transmet le caractère dans le registre **al** par le port série indiqué dans le registre **dx**. Au retour, si ah contient zéro, alors le caractère a été transmis correctement. Si le bit 7 de **ah** contient un, au retour, alors une erreur s'est produite. Les sept autres bits contiennent tous les statuts d'erreur retournés par l'appel à **GetStatus** sauf l'erreur de dépassement de temps (qui est retournée dans le bit sept). Si une erreur est signalée, vous devriez employer la sous-fonction trois pour obtenir les valeurs réelles d'erreur du matériel de port série.

Exemple: Transmettre un caractère par le port COM1:

```
mov    dx, 0        ;Selectionner COM1:
mov    al, 'a'             ;Caractère à transmettre
mov    ah, 1        ;Opcode de transmission
int    14h
test   ah, 80h             ;Vérifier les erreur
jnz    SerialError
```

Cette fonction attendra que le port série finisse de transmettre le dernier caractère (s'il y en a un) et puis stockera le caractère dans le registre de transmission.

13.2.6.3 AH=2: Recevoir un caractère du port série

La sous-fonction deux est employée pour lire un caractère sur le port série. À l'entrée, **dx** contient le numéro du port série. À la sortie, **al** contient le caractère lu sur le port série et le bit sept de ah contient le statut d'erreur. Quand cette routine est appellée, elle ne revient pas à l'appelant avant qu'un caractère soit reçu sur le port série.

Exemple: Lecture d'un caractère sur le port COM1:

```
mov    dx, 0        ;Selectionner COM1:
mov    ah, 2        ;Opcode de réception
int    14h
```

```
        test   ah, 80h            ;Vérifier les erreur
        jnz    SerialError

    <Le caractère reçu est maintenant dans AL>
```

13.2.6.4 AH=3: Statut du port série

Cet appel renvoie des informations de statut sur le port série, y compris si une erreur s'est produite, si un caractère a été reçu dans le tampon de réception, si le tampon de transmission est vide, et d'autres informations utiles. A l'entrée de cette routine, le registre **dx** contient le numéro de port série. A la sortie, le registre **ax** contient les valeurs suivantes:

AX:	**Signification du bit**
15	Erreur de dépassement de temps
14	Le registre de décalage de l'émetteur est vide
13	Le registre de stockage de l'émetteur est vide
12	Erreur de détection d'arrêt
11	Erreur de cadre
10	Erreur de parité
9	Erreur de débordement
8	Données disponibles
7	Détection du signal de la ligne de réception
6	Indicateur de sonnerie
5	Ensemble de données prêt (DSR: Data Set Ready)
4	Prêt à envoyer (CTS: Clear To Send)
3	Détection du signal delta de réception
2	Détecteur de chute décalée de sonnerie
1	Ensemble de données delta prêt
0	Delta prêt à envoyer

Il y a quelques bits utiles, ne concernant pas des erreurs, retournés dans ces informations de statut. Si le bit de données disponibles est à un (bit # 8), alors, le port série a reçu des données et vous devriez les lire sur le port série. Le bit Registre de stockage de l'émetteur vide (bit # 13) vous indique si l'opération de transmission sera retardée en attente du caractère courant à transmettre ou si le prochain caractère sera immédiatement transmis. En testant ces deux bits, vous pouvez effectuer d'autres opérations en attendant que le registre de transmission devienne disponible ou que le registre de réception contienne un caractère.

Si vous vous intéressez aux communications séries, vous devriez trouver un exemplaire du Guide du Programmeur C pour les Communications Série de Joe Campbell. Bien qu'écrit spécifiquement pour des programmeurs en C, ce livre contient beaucoup d'informations utiles aux programmeurs travaillant en n'importe quel langage de programmation. Voyez la bibliographie pour plus de détails.

13.2.7 INT 15h - Services divers

A l'origine, int 15h fournissait les services de lecture et écriture du lecteur de cassettes[1]. Presque tout de suite, chacun s'est rendu compte que les cassettes étaient de l'histoire ancienne, aussi IBM a commencé à employer int 15h pour beaucoup d'autres services. Aujourd'hui, int 15h est employé pour une grande variété de fonctions comprenant accéder à la mémoire étendue, lire la carte d'adapteur de jmanette de jeu, et beaucoup, beaucoup d'autres opérations. Excepté les appels du joystick, la plupart de ces services sont en dehors du sujet de ce texte. Vérifiez la bibliographie pour obtenir des informations sur cet appel du BIOS.

13.2.8 INT 16h - Services clavier

Instruction: int 16h
Opération du BIOS: Lire une touche, tester une touche, ou obtenir le statut du clavier.
Paramètres: **al**

[1] Pour dont la mémoire ne remonte pas si loin, avant qu'il y ait des disques durs, on utilisait seulement des disquettes. Et avant qu'il y ait des disquettes, on utilisait des lecteurs de cassettes pour stocker des programmes et des données. Le PC d'IBM original a été présenté vers la fin de 1981 avec un port pour cassette. Déjà en 1982, plus personne n'employait le lecteur de cassettes pour stocker des données.

Le BIOS du PC d'IBM fournit plusieurs appels de fonction en rapport avec le clavier. Comme pour de nombreuses routines du BIOS du PC, le nombre de fonctions a augmenté au cours des années. Cette section décrit les trois appels qui étaient disponibles sur le PC IBM d'origine.

13.2.8.1 AH=0: Lire une touche du clavier

Si int 16h est appellé avec **ah** égal à zéro, le BIOS ne renverra pas la contrôle à l'appelant tant qu'une touche n'est pas disponible dans tampon de frappes du système. Au retour, **al** contient le code ASCII pour la touche lue dans le tampon et **ah** contient le code de balayage de clavier *(keyboard scan code)*. Les codes de balayage de clavier sont décrits dans les annexes.

Certaines touches sur le clavier du PC n'ont aucun code correspondant ASCII. Les touches de fonction, Pos 1 (Home), Pg AR (PgUp), Fin (End), Pg AV (PgDn), les touches de déplacement du curseur et les touches ALT en sont de bons exemples. Quand une telle touche est appuyée, int 16h renvoie un zéro en **al** et le code de balayage de clavier dans **ah**. Par conséquent, toutes les fois qu'un code ASCII zéro est retourné, vous devez vérifier le registre **ah** pour déterminer quelle touche a été appuyée.

Notez que la lecture d'une touche du clavier employant l'appel int 16h du BIOS ne renvoie pas la touche appuyée en écho à l'affichage. Vous devez appeler **putc** ou employer **int 10h** pour afficher le caractère une fois que vous l'avez lu, si vous le voulez le renvoyer à l'écran.

Exemple: Lire une séquence de frappes du clavier jusqu'à ce qu'Enter soit appuyé.

```
ReadLoop:           mov    ah, 0        ;Opcode de Lecture de touche
                    int    16h
                    cmp    al, 0        ;Fonction speciale?
                    jz     ReadLoop     ;Si oui, pas d'écho pour la frappe
                    putc
                    cmp    al, 0dh      ;Retour Charriot (ENTER)?
                    jne    ReadLoop
```

13.2.8.2 AH=1: Voir si une touche est disponible sur le clavier

Cette sous-fonction particulière d'int 16h vous permet de vérifier si une touche est disponible dans le tampon de frappes du système. Même si une touche n'est pas disponible, le contrôle est retourné (tout de suite!) à l'appelant. Avec cet appel, vous pouvez de temps en temps sonder le clavier pour voir si une touche est disponible et continuer de travailler si une n'a pas été appuyée (au lieu de bloquer l'ordinateur jusqu'à une touche soit appuyée).

Il n'y a aucun paramètre d'entrée à cette fonction. Au retour, le drapeau zéro sera clair si une touche est disponible, à un s'il nj'y a aucune touche dans le tampon de frappes. Si une touche est disponible, alors **ax** contiendra les codes de balayage et ASCII pour cette touche. Cependant, cette fonction n'enlèvera pas cette frappe du tampon de frappes. La sous-fonction #0 doit être employée pour enlever des caractères. L'exemple suivant démontre comment construire un générateur de nombres aléatoires en utilisant la fonction de test du clavier:

Exemple: Générer un nombre aléatoire tout en attendant une frappe

```
; D'abord, effacer les caractères qui pourraient se trouver dans le tampon.

ClrBuffer:          mov    ah, 1        ;Une touche est-elle disponible?
                    int    16h
                    jz     BufferIsClr  ;Si non, arrêter le vidage
                    mov    ah, 0        ;Vider ce caractère du
                    int    16h          ; tampon et recommencer.
                    jmp    ClrBuffer
BufferIsClr: mov    cx, 0               ;Initialiser le nombre "random".
GenRandom:          inc    cx
                    mov    ah, 1        ;Vérifier si une touche est dispo.
                    int    16h
                    jz     GenRandom
                    xor    cl, ch       ;Randomiser encore plus.
                    mov    ah, 0        ;Lire un caractère du tampon
                    int    16h
  ; Le nombre aléatoire est maintenant dans CL, la touche appuyée par
```

```
; l'utilisateur dans AX.;
```

Pendant qu'elle attend une touche, cette routine incrémente constamment le registre **cx**. Puisque les êtres humains ne peuvent pas répondre rapidement (au moins en termes de micro-secondes), le registre cl débordera un grand nombre de fois, même pour la dactylo la plus rapide. En conséquence, le **cl** contiendra une valeur aléatoire puisque l'utilisateur ne pourra pas contrôler quand une touche est appuyée.

13.2.8.3 AH=2: Renvoyer le statut de la touche majuscule du clavier

Cette fonction renvoie l'état de diverses touches sur le clavier de PC dans le registre **al**. Les valeurs retournées sont comme suit:

AL:	**Signification**
7	État d'Insertion (bascule de la touche INS)
6	Blocage des majuscules (1=blocage actif)
5	Blocage des chiffres du pavé numérique (1=blocage actif)
4	Blocage du défilement (1=blocage actif)
3	Alt (1=touche Alt enfoncée)
2	Ctrl (1=touche Ctrl enfoncée)
1	Majuscule gauche (1=touche MAJ gauche enfoncée)
0	Majuscule droite (1=touche MAJ droite enfoncée)

A cause d'un bug dans le code du BIOS, ces bit reflètent seulement l'état actuel de ces touches, ils ne reflètent pas nécessairement le statut de ces touches quand la touche à lire suivante du tampon de frappes du système a été enfoncée. Afin de s'assurer que ces bit de statut correspondent à l'état de ces touches quand un code de balayage est lu du tampon de frappes, vous devez vider le tampon , attendre jusqu'à ce qu'une touche soit appuyée et puis vérifier immédiatement le statut du clavier.

13.2.9 INT 17h - Services Imprimante

Instruction:	int 17h
Opération du BIOS:	Imprimer les données et tester l'état de l'imprimante.
Paramètres:	**ax**, **dx**

L'Int 17h contrôle les interfaces d'imprimante parallèle sur le PC d'IBM plus ou moins de la même façon que l'Int 14h contrôle les ports série. Puisque la programmation d'un port de parallèle est considérablement plus facile que le contrôle d'un port série, utiliser la routine int 17h est certes plus facile qu'utiliser les routines de l'Int 14h.

L'Int 17h fournit trois sous-fonctions, indiquées par la valeur dans le registre **ah**. Ces sous-fonctions sont:

0-Imprime le caractère dans le registre AL.
1-Initialise l'imprimante.
2-Retourne l'état de l'imprimante.

Chacune de ces fonctions est décrite dans les sections suivantes.

Comme les services des ports série, les services des ports d'imprimante vous permettent d'indiquer laquelle des trois imprimantes installées dans le système vous souhaitez employer (LPT1:, LPT2:, ou LPT3:). La valeur dans le registre dx (0..2) indique quel port d'imprimante doit être employé.

Une dernière note: sous DOS, il est possible de réorienter toutes les sorties d'imprimante sur un port série. C'est très utile si vous utilisez une imprimante série. Les services d'imprimante du BIOS parlent seulement aux adapteurs d'imprimante parallèle. Si vous devez envoyer des données à un imprimante série en utilisant le BIOS, vous devrez employer l'Int 14h pour transmettre les données par un port série.

13.2.9.1 AH=0: Imprimer un caractère

Si **ah** est à zéro quand vous appelez int 17h, alors le BIOS imprimera le caractère qui est dans le registre **al**. La manière exacte dont le code caractère dans le registre **al** est traité dépend complètement de l'imprimante que vous utilisez. La plupart des imprimantes, cependant, respectent le jeu de caractères ASCII imprimable et quelques

caractères de contrôle en plus. Beaucoup d'imprimantes imprimeront également tous les symboles dans le jeu de caractères IBM/ASCII (y compris les symboles européen, de dessin de ligne et autres symboles spéciaux). La plupart des imprimantes traitent les caractères de contrôle (particulièrement les séquences ESC) de façons complètement différentes. Par conséquent, si vous avez l'intention d'imprimer quelque chose d'autre que des caractères ASCII standard, soyez prévenu que votre logiciel peut ne pas fonctionner sur des imprimantes autres que celles de la marque avec la quelle vous développez votre logiciel.

Au retour de la sous-fonction zéro d'Int 17h, le registre **ah** contient l'état actuel. Les valeurs réellement retournées sont décrites dans la section sur la sous-fonction numéro deux.

13.2.9.2 AH=1: Initialiser l'imprimante

Exécuter cet appel envoie une impulsion électrique à l'imprimante lui demandant de s'initialiser. Au retour, le registre **ah** contient l'état de l'imprimante comme indiqué dans la fonction numéro deux.

13.2.9.3 AH=2: Renvoyer l'état de l'imprimante

Cet appel de fonction vérifie l'état de l'imprimante et le renvoie dans le registre **ah**. Les valeurs retournées sont:

AH:	Signification
7	1=Imprimante occupée, 0= Imprimante libre
6	1=Reconnaissance de l'imprimante
5	1=Signal de manque de papier
4	1= Imprimante sélectionnée
3	1=Erreur d'E/S
2	Inutilisé
1	Inutilisé
0	Erreur de dépassement de temps

La reconnaissance de l'imprimante est, essentiellement, un signal superflu (puisque Imprimante occupée/libre vous fournit la même information). Aussi longtemps que l'imprimante est occupée, elle n'acceptera pas des données supplémentaires. Par conséquent, appeler la fonction d'impression de caractère (ah=0) provoquera un retard.

Le signal de manque papier est lancé toutes les fois que l'imprimante détecte qu'elle n'a plus de papier. Ce signal n'est pas souvent mis en application sur les adapteurs d'imprimante. Sur de tels adapteurs il est toujours programmé à zéro (même si l'imprimante n'a plus de papier). Par conséquent, voir un zéro dans cette position de bit ne garantit pas toujours qu'il y a du papier dans la machine. Y voir un 1, cependant, signifie certainement que votre imprimante n'a plus de papier.

Le bit d'imprimante selectionnée contient 1 aussi longtemps que l'imprimante est en ligne. Si l'utilisateur déconnecte l'imprimante, alors ce bit sera mis à zéro.

Le bit d'erreur d'E/S contient 1 si une erreur générale d'E/S s'est produite.

Le bit d'erreur de dépassement de temps contient 1 si la routine BIOS a attendu pendant une période prolongée que l'imprimante devienne "libre" mais qu'elle est restée "occupée".

Notez que certains périphériques (autre que des imprimantes) se connectent également au port parallèle, souvent en plus d'une imprimante parallèle. Certains de ces périphériques emploient les lignes de signal d'erreur/état pour renvoyer des données au PC. Le logiciel commandant de tels périphériques remplace souvent la routine Int 17h (par l'intermédiaire d'une technique dont nous parlerons plus tard) et renvoie toujours un état "pas d'erreur" ou " dépassement de temps" si une erreur se produit sur le matériel d'impression. Par conséquent, vous devriez essayer de ne pas dépendre trop fortement du changement de ces signaux quand vous faites des appels à l'Int 17h du BIOS.

13.2.10 INT 18h - Lancer le BASIC

Instruction:	int 18h
Opération du BIOS:	Activer le BASIC de la ROM.
Paramètres:	aucun

Exécuter l'lint 18h déclenche l'interprèteur BASIC de la ROM dans un PC d'IBM. Cependant, vous ne devriez pas utiliser ce mécanisme pour lancer le BASIC puisque beaucoup de compatibles PC n'ont pas de BASIC dans la ROM et le résultat de l'exécution de l'Int 18h est non défini.

13.2.10 INT 19h - Redémarrer l'ordinateur

Instruction: int 19h
Opération du BIOS: Relancer le système.
Paramètres: aucun

Exécuter cette interruption a le même effet qu'appuyer ctrl-alt-sup sur le clavier. Pour des raisons évidentes, ce service d'interruption devrait être manipulé avec précaution !

13.2.12 INT 1Ah - Horloge temps réel (RTC: Real time Clock)

Instruction: int 1Ah
Opération du BIOS: Services de l'Horloge Temps Réel.
Paramètres: **ax**, **cx**, **dx**

Il y a deux services fournis par cette routine BIOS - lire l'horloge et régler l'horloge. L'horloge temps réel du PC maintient un compteur qui compte le nombre de 1/18èmes de seconde qui ont passé depuis minuit. Quand vous lisez l'horloge, vous obtenez le nombre de "tics" qui se sont produit depuis ce moment. Quand vous réglez l'horloge, vous indiquez le nombre de "tics" qui se sont produits depuis minuit. Comme d'habitude, le service en question est choisi par l'intermédiaire de la valeur dans le registre **ah**.

13.2.12.1 AH=0: Lire l'Horloge Temps Réel

Si ah = 0, alors Int 1Ah renvoie une valeur de 33 bits dans **al:cx:dx** comme suit:

Reg:	**Valeur renvoyée**
dx	Mot de poids faible du compteur de l'horloge
cx	Mot de poids fort du compteur de l'horloge
al	Zéro si le timer a dépassé 24 heures, Non-zéro sinon.

La valeur de 32 bits dans cx:dx représente le nombre de périodes de 55 millisecondes qui se sont écoulées depuis minuit..

13.2.12.2 AH=1: Régler l'horloge temps réel

Cet appel vous permet de régler la valeur actuelle du temps système. **cx:dx** contient le compteur courant (en incréments de 55ms) depuis le précédent minuit. Cx contient le mot de poids fort., dx contient le mot de poids faible..

13.3 Une introduction à MS-DOS™

MS-DOS fournit toutes les fonctions de base de gestionnaire de fichiers et de périphériques exigées par la plupart des applications fonctionnant sur un PC IBM. MS-DOS gère les E/S de fichiers, de caractères, la gestion de la mémoire ainsi que d'autres fonctions diverses d'une manière (relativement) efficace. Si voulez sérieusement écrire des logiciels pour le PC, vous devrez sympathiser avec le MS-DOS.

Le titre de cette section est « Une introduction au MS.-DOS ». Et c'est exactement ce que cela signifie. En aucune manière, on ne peut traiter le MS-DOS dans un unique chapitre. Vu tous les livres qui existent déjà sur le sujet, il ne peut probablement même pas être couvert par un livre unique (cela n'a jusqu'à maintenant pas été le cas, pourtant Microsoft a écrit un livre de 1.600 pages sur le sujet et ne couvre même pas le sujet entièrement). Tout ceci amène à une excuse. Il n'y a aucune manière que ce sujet puisse être traité de façon plus que superficielle dans un chapitre unique. Si vous envisagez sérieusement d'écrire des programmes en assembleur pour le PC, vous aurez besoin de compléter ce texte avec d'autres. Les livres additionnels sur le MS-DOS incluent: la Référence du Programmeur MS-DOS (également appelée le Manuel de Référence Technique MS-DOS), le Guide du Programmeur PC IBM de Peter

Norton, l'Encyclopédie du MS-DOS, et le Guide du Developpeur MS-DOS. Ceci, naturellement, est seulement une liste partielle des livres qui sont disponibles. Voyez la bibliographie dans les annexes pour plus de détails. Sans aucun doute, le Manuel de Référence Technique MS-DOS est le texte le plus important à obtenir. C'est la description officielle des appels et des paramètres du MS-DOS.

MS-DOS a une histoire longue et colorée[2]. Durant toute sa vie, il a subi plusieurs révisions, chacune prétendant être meilleure que la précédente. Les origines du MS.-DOS remontent au logiciel d'exploitation CP/M-80 écrit pour le microprocesseur 8080 d'Intel. En fait, le MS-DOS v1.0 n'était guère mieux qu'un clone du CP/M-80 pour le microprocesseur 8088 d'Intel. Malheureusement, les capacités de gestion de fichiers du CP/M-80 étaient horribles, pour rester poli. Par conséquent, DOS[3] a amélioré le CP/M. De nouvelles capacités de gestion de fichiers, compatibles avec Xenix et Unix, ont été ajoutées au DOS, produisant le MS-DOS v2.0. Des appels supplémentaires ont été ajoutés aux versions postérieures du MS-DOS. Même avec l'introduction d'OS/2 et de Windows NT (qui, au moment où ceci est écrit, a toujours à conquérir le monde), Microsoft travaille toujours aux perfectionnements du MS-DOS qui peut produire encore des versions postérieures.

Chaque nouvelle fonctionnalité ajoutée au DOS a introduit de nouvelles fonctions du DOS, tout en préservant toute la fonctionnalité des versions précédentes du DOS. Quand Microsoft a récrit les routines de gestion de fichiers dans la version deux du DOS, elles ne remplacèrent pas les anciens appels, elles en ont simplement ajouté de nouveaux. Tandis que ceci préservait la compatibilité logicielle des programmes qui fonctionnaient sous l'ancienne version du DOS, cela a produit un DOS avec deux ensembles de services de fichiers, fonctionellement identiques, mais cependant incompatibles.

Nous allons seulement nous concentrer sur un petit sous-ensemble des commandes du DOS en ce chapitre. Nous allons ignorer totalement les commandes désuètes qui ont été remplacées par de nouvelles commandes, meilleures, dans des versions postérieures du DOS. En outre, nous allons sauter la description de ces appels qui ont très peu d'utilisation dans la programmation quotidienne. Pour un examen complet, détaillé, des commandes non couvertes en ce chapitre, vous devriez considérer l'acquisition d'un des livres mentionnés ci-dessus.

13.3.1 La Séquence d'appel de MS-DOS™

MS-DOS est appelé par l'intermédiaire de l'instruction **int 21h**. Pour choisir une fonction DOS appropriée, vous chargez le registre **ah** avec un numéro de fonction avant de lancer l'instruction **int 21h**. La plupart des appels au DOS exigent d'autres paramètres en plus. Généralement, ces autres paramètres sont passés dans l'ensemble des registres du CPU. Les paramètres spécifiques seront discutés pour chaque appel. À moins que MS-DOS ne renvoie une valeur spécifique dans un registre, tous les registres de CPU sont préservés lors d'un appel au DOS[4].

13.3.2 Les Fonctions orientées caractères de MS-DOS™

Le DOS fournit 12 appels d'E/S orientés caractères. La plupart de celles-ci s'occupent d'écrire et de lire les données sur/depuis le clavier, l'écran vidéo, le port série et le port d'imprimante. Toutes ces fonctions ont des services BIOS correspondants. En fait, le DOS appelle habituellement la fonction appropriée du BIOS pour effectuer l'opération d'E/S. Cependant, en raison de redirection d'E/S par le DOS et de routines de gestionnaires de périphérique, ces fonctions n'appellent pas toujours les routines du BIOS. Par conséquent, vous ne devriez pas appeler les routines du BIOS (au lieu du DOS) sous prétexte que le DOS finit par appeler le BIOS. Ceci peut empêcher votre programme de fonctionner avec certains périphériques supportés par le DOS.

Excepté la fonction numéro sept, tous les appels orientés caractères suivants contrôlent le dispositif d'entrée de console (clavier) pour un éventuel Ctrl-C. Si l'utilisateur appuye sur Contrôle-C, le DOS exécute une instruction **int 23h**. Habituellement, cette instruction arrrêtera le programme et le contrôle sera rendu au DOS. Gardez ceci à l'esprit en lançant ces appels.

Microsoft considère ces appels obsolètes et ne garantit pas qu'ils seront présents dans de futures versions de DOS. Prenez donc ces 12 premières routines avec des pincettes. Notez que la Bibliothèque Standard UCR fournit de toute

[2] L'Encyclopédie du MS-DOS fait l'exposé de l'histoire du MS-DOS du point de vue de Microsoft. Naturellement, c'est une présentation partiale, mais néanmoins intéressante.

[3] Ce texte emploie "DOS" pour signifier MS-DOS..

[4] C'est ce que prétend Microsoft. Ceci peut ou peut ne pas être vrai pour toutes les versions du DOS.

façon les fonctionnalités de plusieurs de ces appels, et elles font les appels appropriés au DOS, pas ceux qui sont obsolètes.

Tableau 51: Fonctions DOS Orientées Caractères

# Fonction (AH)	Paramètres en entrée	Paramètres en sortie	Description
1		**al**- caractère lu	Entrée console avec écho : lit un seul caractère sur le clavier et l'affiche à l'écran.
2	**dl**- caractère affiché.		Sortie console: affiche un seul caractère à l'écran.
3		**al**- caractère lu	Entrée auxiliaire : lit un seul caractère sur le port série.
4	**dl**- caractère affiché.		Sortie auxiliaire : écrit un seul caractère sur le port de sortie.
5	**dl**- caractère affiché.		Sortie imprimante : écrit un seul caractère sur le port d'imprimante .
6	**dl**- caractère affiché (si pas 0FFh).	**al**- caractère lu (si **dl**=0FFh)	E/S console directes : à l'entrée, si **dl** contient 0FFh, cette fonction essaie de lire un caractère sur le clavier. Si un caractère est disponible, elle renvoie le flag zéro à 0 et le caractère dans al. Si aucun caractère n'est disponible, elle renvoie le flag zéro à 1. A l'entrée, si **dl** contient une valeur autre que 0FFh, cette routine envoie le caractère à l'affichage. Cette routine ne fait pas la vérification de Ctrl-C.
7		**al**- caractère lu	Entrée console directe : lit un caractère sur le clavier. Ne l'affiche pas à l'écran. Cette routine ne fait pas la vérification de Ctrl-C.
8		**al**- caractère lu	Lecture clavier sans écho : tout comme la fonction 7 mais fait la vérification de Ctrl-C.
9	**ds :dx**- pointeur sur une chaîne terminée par $.		Affiche une chaîne : cette fonction affiche les caractères de l'emplacement **ds:dx** jusqu'à (mais excepté) un caractère de terminaison "$".
0Ah	**ds :dx**- pointeur sur le tampon d'entrée.		Entrée clavier avec tampon: cette fonction lit une ligne de texte sur le clavier et la stocke dans le tampon d'entrée sur lequel pointe **ds:dx**. Le premier byte du tampon doit contenir un compteur entre un et 255 qui contient le nombre maximum de caractères permis dans le tampon d'entrée. Cette routine stocke le nombre réel de caractères lus dans le deuxième byte. Les caractères d'entrée réels commencent au troisième byte du tampon .
0Bh		**al**- statut (0 = prêt, 0FFh = pas prêt)	Vérification du statut du clavier : détermine si un caractère est fourni par le clavier.
0Ch	**al**- opcode DOS 1, 6, 7 ou 8.	**al**- caractère en entrée si opcode 1, 6, 7 ou 8	Vidage du tampon : cet appel vide le tampon de frappes système et puis exécute la commande DOS indiquée dans le registre **al** (si **al**=0, aucune autre action)

Les fonctions d'E/S 1, 2, 3, 4, 5, 9, et 0Ah sont obsolètes et vous ne devriez pas les employer. Employez les appels d'E/S de fichier du DOS à la place (opcodes 3Fh et 40h).

13.3.3 Les Commandes de disque de MS-DOS™

MS-DOS fournit plusieurs commandes qui vous permettent de déterminer le lecteur par défaut et effectuer d'autres opérations. Le tableau suivant présente ces fonctions.

Tableau 52: Les fonctions d'unité de disque du DOS

# Fonction (AH)	Paramètres en entrée	Paramètres en sortie	Description
0Dh			Remise à zéro des lecteurs : vide tous les tampons de fichiers sur le disque. En général, appelé par les gestionnaires de Ctrl-C ou des sections de code qui ont besoin de garantir la cohérence des fichiers au cas ou une erreur se produise
0Eh	**dl**- numéro de lecteur.	**al**- nombre de partitions logiques	Détermine le lecteur par défaut : met le lecteur DOS par défaut à la valeur spécifiée (0=A, 1=B, 2=C, etc.). Renvoie le nombre de partitions logiques du système, bien qu'elles puissent ne pas être contigües dans **al**.
19h		**al**- numéro de lecteur par défaut	Renvoie le lecteur par défaut: renvoie le numéro courant du lecteur par défaut du système (0=A, 1=B, 2=C, etc.)..
1Ah	**ds:dx** – adresse de la Disk Transfer Area (Zone de Transfert de Disque).		Détermine l'adresse de la DTA: détermine l'adresse que MS-DOS emploie pour les commandes obsolètes d' E/S de fichier et Find First/Find Next..
1Bh		**al**- secteurs/cluster **cx**- bytes/secteur **dx**- # de secteurs **ds:bx**- pointe sur le byte descripteur de média	Obtient des données sur le lecteur par défaut: renvoie des informations sur le disque dans le lecteur par défaut. Voir également la fonction 36h. Des valeurs classiques pour le byte descripteur de média incluent: 0F0h- 3.5" 0F8h- disque dur 0F9h- 720K 3.5" ou 1.2M 5.25" 0FAh- 320K 5.25" 0FBh- 640K 3.5" 0FCh- 180K 5.25" 0FDh- 360K 5.25 0FEh- 160K 5.25" 0FFh- 320K 5.25.
1Ch	**dl**- numéro de lecteur	Voir ci-dessus	Obtient des données sur le lecteur: comme ci-dessus sauf que vous pouvez spécifier un numéro de lecteur dans le registre **dl** (0=default, 1=A, 2=B, 3=C, etc.)...
1Fh		**al**- contient 0FFh si erreur, 0 sinon. **ds:bx**- pointeur sur le DPB	Obtient le Bloc de Paramètres de Disque (DPB) par défaut: si cette fonction réussit, elle renvoie un pointeur sur la structure suivante: Drive (byte) – numéro du lecteur (0-A, 1=B, etc.). Unit (byte) - numéro d'unité pour le lecteur SectorSize (mot) - # bytes/secteur. ClusterMask (byte) - secteurs/cluster moins un. Cluster2 (byte) - $2^{\text{clusters/secteur}}$ FirstFAT (word) - adresse du secteur où commence la FAT. FATCount (byte) - # de FATs. RootEntries (word) - # d'entrées dans le répertoire racine. FirstSector (word) - premier secteur du premier cluster.

			MaxCluster (word) - # de clusters sur le lecteur, plus un. FATsize (word) - # de secteurs de la FAT. DirSector (word) - premier secteur contenant un répertoire. DriverAdrs (dword) - adresse du pilote de périphérique. Médias (byte) - byte descripteur de média. FirstAccess (byte) – à un s'il y a eu un accès au lecteur. NextDPB (dword) - lien sur le DPB suivant dans la liste. NextFree (word) - dernier cluster alloué. FreeCnt (word) - nombre de clusters libres..
2Eh	**al**- drapeau de vérification (0 = pas de vérification, 1 = vérification active)		Bascule le drapeau de vérification: met en marche/ arrêt la vérification d'écriture. Habituellement arrêtée puisque c'est une opération lente, mais vous pouvez l'activer en exécutant des E/S critiques.
2Fh		**es :bx**- pointeur sur le DTA	Obtient l'adresse de la Zone de Transfert de Disque: renvoie un pointeur sur le DTA courant dans **es:bx**..
32h	**dl**- numéro de lecteur	Comme 1Fh	Obtient le DPB : comme la fonction 1Fh sauf que vous devez spécifier le numéro de lecteur (0=default, 1=A, 2=B, 3=C, etc.).
33h	**al**- 05 (code de sous-fonction)	**dl**- numéro de lecteur de démarrage	Obtient le lecteur de démarrage : renvoie le numéro du lecteur utilisé pour lancer le DOS (1=A, 2=B, 3=C, etc.).
36h	**dl**- numéro de lecteur.	**ax**- secteurs/cluster **bx**- clusters disponibles **cx**- bytes/ secteur **dx**- total des clusters	Obtient l'espace disque libre : renvoie la quantité d'espace libre. Cette fonction remplace les fonctions 1Bh et 1Ch qui ne supportent que des lecteurs de moins de 32 Mo. Cette fonction gère de plus grands disques. Vous pouvez calculer la quantité d'espace libre (en bytes) avec **bx*ax*cx**. Si une erreur survient, cette fonction renvoie 0FFFFh dans **ax**.
54h		**al**- statut de vérification	Obtient le statut de vérification : renvoie le statut courant du drapeau de vérification d'écriture dans **al**.

13.3.4 Les fonctions de gestion de fichiers « obsolètes » de MS-DOS™

Les fonctions DOS 0Fh - 18h, 1Eh, 20h-24h, et 26h - 29h sont un reliquat datant du CP/M-80. En général, vous ne vous occuperez pas du tout de ces derniers appels puisque MS-DOS v 2.0 et suivants procurent de bien meilleurs moyens d'accomplir les opérations remplies par ces fonctions.

13.3.5 Les fonctions de date et d'heure de MS-DOS™

Les fonctions de date et d'heure de MS-DOS renvoient la date du jour et l'heure basées sur des valeurs internes maintenues par l'horloge en temps réel (RTC). Les fonctions fournies par le DOS incluent la lecture et le réglage de la date et de l'heure. Ces valeurs de date et d'heure sont employées pour ajouter la date et l'heure aux fichiers qui sont créés sur le disque. Par conséquent, si vous changez la date ou l'heuregardez à l'esprit que cela aura des répercussions sur les fichiers que vous créerez par la suite. Notez que la Bibliothèque Standard UCR fournit également un ensemble de fonctions de date et d'heure qui sont souvent plus facile d'emploi que, ces appels au DOS.

Tableau 53: Les Fonctions de date et d'heure

# Fonction (AH)	Paramètres en entrée	Paramètres en sortie	Description

2Ah		**al**- jour (0=dim, 1=lun, etc.). **cx**- année **dh**- mois (1=Jan, 2=Fév, etc.). **dl**- jour du mois (1-31).	Obtient la Date : renvoie la date courante de MS-DOS.
2Bh	**cx**- année (1980-2099) **dh**- mois (1-12.). **dl**- jour (1-31).		Règle la Date : règle la date courante de MS-DOS.
2Ch		**ch**- heure (format 24h). **cl**- minutes **dh**- secondes **dl**- centièmes	Obtient l'Heure : renvoie l'heure courante de MS-DOS. Notez que le champ des centièmes de seconde a une résolution de 1/18ème de seconde.
2Dh	**ch**- heure **cl**- minutes **dh**- secondes **dl**- centièmes		Règle l'Heure : règle l'heure courante MS-DOS..

13.3.6 Les Fonctions de gestion de la mémoire de MS-DOS™

MS-DOS fournit trois fonctions de gestion de mémoire allouer, libérer, et redimensionner (modifier). Pour la plupart des programmes, ces trois appels d'attribution de mémoire ne sont pas employés. Quand le DOS exécute un programme, il donne toute la mémoire disponible, du début de ce programme jusqu'à la fin de la RAM, au processus qui s'exécute. Touter tentative d'allouer de la mémoire sans redonner d'abord au système la mémoire inutilisée produira une erreur "mémoire insuffisante".

Les programmes sophistiqués qui se terminent et demeurent résidents (TSR), exécutent d'autres programmes, ou accomplissent des tâches complexes de gestion de mémoire, peuvent nécessiter l'utilisation de ces fonctions de gestion de mémoire. Généralement ce type de programmes libère immédiatement toute la mémoire qu'il n'utilise pas et commence ensuite à allouer et à libérer la mémoire comme il le juge nécessaire.

Comme ce sont des fonctions complexes, elles ne devraient être employées que lorsque vous en avez une utilisation spécifique. L'abus de ces commandes peut avoir comme conséquence la perte de mémoire système qui peut être récupérée seulement en rebootant le système. Chacun des appels suivants renvoie l'état d'erreur dans le drapeau de retenue (carry flag). Si le drapeau est à zéro au retour, alors l'opération a été accomplie avec succès. Si le drapeau de retenue est à un quand le DOS retourne, alors le registre AX contient un des codes d'erreur suivants:

7- Blocs de contrôle de mémoire détruits
8- Mémoire insuffisante
9- Bloc d'adresse de mémoire invalide

Des notes additionnelles au sujet de ces erreurs seront discutées selon le besoin.

13.3.6.1 Allocation de mémoire

Fonction (**ah**): 48h
Paramètres d'entrée: **bx**- Taille du bloc demandé (en paragraphes)
Paramètres de sortie: Si pas d'erreur (retenue à zéro):
ax:0 pointe sur le bloc de mémoire alloué
Si erreur (retenue à un):
bx- Taille maximum d'allocation possible
ax- code d'erreur (7 ou 8)

Cet appel est employé pour allouer un bloc de mémoire. A l'entrée dans le DOS, **bx** contient la taille du bloc demandé en paragraphes (groupes de 16 bytes). A la sortie, en supposant aucune erreur, le registre **ax** contient l'adresse du segment du début du bloc alloué. Si une erreur se produit, le bloc n'est pas alloué et le registre **ax** est renvoyé contenant le code d'erreur. Si la demande d'attribution échouait en raison de mémoire insuffisante, le registre **bx** est renvoyé avec le nombre maximum de paragraphes réellement disponibles.

13.3.6.2 Désallocation de mémoire

Fonction (**ah**): 49h
Paramètres d'entrée: **es:0**- Adresse du segment du bloc à libérer
Paramètres de sortie: Si retenue à un, **ax**- contient le code d'erreur (7 ou 9)

Cet appel est employé pour libérer la mémoire allouée par l'intermédiaire de la fonction 48h ci-dessus. Le registre **es** ne peut pas contenir une adresse de mémoire arbitraire. Il doit contenir une valeur retournée par la fonction d'allocation de mémoire . Vous ne pouvez pas employer cet appel pour désaffecter une partie d'un bloc alloué. La fonction de modification d'allocation est utilisée pour cette opération.

13.3.6.3 Modification d'allocation de mémoire

Fonction (**ah**): 4Ah
Paramètres d'entrée: **es:0**- Adresse du bloc à modifier concernant la taille d'allocation
bx- Taille du nouveau bloc (en paragraphes)
Paramètres de sortie: Si erreur (retenue à un):
bx- Taille maximum d'allocation possible
ax- code d'erreur (7, 8 ou 9)

Cet appel est employé pour changer la taille d'un bloc alloué. A l'entrée, **es** doit contenir l'adresse de segment du bloc alloué retourné par la fonction d'allocation de mémoire. **bx** doit contenir la nouvelle taille de ce bloc en paragraphes. Alors que vous pouvez presque toujours réduire la taille d'un bloc, vous ne pouvez pas normalement augmenter la taille d'un bloc si d'autres blocs ont été alloués après le bloc à modifier. Gardez ceci à l'esprit en utilisant cette fonction.

13.3.6.4 Fonctions avancées de gestion de mémoire

L'opcode MS-DOS 58h permet aux programmeurs d'ajuster la stratégie d'attribution de mémoire de MS-DOS et contrôler l'utilisation des blocs de mémoire supérieure (UMBs). Il y a quatre sous-fonctions à cet appel, avec la valeur de sous-fonction apparaissant dans le registre **al**. La table suivante décrit ces appels:

Tableau 54: Les fonctions avancées de gestion de la mémoire

# Fonction (AH)	Paramètres en entrée	Paramètres en sortie	Description
58h	**al**- 0	**ax**- stratégie	Obtient la Stratégie d'Allocation : renvoie la stratégie d'allocation courante dans **ax** (voir tableau ci-dessous pour les détails).
58h	**al**- 1 **bx**- stratégie		Détermine la Stratégie d'Allocation : fixe la stratégie d'allocation courante à la valeur spécifiée dans **bx** (voir tableau ci-dessous pour les détails)..
58h	**al**- 2	**al**- drapeau de liaison	Obtient la liaison vers la mémoire supérieure : renvoie vrai ou faux (1/0) dans **al** pour déterminer si un programme peut allouer de la mémoire dans les blocs de mémoire supérieure.
58h	**al**- 3		Fixe la liaison vers la mémoire supérieure : connecte ou déconnecte la zone de mémoire supérieure. Quand elle

	bx- drapeau de liaison (0 = pas de liaison, 1 = OK)		est connectée, une application peut allouer de la mémoire à partir des UMB (en utilisant l'appel d'allocation normal du DOS)

Tableau 55: Les stratégies d'allocation de mémoire

Valeur	Nom	Description
0	Première occurrence basse (First Fit low)	Parcourt la mémoire conventionnelle à la recherche du premier bloc de mémoire libre de taille suffisante pour satisfaire la reqiuête d'allocation. C'est le cas par défaut.
1	Meilleure occurence basse (Best Fit low)	Parcourt la mémoire conventionnelle à la recherche du plus petit bloc de mémoire libre de taille suffisante pour satisfaire la reqiuête d'allocation.
2	Dernière occurence basse (Last Fit low)	Parcourt la mémoire conventionnelle à la recherche du premier bloc de mémoire libre de taille suffisante pour satisfaire la reqiuête d'allocation en partant de l'adresse la plus haute en descendant.
80h	Première occurence haute (First Fit High)	Parcourt la mémoire haute, puis la mémoire conventionnelle à la recherche du premier bloc de mémoire libre de taille suffisante pour satisfaire la reqiuête d'allocation.
81h	Meilleure occurence haute (Best Fit High)	Parcourt la mémoire haute, puis la mémoire conventionnelle à la recherche du plus petit bloc de mémoire libre de taille suffisante pour satisfaire la reqiuête d'allocation.
82h	Dernière occurence haute (Last Fit High)	Parcourt la mémoire haute, puis la mémoire conventionnelle à la recherche du premier bloc de mémoire libre de taille suffisante pour satisfaire la reqiuête d'allocation en partant des adresses les plus hautes en descendant.
40h	Première occurence haute seulement (First Fit Highonly)	Parcourt la mémoire haute seulement à la recherche du premier bloc de mémoire libre de taille suffisante pour satisfaire la reqiuête d'allocation.
41h	Meilleure occurence haute seulement (Best Fit Highonly)	Parcourt la mémoire haute seulement à la recherche du plus petit bloc de mémoire libre de taille suffisante pour satisfaire la reqiuête d'allocation.
42h	Dernière occurence haute seulement(Last Fit Highonly)	Parcourt la mémoire haute seulement à la recherche du premier bloc de mémoire libre de taille suffisante pour satisfaire la reqiuête d'allocation en partant des adresses les plus hautes en descendant.

Ces différentes stratégies d'allocation peutvent avoir un impact sur les performances du système. Pour une analyse des différentes stratégies de gestion de mémoire, consultez un bon texte de théorie des systèmes d'exploitation.

13.3.7 Fonctions de contrôle de processus de MS-DOS

Le DOS fournit plusieurs services traitant le chargement, l'exécution et la terminaison des programmes. Plusieurs de ces fonctions ont été rendues obsolètes par des versions postérieures du DOS. Il y a trois[5] fonctions d'intérêt général - terminaison de programme, TSR et exécution de programme. Ces trois fonctions seront discutées dans les sections suivantes.

13.3.7.1 Terminer l'exécution du programme

[5] En fait, il y en a d'autres. Voyez le manuel de référence technique du DOS pour plus de détails. Nous considérerons seulement ces trois-ci.

Fonction (**ah**): 4Ch

Paramètres d'entrée: **al**- code de sortie
Paramètres de sortie: Ne retourne pas à votre programme.

C'est l'appel de fonction normalement employé pour terminer votre programme. Il renvoie le contrôle au processus appelant (normalement, mais pas nécessairement, DOS). Un code de sortie peut être passé au processus appelant dans le registre **al**. La signification exacte de ce code de sortie est entièrement de votre ressort. Ce code de sortie peut être examiné avec la commande "IF ERROR LEVEL return code" du DOS dans un fichier batch. Tous les fichiers ouverts par le processus courant seront automatiquement fermés à l'arrêt du programme.

Notez que la fonction "**ExitPgm**" de la Bibliothèque Standard UCR est simplement une macro qui fait appel à cette fonction DOS-même. C'est la manière normale de renvoyer le contrôle de nouveau à MS-DOS ou à un autre programme qui ont lancé l'application actuellement active.

13.3.7.2 Terminate and stay resident

Fonction (**ah**): 31h

Paramètres d'entrée: **al**- code de sortie
dx- taille de la mémoire (en paragraphes)
Paramètres de sortie: Ne retourne pas à votre programme.

Cette fonction termine également l'exécution du programme, mais lors du retour au DOS, la mémoire utilisée par le processus n'est pas retournée à la réserve de mémoire libre du DOS. Essentiellement, le programme demeure en mémoire. Des programmes qui demeurent résidents en mémoire après retour au DOS sont souvent appellés TSRs (termate and stay resident).

Quand cette commande est exécutée, le registre **dx** contient le nombre de paragraphes de mémoire à laisser en mémoire. Cette valeur est mesurée à partir du commencement du « préfixe de segment de programme », un segment marquant le début de votre fichier dans la mémoire. L'adresse du PSP (préfixe de segment de programme) est passée à votre programme dans le registre **ds** quand votre programme vient d'être lancé. Vous doivrez sauvegarder cette valeur si votre programme est un TSR[6].

Les programmes qui se terminent et restent résident ont besoin de fournir un certain mécanisme pour redémarrer. Une fois qu'ils reviennent au DOS, ils ne peuvent pas normalement être redémarrés. La plupart des TSRs détournent un des vecteurs d'interruption (tels qu'un vecteur d'interruption clavier, imprimante, ou série) afin de redémarrer chaque fois qu'un certain événement relié au matériel se produit (comme quand une touche est pressée). C'est ainsi que les programmes "pop-up" comme SmartKey fonctionnent.

Généralement, les programmes TSR sont des "pop-ups" ou des pilotes de périphérique spécifiques. Le mécanisme TSR fournit une moyen commode de charger vos propres routines pour remplacer ou augmenter des routines du BIOS. Votre programme se charge en mémoire, modifie le vecteur d'interruption approprié de façon à pointer sur un gestionnaire d'interruption interne à votre code, et puis se termine et reste résident. Maintenant, quand l'instruction d'interruption appropriée est exécutée, votre code sera appellé au lieu de celui du BIOS.

Il y a beaucoup trop de détails au sujet des TSRs, y compris des problèmes de compatibilité, des problèmes de ré-entrance du DOS et le traitement des interruptions, pour qu'ils soient être considérés ici. Des détails supplémentaires apparaîtront dans un chapitre ultérieur.

13.3.7.3 Exécuter un programme

Fonction (**ah**): 40h

Paramètres d'entrée: **ds:dx**- pointeur sur le nom et le chemin du programme à exécuter
es:bx- pointeur sur le bloc de paramètres
al- 0=charge et exécute, 1=charge seulement, 3=charge un overlay (ajout)
Paramètres de sortie: Si la retenue est à 1, **ax**. contient un de codes d'erreur suivants:

1- fonction non valide

[6] Le DOS fournit également un appel qui renvoie le PSP de votre program

2- fichier non trouvés
5- accès refusé
8- pas assez de mémoire
10- mauvais environnement
11- mauvais format

La fonction d'exécution (**exec**) est une opération extrêmement complexe, mais en même temps, très utile. Cette commande vous permet de charger ou charger et exécuter un programme situé sur le lecteur de disques. À l'entrée dans la fonction **exec**, les registres **ds:dx** contiennent un pointeur sur une chaîne terminée par zéro contenant le nom du fichier à charger ou à exécuter, **es:bx** un pointeur sur un bloc de paramètres, et **al** contient zéro ou un selon que vous voulez charger et exécuter un programme ou le charger seulement dans la mémoire. Au retour, si la retenue est à zéro, alors le DOS a correctement exécuté la commande. Si le drapeau de retenue est à un, alors le DOS a rencontré une erreur en exécutant la commande.

Le paramètre nom de fichier peut être un nom de chemin complet comprenant les informations de lecteur et de sous-répertoire. "B:\DIR1\DIR2\MYPGM.EXE" est un nom de fichier parfaitement valide (rappelez-vous, cependant, il doit être terminé par zéro). L'adresse segmentée de ce nom est passée dans les registres **ds:dx**.

Les registres **es:bx** pointent sur un bloc de paramètres lors de l'appel à **exec**. Ce bloc de paramètre prend trois formes différentes selon qu' un programme est chargé et exécuté (**al**=0), seulement chargé en mémoire (**al**=1), ou chargé comme overlay (**al**=3).

Si **al**=0, l'appel à **exec** charge et exécute un programme. Dans ce cas les registres **es:bx** pointent sur un bloc de paramètres contenant les valeurs suivantes:

Déplacement	Description
0	Une valeur word contenant l'adresse de segment de l'environnement par défaut (habituellement ceci est placé à zéro, ce qui implique l'utilisation de l'environnement standard du DOS).
2	Un pointeur double word contenant l'adresse de segment d'une chaîne ligne de commande.
6	Pointeur double mot sur le FCB par défaut à l'adresse 5Ch.
0Ah	Pointeur double mot sur le FCB par défaut à l'adresse 6Ch.

La zone d'environnement est un ensemble de chaînes contenant des chemins par défaut et toute autre information (ces informations sont fournies par le DOS en utilisant les commandes DOS PATH, SET etc.). Si cette entrée de paramètre contient zéro, alors **exec** transmettra l'environnement standard du DOS au nouveau processus. S'il est différent de zéro, alors ce paramètre contient l'adresse du segment du bloc d'environnement que votre processus transmet au programme à exécuter. Généralement, vous mettrez zéro à cette adresse.

Le pointeur sur la chaîne de commande devrait contenir l'adresse segmentée d'une chaîne préfixée par sa longueur et également terminée par un caractère de retour de chariot (le caractère de retour de chariot n'est pas compté dans la longueur de la chaîne). Cette chaîne correspond aux données qui sont normalement tapées après le nom du programme sur la ligne de commande DOS. Par exemple, si vous exécutez l'éditeur de liens automatiquement, vous pourriez passer une chaîne de commande de la forme suivante:

```
CmdStr          byte       16,"MyPgm+Routines /m", 0dh
```

Le deuxième élément dans le bloc de paramètre doit contenir l'adresse segmentée de cette chaîne.

Les troisième et quatrièmes éléments dans le bloc de paramètre pointent sur les FCBs par défaut. Les FCBs sont employés par les commandes de fichiers obsolètes du DOS, ainsi ils sont rarement employés dans des programmes d'application modernes. Puisque les structures de données sur lesquelles pointent ces deux pointeurs sont rarement employées, vous pouvez les faire pointer sur un groupe de 20 zéros.

Exemple: Formatez une disquette dans le lecteur A: en utilisant la commande FORMAT.EXE

```
mov    ah, 4Bh
mov    al, 0
mov    dx, seg PathName
mov    ds, dx
```

```
                    lea    dx, PathName
                    mov    bx, seg ParmBlock
                    mov    es, bx
                    lea    bx, ParmBlock
                    int    21h
  .
  .
  .
  PathName          byte   'C:\DOS\FORMAT.EXE',0
  ParmBlock   word  0                  ;Environnement par défaut
  dword             CmdLine            ;Chaîne de ligne de commande
  dword             Dummy,Dummy        ;Faux FCBs

  CmdLine           byte   3,' A:',0dh
  Dummy             byte   20 dup (?)
```

Les versions MS-DOS avant 3.0 ne préservent aucun registre à part **cs:ip** quand vous exécutez l'appel à **exec**. En particulier, **ss:sp** n'est pas préservé. Si vous utilisez DOS v2.x ou plus ancienne, vous aurez besoin d'employer le code suivant:

```
;Exemple: Formater une disquette dans le lecteur A: en utilisant
;la commande FORMAT.EXE

              <push any registers you need preserved>

              mov    cs:SS_Save, ss      ; Sauve SS:SP à un emplacement
              mov    cs:SP_Save, sp      ; où nous accéderons plus tard.
              mov    ah, 4Bh             ; opcode EXEC du DOS.
              mov    al, 0               ; Charge et exécute.
              mov    dx, seg PathName    ; Obtient le fichier dans DS:DX.
              mov    ds, dx
              lea    dx, PathName
              mov    bx, seg ParmBlock   ; Fait pointer ES:BX sur le
              mov    es, bx              ; bloc de paramètres.
              lea    bx, ParmBlock
              int    21h
              mov    ss, cs:SS_Save      ; Restaure SS:SP depuis
              mov    sp, cs:SP_Save      ; les emplacements de stockage

              <Restaure les registres poussés sur la pile>
.
.
.
SS_Save       word ?
SP_Save       word ?
.
.
.
PathName      byte          'C:\DOS\FORMAT.EXE',0
ParmBlock     word          0                        ;Environment par défaut
              dword  CmdLine                  ;Ligne de commande
              dword  Dummy,Dummy         ;Faux FCBs
CmdLine       byte          3,' A:',0dh
Dummy  byte           20 dup (?)
```

SS_Save et SP_Save doivent être déclarés à l'intérieur de votre segment de code. Les autres variables peuvent être déclarées n'importe où.

La commande **exec** alloue automatiquement la mémoire pour le programme à exécuter. Si vous n'avez pas libéré la mémoire inutilisée avant d'exécuter cette commande, vous pouvez obtenir une erreur insuffisance de mémoire. Par conséquent, vous devriez utiliser la commande DOS libérer de la mémoire pour libérer la mémoire inutilisée avant d'essayer d'employer la commande **exec**.

Si al=1 quand la fonction **exec** s'exécute, DOS chargera le fichier indiqué mais ne l'exécutera pas. Cette fonction est généralement employée pour charger un programme à exécuter dans la mémoire mais pour en donner le contrôle à

l'appelant et lui laisser le soin de démarrer ce code. Quand cet appel de fonction est fait, **es:bx** pointe sur le bloc de paramètres suivant :

Déplacement	Description
0	Une valeur word contenant l'adresse de segment du bloc d'environnement du nouveau processus. Si vous voulez employer le bloc d'environnement du processus parent mettez ce mot à zéro.
2	Un pointeur double word sur la queue de commande pour cette opération. La queue de commande est la chaîne de ligne de commande qui apparait à l'emplacement PSP:80 (voir "Le Préfixe de Segment de Programme (PSP)" à la section 13.3.11 et "Accéder aux Paramètres de la Ligne de Commande "à la section 13.3.12).
6	Adresse du FCB #1 par défaut . Pour la plupart des programmes, il devrait pointer sur un bloc de 20 zéros (à moins, bien sur, que vous lanciez un programme qui utilise les FCBs).
0Ah	Adresse du FCB # 2 par défaut. Devrait également pointer sur un bloc de 20 zéros
0Eh	Valeur de **ss:sp**. Vous devez charger ces quatre bytes dans **ss** et **sp** avant de démarrer l'application.
12h	Valeur de **cs:ip**. Ces quatre bytes contiennent l'adresse de début du programme.

Les champs SS:SP et CS:IP sont des valeurs de sortie. Le DOS remplit les champs et les renvoie dans la structure de chargement. Les autres champs sont toutes les entrées que vous devez remplir avant d'appeler la fonction exec avec **al**=1.

Quand vous exécutez la commande **exec** avec **al**=-3, le DOS charge simplement un *overlay* (sous-programme) dans la mémoire. Les overlays se composent généralement d'un seul segment de code qui contient quelques fonctions que vous voulez exécuter. Puisque vous ne créez pas un nouveau processus, le bloc de paramètre pour ce type de chargement est beaucoup plus simple que pour les deux autres types d'opérations de chargement. À l'entrée, **es:bx** doit pointer sur le bloc de paramètre suivant dans la mémoire :

Valeur excentrée de mot de la description 0 contenant l'adresse de segment d'où ce fichier va être chargé en la mémoire.

Déplacement	Description
0	Une valeur word contenant l'adresse de segment où ce fichier va être chargé en mémoire. Le fichier sera chargé à l'offset zéro dans ce segment.
2	Une valeur word contenant un facteur de relocalisation pour ce fichier.

À la différence des fonctions charger et exécuter, la fonction overlay n'alloue pas automatiquement d'espace de stockage pour le fichier chargé. Votre programme doit allouer suffisamment de stockage et ensuite passer l'adresse de ce bloc de stockage à la commande **exec** (par le bloc de paramètres ci-dessus). Seule l'adresse de segment de ce bloc est passée à la commande **exec**, l'offset est toujours censé être zéro. Le facteur de relocalisation devrait également contenir l'adresse de segment pour les fichiers ".EXE". Pour les fichiers ".COM", le paramètre de facteur de relocalisation devrait être zéro..

La commande overlay est très utile pour charger des overlays en mémoire à partir du disque. Un overlay est un segment de code qui réside sur le lecteur de disques jusqu'à ce que le programme ait réellement besoin d'exécuter son code. Alors, le code est chargé en la mémoire et exécuté. Les overlays peuvent réduire la quantité de mémoire que votre programme utilise en vous permettant de réutiliser la même partie de la mémoire pour différentes procédures en overlay (évidemment, une seule de ces procédures peut être active à un moment donné). En plaçant le code rarement-utilisé et le code d'initialisation dans des fichiers overlay, vous pouvez aider à réduire la quantité de mémoire employée par votre fichier de programme. Un conseil de prudence, cependant, la gestion des overlays est une tâche très complexe. Ce n'est pas quelque chose qu'un programmeur en assembleur débutant devrait aborder tout de suite. En chargeant un fichier dans la mémoire (au contraire de charger et exécuter un fichier), le DOS ne bouleverse pas tous les registres, ainsi vous n'avez pas besoin de prendre le soin spécifique de préserver **ss:sp** et d'autres registres.

L'encyclopédie de MS-DOS contient une excellente description de l'utilisation de la fonction **exec**.

13.3.8 Les "Nouveaux" appels de gestion de fichiers de MS-DOS

A partir de DOS v2.0, Microsoft a introduit un ensemble de procédures de gestion de fichiers qui (enfin) ont rendu accès aux fichier des disques supportable sous MS-DOS. Non seulement supportable, mais effectivement facile à utiliser ! Les sections suivantes décrivent l'utilisation de ces commandes pour accéder à des fichiers sur un lecteur de disques.

Les commandes de fichiers qui ont rapport avec les noms de fichier (Créer, Ouvrir, Supprimer, Renommer et autres) se voient passer l'adresse d'un nom de chemin terminé par zéro. Celles qui ouvrent effectivement un fichier (Créer et Ouvrir) renvoient un "handle" de fichier comme résultat (en supposant, naturellement, qu'il n'y a pas eu d'erreur). Ce handle de fichier est utilisé avec d'autres appels (Lire, Écrire, Chercher, Fermer, etc...) pour accéder au fichier que vous avez ouvert. À cet égard, un handle de fichier n'est pas très différent d'une variable de fichier en Pascal. Considérez le code suivant en Turbo Pascal/Microsoft:

```
program DemoFiles; var F:TEXT;

begin
      assign(f,'FileName.TXT');
      rewrite(f);
      writeln(f,'Hello there');
      close(f);
end.
```

La variable de fichier "f" est employée plus ou moins de la même façon dans cet exemple en Pascal qu'un handle de fichier dans un programme en assembleur - pour accéder au fichier qui a été créé dans le programme.

Toutes les commandes DOS de gestion de fichiers suivantes renvoient un état d'erreur dans le drapeau de retenue. Si le drapeau de retenue est à zéro quand le DOS revient à votre programme, alors l'opération a été accomplie avec succès. Si ce drapeau est à un au retour, alors une erreur quelconque s'est produite et le registre AX contient le numéro d'erreur. On parlera des valeurs de retour d'erreur, avec chaque fonction, dans les sections suivantes.

13.3.8.1 Ouvrir un fichier

Fonction (**ah**): 3Dh

Paramètres d'entrée: **al** - valeur d'accès au fichier

- 0- Fichier ouvert pour lecture
- 1- Fichier ouvert pour écriture
- 2- Fichier ouvert pour lecture et écriture

ds:dx- pointe sur une chaîne terminée par zéro contenant le nom de fichier

Paramètres de sortie: Si la retenue est à 1, **ax**. contient un de codes d'erreur suivants:

- 2- Fichier non trouvé
- 4- Trop de fichiers ouverts
- 5- Accès refusé
- 12- Accès non valide

Si la retenue est à zéro, **ax** contient la valeur de handle de fichier assignée par le DOS.

Un fichier doit être ouvert avant que vous puissiez y accéder. La commande ouvrir ouvre un fichier qui existe déjà. Ceci la rend tout à fait semblable à la procédure Reset du Pascal. Essayer d'ouvrir un fichier qui n'existe pas produit une erreur. Exemple :

```
lea    dx, Filename ;Suppose  que DS pointe sur le

mov    ah, 3dh             ; segment du nom de fichier
mov    al, 0        ;Ouvre en lecture.
int    21h
jc     OpenError
mov    FileHandle, ax
```

Si une erreur se produit en ouvrant un fichier, le fichier ne sera pas ouvert. Vous devriez toujours vérifier la présence d'une erreur après exécution d'une commande ouvrir du DOS, puisque continuer à travailler sur le fichier qui n'a pas été correctement ouvert produira des conséquences désastreuses. La manière précise dont vous traitez une erreur

d'ouverture dépend de vous, mais au moins, vous devriez afficher un message d'erreur et donner à l'utilisateur la chance d'indiquer un nom de fichier différent.

Si la commande ouvrir s'exécute sans produire d'erreur, le DOS renvoie un handle de fichier pour ce fichier dans le registre **ax**. Normalement, vous devriez sauver cette valeur quelque part pour pouvoir l'utiliser plus tard pour accéder au fichier.

13.3.8.2 Créer un fichier

Fonction (**ah**): 3Ch

Paramètres d'entrée: **ds:dx**- Adresse d'un nom de chemin terminé par zéro
cx- Attribut de fichier

Paramètres de sortie: Si la retenue est à 1, **ax**. contient un de codes d'erreur suivants:

3- Chemin non trouvé
4- Trop de fichiers ouverts
5- Accès refusé

Si la retenue est à zéro, **ax** contient le handle de fichier.

Créer ouvre un nouveau fichier pour traitement. Comme avec la commande OPEN, ds:dx pointe sur une chaîne terminée par zéro contenant le nom du fichier. Puisque cet appel crée un nouveau fichier, DOS présume que vous ouvrez le fichier en écriture seule. Un autre paramètre, passé dans cx, est la disposition initiale des attributs du fichier. Les six bits de L.O. (les moins significatifs) de **cx** contiennent les valeurs suivantes:

Bit	Signification si égal à un
0	Le fichier est en lecture-seule
1	Le fichier est un fichier caché
2	Le fichier est un fichier système
3	Le fichier est un nom d'étiquette de volume
4	Le fichier est un sous-répertoire
5	Le fichier a été archivé

En général, vous ne devriez positionner aucun de ces bits. La plupart des fichiers normaux devraient être créés avec un attribut de fichier de zéro. Par conséquent, le registre cx devrait être chargé avec zéro avant d'appeler la fonction Créer.

En sortie, le drapeau de retenue est à un si une erreur se produit. L'erreur "Chemin non trouvé" mérite une explication supplémentaire. Cette erreur est produite, non pas si le fichier n'est pas trouvé (ce qui serait le cas la plupart du temps puisque cette commande est normalement utilisée comme moyen de créer un nouveau fichier), mais si un sous-répertoire dans le nom de chemin n'est pas trouvé.

Si le drapeau de retenue est à zéro quand le DOS revient à votre programme, alors le fichier a été correctement ouvert pour le traitement et le registre **ax** contient la handle de fichier pour ce fichier.

13.3.8.3 Fermer un fichier

Fonction (**ah**): 3Eh

Paramètres d'entrée: **bx**- Handle de fichier

Paramètres de sortie: Si la retenue est à 1, **ax**. contient 6, la seule erreur possible, c'est à dire l'erreur handle non valide

Cet appel est employé pour fermer un fichier ouvert avec les commandes Ouvrir ou Créer ci-dessus. On lui passe la handle de fichier dans le registre **bx** et, à supposer que la handle de fichier soit valide, il ferme le fichier indiqué.

Vous devriez fermer tous les fichiers que votre programme utilise dès que vous avez terminé avec eux pour éviter la corruption de fichiers sur le disque au cas où l'utilisateur met le système hors tension ou reboote la machine alors que vos fichiers sont ouverts.

Notez que quitter pour retourner au DOS (ou en qittant le programme par la commande Ctrl-C ou Ctrl-Pause) ferme automatiquement tous les fichiers ouverts. Cependant, vous devriez ne jamais compter sur cette disposition puisque procéder ainsi est une pratique de programmation extrêmement peu rigoureuse.

13.3.8.4 Lire un fichier

Fonction (**ah**): 3Fh

Paramètres d'entrée: **bx**- Handle de fichier
cx- Nombre d'octets à lire
ds:dx- Tampon assez grand pour contenir les octets lus

Paramètres de sortie: Si la retenue est à 1, **ax**. contient un de codes d'erreur suivants:

6- Accès refusé
7- Handle non valide

Si la retenue est à zéro, **ax** contient le nombre d'octets effectivements lus dans le fichier.

La fonction Lire est utilisée pour lire un certain nombre de bytes à partir d'un fichier. Le nombre réel de bytes est indiqué par le registre cx à l'entrée dans le DOS. La handle de fichier, qui indique le fichier dont les bytes doivent être lus, est passée dans le registre **bx**. Le registre **ds:dx** contient l'adresse d'un tampon dans lequel les bytes lus à partir du fichier doivent être stockés.

Au retour, s'il n'y a pas eu d'erreur, le registre **ax** contient le nombre de bytes réellement lus. À moins que la fin du fichier (EOF: *End Of File*) ait été atteinte, ce nombre correspondra à la valeur passée au DOS dans le registre **cx**. Si la fin du fichier a été atteinte, la valeur retournée dans ax sera quelque part entre zéro et la valeur passée au DOS dans le registre **cx**. *C'est le seul test pour la condition EOF.*

Exemple: cet exemple ouvre un fichier et le lit jusqu'à EOF

```
            mov   ah, 3dh                 ;Ouvre le fichier

            mov   al, 0             ;Ouvre en lecture
            lea   dx, Filename      ;Suppose que DS pointe sur le
            int   21h                     ; segment du nom du fichier
            jc    BadOpen
            mov   FHndl, ax               ;Enregistrement de handle du fichier
LP:         mov   ah,3fh                  ;Lit les données du fichier
            lea   dx, Buffer        ;Adresse du tampon de données
            mov   cx, 1             ;Lit un byte
            mov   bx, FHndl               ;Obtient la handle du fichier
            int   21h
            jc    ReadError
            cmp   ax, cx                  ;EOF atteinte ?
            jne   EOF
            mov   al, Buffer        ;Obtient le caractère lu
            putc                          ;L'affiche
            jmp   LP                      ;Lit le byte suivant
EOF:        mov   bx, FHndl
            mov   ah, 3eh                 ;Ferme le fichier
            int   21h
            jc    CloseError
```

Ce segment de code lira le fichier entier dont le nom de fichier (terminé par zéro) est trouvé à l'adresse "Filename" dans le segment de données courant et écrira chaque caractère du fichier sur le périphérique de sortie standard en utilisant la routine **putc** de la StdLib d'UCR. Soyez prévenu qu'une E/S caractère par caractère de ce type est extrêmement lente. Nous discuterons de meilleures manières de lire rapidement un fichier un peu plus tard dans ce chapitre.

13.3.8.5 Écrire dans un fichier

Fonction (**ah**): 40h

Paramètres d'entrée: **bx**- Handle de fichier
cx- Nombre d'octets à écrire

ds:dx- Tampon contenant les données à écrire

Paramètres de sortie: Si la retenue est à 1, **ax**. contient un de codes d'erreur suivants:

5- Accès refusé
6- Handle non valide

Si la retenue est à zéro, **ax** contient le nombre d'octets effectivements écrits dans le fichier.

Cet appel est presque l'inverse de la commande Lire présentée plus tôt. Il écrit le nombre de bytes indiqué dans **ds:dx** dans le fichier au lieu de les lire. Au retour, si le nombre de bytes écrits au fichier n'est pas égal au nombre indiqué à l'origine dans le registre **cx**, le disque est plein et ceci devrait être traité comme une erreur.

Si **cx** contient zéro quand cette fonction est appellée, le DOS tronquera le fichier à la position courante dans le fichier (c.-à-d., toutes les données suivant la position courante dans le fichier seront supprimées).

13.3.8.6 Rechercher (déplacer le pointeur de fichier)

Fonction (**ah**): 42h

Paramètres d'entrée: **al-** Méthode de déplacement

0- Le déplacement est spécifié depuis le début du fichier
1- Le déplacement est spécifié depuis le pointeur de fichier courant
2- Le pointeur est déplacé à la fin du fichier moins l'offset spécfié.

bx- Handle de fichier

cx:dx- Distance du déplacement en octets

Paramètres de sortie: Si la retenue est à 1, **ax**. contient un de codes d'erreur suivants:

1- Fonction non valide
6- Handle non valide

Si la retenue est à zéro, **ds:ax** contient la nouvelle position dans le fichier.

Cette commande est utilisée comme moyen de déplacer le pointeur de fichier dans un fichier à accès aléatoire. Il y a trois méthodes de déplacement du pointeur de fichier, une distance absolue dans le fichier (si **al**=0), une certaine distance positive depuis la position courante dans le fichier (si **al**=1), ou une certaine distance depuis la fin du fichier (si **al**=2). Si **al** ne contient ni 0, ni 1, ni 2, le DOS renverra une erreur fonction non valide. Si cet appel accompli est avec succès, la lecture ou l'écriture du prochain byte se produira à l'emplacement indiqué.

Notez que le DOS traite **cx:dx** comme un nombre entier non signé. Par conséquent, une commande Rechercher seule ne peut pas être utilisée pour se déplacer en l'arrière dans le fichier. Au lieu de cela, la méthode #0 doit être utilisée pour placer le pointeur de fichier à une position absolue dans le fichier à déterminer. Si vous ne savez pas où vous êtes actuellement et voulez reculer de 256 bytes, vous pouvez utiliser le code suivant:

```
mov    ah, 42h             ;Commande Rechercher
mov    al, 1         ;Depuis le pointeur courant
xor    cx, cx              ;Met à zéro CX et DX pour
xor    dx, dx              ; rester sur place
mov    bx, FileHandle
int    21h
jc     SeekError
sub    ax, 256             ;DX:AX contiennent maintenant
sbb    dx, 0         ; la position courante, ainsi
mov    cx, dx              ; calcule une position 256
mov    dx, ax              ; bytes en arrière.
mov    ah, 42h
mov    al, 0         ;Position absolue dans le fichier
int    21h                 ;BX contient toujours la handle.
```

13.3.8.7 Positionner l'adresse de transfert sur disque (DTA)

Fonction (**ah**): 1Ah

Paramètres d'entrée: **ds:dx**- Pointeur sur la DTA

Paramètres de sortie: Aucun

Cette commande est appellée "Positionner DTA" parce qu'elle était (est) utilisée avec les fonctions originales de fichier de DOS v1.0. Normalement, nous n'examinerions pas cette fonction si ce n'était qu'elle est également utilisée par les fonctions 4Eh et 4Fh (décrites ci-dessous) pour installer un pointeur sur une zone-tampon de 43 bytes. Si cette fonction n'est pas exécutée avant d'exécuter les fonctions 4Eh ou 4Fh, le DOS utilisera l'espace tampon par défaut situé à PSP:80h.

13.3.8.8 *Trouver premier fichier*

Fonction (**ah**): 4Eh
Paramètres d'entrée: **cx**- Attributs
ds:dx- Pointeur sur le nom de fichier
Paramètres de sortie: Si la retenue est à 1, **ax**. contient un de codes d'erreur suivants:
2- Fichier non trouvé
18- Plus de fichiers

Les fonctions *Trouver premier fichier* et *Trouver fichier suivant* (décrite ci-dessous) sont utilisées pour rechercher des fichiers indiqués grâce à des références de fichier ambiguës. Une référence de fichier ambiguë est tout nom de fichier contenant les caractères joker "*"et "?". La fonction *Trouver premier fichier* est utilisée pour localiser le premier fichier avec un tel nom dans un répertoire indiqué, la fonction *Trouver fichier suivant* est utilisée pour trouver les entrées successives dans le répertoire.

Généralement, quand une référence de fichier ambiguë est fournie, la commande *Trouver premier fichier* est lancée pour localiser la première occurrence du fichier, et ensuite une boucle est utilisée, qui appelle *Trouver fichier suivant*, pour localiser toutes autres occurrences du fichier dans cette boucle jusqu'à ce qu'il n'y ait plus de fichiers (erreur # 18). Chaque fois que *Trouver premier fichier* est appellée, elle installe l'information suivante dans la DTA:

Offset	Description
0	Réservé à l'usage de *Trouver premier fichier*
21	Attribut du fichier trouvé
22	Heure de création du fichier
24	Date de création du fichier
26	Taille du fichier en bytes
30	Nom du fichier et extension (terminé par zéro)

(Les offsets sont décimaux)

A supposer que *Trouver premier fichier* ne renvoie quelque erreur, le nom du premier fichier correspondant à la description ambiguë de fichier apparaîtra à l'offset 30 dans le DTA.

Note: si le chemin spécifié ne contient n'importe pas de caractères joker, alors *Trouver premier fichier* renverra le nom de fichier spécifié exact, s'il existe. Tout appel consécutif à *Trouver fichier suivant* renverra une erreur.

Le registre **cx** contient les attributs à rechercher pour le fichier. Normalement, **cx** devrait contenir zéro. Si différent de zéro, *Trouver premier fichier* (et *Trouver fichier suivant*) inclura les noms de fichier qui ont les attributs spécifiés aussi bien que tous les fichiers normaux.

13.3.8.9 *Trouver fichier suivant*

Fonction (**ah**): 4Fh
Paramètres d'entrée: Aucun
Paramètres de sortie: Si la retenue est à 1, il n'y a plus de fichiers et **ax**. retournera avec le code 18.

La fonction *Trouver fichier suivant* est utilisée pour rechercher des noms de fichier supplémentaires correspondant à une référence de fichier ambiguë après un appel à *Trouver premier fichier*. La DTA doit pointer sur un enregistrement de données établi par la fonction *Trouver premier fichier*.

Exemple: Le code suivant liste les noms de tous les fichiers dans l'annuaire courant qui finissent par ".EXE". On suppose que la variable "DTA" est dans le segment de données courant:

```
mov    ah, 1Ah                    ;Installe la DTA
lea    dx, DTA
```

```
            int    21h
            xor    cx, cx                    ;Pas d'attributs.
            lea    dx, FileName
            mov    ah, 4Eh                   ;Trouver premier fichier
            int    21h
            jc     NoMoreFiles         ;Si erreur, terminé
DirLoop:    lea    si, DTA+30          ;Adresse du nom de fichier
            cld
PrtName:    lodsb
            test   al, al                    ;Byte zéro?
            jz     NextEntry
            putc                             ;Affiche ce caractère
            jmp    PrtName
NextEntry:  mov    ah, 4Fh                   ;Trouver fichier suivant
            int    21h
            jnc    DirLoop                   ;Affiche ce nom
```

13.3.8.10 *Detruire fichier*

Fonction (**ah**): 41h
Paramètres d'entrée: **ds:dx**- Adresse du chemin du fichier à détruire
Paramètres de sortie: Si la retenue est à 1, **ax**. contient un de codes d'erreur suivants:

2- Fichier non trouvé
5- Accès refusé

Cette fonction supprimera le fichier indiqué du répertoire. Le nom de fichier doit être un nom de fichier non ambigu (c.-à-d., il ne peut contenir aucun caractère joker).

13.3.8.11 *Renommer fichier*

Fonction (**ah**): 56h
Paramètres d'entrée: **ds:dx**- Adresse du chemin d'un fichier existant
es:di-Pointeur sur le nouveau nom
Paramètres de sortie: Si la retenue est à 1, **ax**. contient un de codes d'erreur suivants:

2- Fichier non trouvé
5- Accès refusé
17- Pas le même support

Cette commande a deux fonctions: elle vous permet de renommer un fichier en autre chose et elle vous permet de déplacer un fichier d'un répertoire à un autre (tant que les deux sous-répertoires sont sur le même disque).

```
Exemple #1: Renommer "MYPGM.EXE" en "YOURPGM.EXE"


 ; Supposez que ES et DS pointent les deux sur le segment de données
; courant contenant le nom du fichier.

            lea    dx, OldName
            lea    di, NewName
            mov    ah, 56h
            int    21h
            jc     BadRename
            .
            .
            .
OldName     byte   "MYPGM.EXE",0       ; nota: il y avait des "pretty quotes" dans
NewName     byte   "YOURPGM.EXE",0     ; le texte original, nous les avons
                                       ; conservées, ndt

Example #2: Déplacer un fichier d'un répertoire à un autre:

; Supposez que ES et DS pointent les deux sur le segment de données
```

```
; courant contenant le nom du fichier.

                lea     dx, OldName
                lea     di, NewName
                mov     ah, 56h
                int     21h
                jc      BadRename
                .
                .
                .
OldName         byte    "\DIR1\MYPGM.EXE",0
NewName         byte    "\DIR2\MYPGM.EXE",0
```

13.3.8.12 Changer/obtenir les attributs de fichier

Fonction (**ah**): 43h
Paramètres d'entrée: **al**- Code de sous-fonction
- 0- Renvoie les attributs de fichier dans **cx**
- 1- Change les attributs en ceux de **cx**

cx- Valeur des attributs si AL=01
ds:dx- Adresse du chemin du fichier

Paramètres de sortie: Si la retenue est à 1, **ax**. contient un de codes d'erreur suivants:
- 1- Fonction non valide
- 3- Chemin/fichier non trouvé
- 5- Accès refusé

Si le drapeau de retenue est à zéro et si la sous-fonction était zéro, **cx** contiendra les attributs du fichier

Cet appel est utile pour spécifier/changer et lire les bits d'attribut d'un fichier. Il peut être utilisé pour placer un fichier en lecture seule, positionner/effacer le bit d'archivage ou sinon à bidouiller les attributs de fichier.

13.3.8.13 Spécifier/obtenir la date et l'heure du fichier

Fonction (**ah**): 57h
Paramètres d'entrée: **al**- Code de sous-fonction
- 0- Obtenir la date et l'heure
- 1- Spécifier la date et l'heure

bx- Handle du fichier
cx- Heure à indiquer (si AL=01)
dx- Date à indiquer (si AL=01)

Paramètres de sortie: Si la retenue est à 1, **ax**. contient un de codes d'erreur suivants:
- 1- Fonction non valide
- 6- Handle non valide

Si le drapeau de retenue est à zéro, **cx/dx** contiendra la date/heure si **al** =00

Cet appel spécifie la date/heure de "last-write" pour le fichier indiqué. Le fichier doit être ouvert (utiliser Ouvrir ou Créer) avant d'utiliser cette fonction. La date ne sera pas enregistrée jusqu'à ce que le fichier soit fermé.

13.3.8.14 Les autres appels DOS

Les tables suivantes énumèrent brièvement plusieurs des autres appels du DOS. Pour plus d'information sur l'utilisation de ces fonctions DOS consultez la Référence du Programmeur MS-DOS ou la *Référence technique de MS-DOS de Microsoft*.

Tableau 56: autres fonctions de gestion des fichiers du DOS

# Fonction (AH)	Paramètres en entrée	Paramètres en sortie	Description

39h	**ds:dx**- pointeur sur un nom de chemin terminé par zéro.		Créer un répertoire : Crée un nouveau répertoire avec le nom indiqué.
3Ah	**ds:dx**- pointeur sur un nom de chemin terminé par zéro.		Oter un répertoire : Supprime le répertoire du nom indiqué. Erreur si le répertoire n'est pas vide ou si le répertoire indiqué est le répertoire courant.
3Bh	**ds:dx**- pointeur sur un nom de chemin terminé par zéro.		Changer de répertoire : Remplace le répertoire par défaut par celui du nom indiqué.
45h	**bx**- handle de fichier	**ax**- nouvelle handle	Reproduire une handle de fichier : crée une copie d'une handle de fichier si bien qu'un programme peut accéder à un fichier en utilisant deux variables séparées de fichier. Ceci permet au programme de fermer le fichier avec une handle et de continuer à lui accéder avec l'autre.
46h	**bx**- handle de fichier **cx**- double de la handle		Forcer la reproduction d'une handle de fichier : Comme la fonction 45h ci-dessus, sauf que vous indiquez à quelle handle (dans **cx**) vous voulez vous référer pour le fichier existant (indiqué par **bx**).
47h	**ds:si**- pointeur sur un tampon **dl**- lecteur		Obtenir le répertoire courant : Stocke une chaîne contenant le nom du chemin courant (terminé par un zéro) commençant à l'emplacement **ds:si**. Ces registres doivent pointer sur un tampon contenant au moins 64 bytes. Le registre **dl** indique le numéro de lecteur (0=défaut, 1=A, 2=B, 3=C, etc.).
5Ah	**cx**- attributs **ds:dx**- pointeur sur un chemin temporaire.	**ax**- handle	Créer un fichier temporaire : Crée un fichier avec un nom unique dans le répertoire indiqué par la chaîne terminée par zéro sur laquelle pointe **ds:dx**. Il doit y avoir au moins 13 bytes à zéro après la fin du nom parce que cette fonction stockera le nom de fichier produit à la fin du nom. Les attributs sont identiques que pour la fonction Créer.
5Bh	**cx**- attributs **ds:dx**- pointeur sur un nom de chemin terminé par zéro	**ax**- handle	Créer un nouveau fichier : Comme l'appel Créer, mais cet appel insiste sur le fait que le fichier n'existe pas. Il renvoie une erreur si le fichier existe (au lieu de supprimer le vieux fichier).
67h	**bx**- handle		Spécifier le nombre maximum de handles : Cette fonction spécifie le nombre maximum de handles qu'un programme peut utiliser à un moment donné.
68h	**bx**- handle		Assurer un fichier : Copie sur le disque toutes les données dans un fichier sans le fermer, s'assurant que les données du fichier sont actualisées et cohérentes

Tableau 57: autres fonctions du DOS

# Fonction (AH)	Paramètres en entrée	Paramètres en sortie	Description
25h	**al**- n° d'interruption **ds:dx**- pointeur sur une routine de service d'interruption.		Spécifier un vecteur d'interruption : Stocke l'adresse indiquée dans **ds:dx** dans la table de vecteurs d'interruption à l'entrée indiquée par le registre **al**.

30h		**bx**- version majeure **ah-** version mineure **bh-** drapeau de version **bl:cx-** numéro de série sur 24 bits	Obtenir le Numéro de version : Renvoie le numéro de version courant du DOS (ou la valeur spécifiée par SETVER).
33h	**al**- 0	**dl**- break flag (0=inactivé, 1= activé)	Obtenir le drapeau de pause (Break Flag) : Renvoie l'état du break flag du DOS. Si activé, MS-DOS vérifie Ctrl-C lors du traitement de toute commande DOS; si inactivé, MS-DOS le vérifie seulement pour les fonctions 1-0Ch.
33h	**al**- 1 **dl**- break flag		Activer le break flag : Met le break flag de MS-DOS à la valeur dans **dl** (voir la fonction ci-dessus pour les détails).
33h	**al**- 6	**bl**- version majeure **bh-** version mineure **dl-** révision **dh-** drapeaux de version	Obtenir la version de MS-DOS : Renvoie le "vrai" numéro de versio, pas celui spécifié par la commande SETVER. Les bits trois et quatre des drapeaux de version sont à un si le DOS est dans la ROM ou si le DOS est dans la mémoire haute, respectivement..
34h		**es:bx**- pointeur sur le drapeau InDOS	Obtenir l'adresse du drapeau InDOS : Renvoie l'adresse du drapeau InDOS. Ce drapeau aide à empêcher la réentrance dans les applications TSR.
35h	**al**- n° d'interruption	**ds:dx**- pointeur sur une routine de service d'interruption.	Obtenir l'adresse d'un vecteur d'interruption : Renvoie un pointeur sur la routine de service d'interruption pour le numéro d'interruption indiqué. Voir la fonction 25h ci-dessus pour plus de détails.
44h	**al**- n° de sous-fonction D'autres paramètres!		Contrôle de périphérique : Voici toute une famille de commandes DOS supplémentaires pour contrôler divers périphériques. Voyez le manuel de référence du programmeur DOS pour plus de détails.
4Dh		**al**- valeur de retour **ah-** méthode de terminaison	Obtenir la valeur de retour d'un programme fils : Renvoie le dernier code de retour d'un programme enfant dans **al**. Le registre **ah** contient la méthode de terminaison, qui est une des valeurs suivantes: 0- normal, 1- Ctrl-C, 2- erreur critique de périphérique, 3- terminate and stay resident.
50h	**bx**- adresse du PSP		Spécifier l'adresse du PSP : Met l'adresse du PSP courant du DOS à la valeur spécifiée dans le registre **bx**.
51h		**bx**- adresse du PSP	Obtenir l'adresse du PSP :Renvoie un pointeur sur le PSP courant dans le registre **bx**.
59h		**ax**- erreur étendue **bh-** classe d'erreur **bl-** action d'erreur **ch-** emplacement de l'erreur	Obtenir l'erreur étendue : Renvoie des informations supplémentaires quand une erreur survient dans un appel au DOS. Voyez le manuel de référence du programmeur DOS pour plus de détails sur ces erreurs et comment les traiter.
5Dh	**al**- 0Ah **ds:si**- pointeur sur une structure d'erreur étendue		Spécifier une erreur étendue : copie les données de la structure d'erreur étendue dans l'enregistrement interne du DOS.

En plus des commandes ci-dessus, il y a plusieurs appels additionnels du DOS qui traitent des réseaux et des jeux de caractères internationaux. Voyez la référence de MS-DOS pour plus de détails.

13.3.9 Exemples d'E/S de fichier

Naturellement, une des raisons principales de faire des appels au DOS est de manipuler des fichiers sur un périphérique de mémoire de masse. Les exemples suivants démontrent quelques utilisations d'E/S par caractère à l'aide du DOS.

13.3.9.1 Exemple n°1: un utilitaire de vidage hexadécimal

Ce programme vide un fichier en format hexadécimal. Le nom de fichier doit être codé en dur dans le fichier (voir "Accéder aux paramètres de ligne de commande" plus loin dans ce chapitre).

```
                    include     stdlib.a
                    includelib stdlib.lib

cseg                segment     byte public 'CODE'
                    assume      cs:cseg, ds:dseg, es:dseg, ss:sseg

MainPgm             proc        far

; Met en place les registres de segment:

                    mov    ax, seg dseg
                    mov    ds, ax
                    mov    es, ax
                    mov    ah, 3dh
                    mov    al, 0        ;Ouvre le fichier en lecture
                    lea    dx, Filename ;Fichier à ouvrir
                    int    21h
                    jnc    GoodOpen
                    print
                    byte   'Cannot open fichier, aborting program...',cr,0
                    jmp    PgmExit

GoodOpen:           mov    FileHandle, ax      ;Sauve la handle de fichier
                    mov    Position, 0  ;Initialise compteur position
ReadFileLp:  mov    al, byte ptr Position
                    and    al, 0Fh             ;Calcule (Position MOD 16)
                    jnz    NotNewLn            ;Nouvelle ligne tous les 16 oct
                    putcr
                    mov    ax, Position ;Imprime offset dans fichier
                    xchg   al, ah
                    puth
                    xchg   al, ah
                    puth
                    print
                    byte   ': ',0

NotNewLn:           inc    Position            ;Incremente compteur caractères
                    mov    bx, FileHandle
                    mov    cx, 1        ;Lit un octet
                    lea    dx, buffer   ;Endroit où stocker cet octet
                    mov    ah, 3Fh             ;Opération de lecture
                    int    21h
                    jc     BadRead
                    cmp    ax, 1        ;EOF atteinte?
                    jnz    AtEOF
                    mov    al, Buffer   ;Obtient le caractère lu et
                    puth                       ; l'imprime en hexadécimal
                    mov    al, ' '             ;Imprime espace entre valeurs
                    putc
                    jmp    ReadFileLp
```

```
BadRead:            print
                    byte   cr, lf
                    byte   'Error reading data from fichier, aborting'
                    byte   cr,lf,0

AtEOF:              mov    bx, FileHandle      ;Ferme le fichier
                    mov    ah, 3Eh
                    int    21h

PgmExit:            ExitPgm
MainPgm             endp

cseg                ends
dseg                segment      byte public 'data'

Filename            byte         'hexdump.asm',0           ;Nom fichier traité
FileHandle   word          ?
Buffer              byte         ?
Position            word         0

dseg                ends

sseg                segment      byte stack 'stack'
stk                 word         0ffh dup (?)
sseg                ends

zzzzzzseg           segment para public 'zzzzzz'
LastBytes           byte         16 dup (?)
zzzzzzseg           ends
                    end          MainPgm
```

13.3.9.2 Exemple n°2: conversion en majuscules

Le programme suivant lit un fichier, convertit toutes les minuscules en majuscules, et écrit les données dans un deuxième fichier en sortie.

```
                    include             stdlib.a
                    includelib   stdlib.lib

cseg                segment      byte public 'CODE'
                    assume       cs:cseg, ds:dseg, es:dseg, ss:sseg

MainPgm             proc         far

; Met en place les registres de segment:

                    mov    ax, seg dseg
                    mov    ds, ax
                    mov    es, ax
;-----------------------------------------------------------------
;
; Convertit UCCONVRT.ASM en majuscules
;
; Ouvre le fichier en entrée:

                    mov    ah, 3dh
                    mov    al, 0        ;Ouvre fichier en lecture
                    lea    dx, Filename ;Fichier à ouvrir
                    int    21h
                    jnc    GoodOpen
                    print
                    byte   'Cannot open fichier, aborting program...',cr,lf,0
                    jmp    PgmExit

GoodOpen:           mov    FileHandle1, ax     ;Sauve handle fichier d'entrée
```

```
; Ouvre le fichier en sortie:

                    mov    ah, 3Ch             ;Appel à Créer fichier
                    mov    cx, 0        ;Attributs de fichier normaux
                    lea    dx, OutFileName ;Fichier à ouvrir
                    int    21h
                    jnc    GoodOpen2
                    print
                    byte   'Cannot open output fichier, aborting program...'
                    byte   cr,lf,0
                    mov    ah, 3eh             ;Ferme fichier en entrée
                    mov    bx, FileHandle1
                    int    21h
                    jmp    PgmExit             ;Ignore les erreurs.

GoodOpen2:   mov    FileHandle2, ax     ;Sauve handle fichier en sortie

ReadFileLp:  mov    bx, FileHandle1
                    mov    cx, 1        ;Lit un octet
                    lea    dx, buffer   ;Endroit où stocker cet octet
                    mov    ah, 3Fh             ;Opération de lecture
                    int    21h
                    jc     BadRead
                    cmp    ax, 1        ;EOF atteinte?
                    jz     ReadOK
                    jmp    AtEOF

ReadOK:             mov    al, Buffer   ;Obtient le caractère lu et
                    cmp    al, 'a'             ; le convertit en majuscule
                    jb     NotLower
                    cmp    al, 'z'
                    ja     NotLower
                    and    al, 5fh             ;Met le Bit #5 à zéro.
NotLower:           mov    Buffer, al

; Maintenant écrit les données dans le fichier en sortie

                    mov    bx, FileHandle2
                    mov    cx, 1        ;Lit un octet
                    lea    dx, buffer   ;Endroit où stocker cet octet
                    mov    ah, 40h             ;Opération d'écriture
                    int    21h
                    jc     BadWrite
                    cmp    ax, 1        ;S'assure disque pas plein
                    jz     ReadFileLp

BadWrite:           print
                    byte   cr, lf
                    byte   'Error writing data, aborting operation'
                    byte   cr,lf,0
                    jmp    short AtEOF

BadRead:            print
                    byte   cr, lf
                    byte   'Error reading data from fichier, aborting '
                    byte   'operation',cr,lf,0

AtEOF:              mov    bx, FileHandle1     ;Ferme le fichier
                    mov    ah, 3Eh
                    int    21h
                    mov    bx, FileHandle2
                    mov    ah, 3eh
                    int    21h

;-----------------------------------------------------------------

PgmExit:            ExitPgm
MainPgm             endp
```

```
cseg                ends

dseg                segment      byte public 'data'

Filename            byte         'ucconvrt.asm',0    ;Fichier à convertir
OutFileName  byte          'output.txt',0            ;Fichier en sortie
FileHandle1  word          ?
FileHandle2  word          ?
Buffer              byte         ?
Position            word         0

dseg                ends

sseg                segment      byte stack 'stack'
stk                 word         0ffh dup (?)
sseg                ends

zzzzzzseg           segment      para public 'zzzzzz'
LastBytes           byte         16 dup (?)
zzzzzzseg           ends
                    end          MainPgm
```

13.3.10 E/S de fichier bloquées

Les exemples dans la section précédente souffrent d'un inconvénient important, ils sont extrêmement lents. Les problèmes d'exécution avec le code ci-dessus sont entièrement dus au DOS. Faire un appel au DOS n'est pas, dirons-nous, l'opération la plus rapide au monde. Appeler le DOS chaque fois que nous voulons lire ou écrire un caractère unique sur un fichier mettra le système à genoux. Tant qu'à faire, cela ne prend (pratiquement) pas plus de temps pour faire écrire ou lire deux caractères au DOS que lui faire lire ou écrire un caractère. Puisque la quantité de temps que nous dépensons (généralement) pour traiter les données est négligeable comparé à la quantité de temps que le DOS prend pour retourner ou écrire les données, lire deux caractères à la fois va effectivement doubler la vitesse du programme. Si lire deux caractères double la vitesse de traitement, qu'en est-il de lire quatre caractères? Assez sûrement, cela quadruple presque la vitesse de traitement. De même le traitement de dix caractères à la fois augmente presque la vitesse de traitement d'un ordre de grandeur. Hélas, cette progression ne continue pas toujours. Il arrive un point de rentabilité moindre - quand cela prend beaucoup trop de mémoire pour justifier une (très) petite amélioration) de l'exécution (en gardant à l'esprit que la lecture de 64K en une opération unique exige d'un tampon de mémoirede de 64K pour contenir les données). Un bon compromis est 256 ou 512 bytes. Lire plus de données n'améliore pas vraiment l'exécution, cependant il est plus facile traiter un tampon de 256 ou 512 bytes que de plus gros tampons.

Lire des données en groupes ou blocs s'appelle *E/S bloquées*. Les E/S bloquées sont souvent un à deux ordres de grandeur plus rapides que les E/S de caractère unique, aussi évidemment vous devriez utiliser les E/S bloquées chaque fois que possible.

Il y a un inconvénient mineur aux E/S bloquées -- elles sont plus complexes à programmer que les E/S par caractère unique. Considérez l'exemple présenté dans la section sur la commande Lire du DOS:

Cet exemple ouvre un fichier et le lit jusqu'à l'EOF

```
                    mov    ah, 3dh              ;Ouvre le fichier
                    mov    al, 0         ;Ouvre en lecture
                    lea    dx, Filename  ;Suppose que DS pointe sur
                    int    21h                  ; le segment du nom de fichier
                    jc     BadOpen
                    mov    FHndl, ax            ;Sauve la handle du fichier

LP:                 mov    ah,3fh               ;Lit données du fichier
                    lea    dx, Buffer    ;Adresse du buffer de données
                    mov    cx, 1         ;Lit un octet
                    mov    bx, FHndl            ;Obtient valeur handle fichier
                    int    21h
                    jc     ReadError
                    cmp    ax, cx               ;EOF atteinte?
                    jne    EOF
                    mov    al, Buffer    ;Obtient caractère lu
```

```
                    putc                        ;L'affiche (appel IOSHELL)
                    jmp     LP                  ;Lit octet suivant
EOF:                mov     bx, FHndl
                    mov     ah, 3eh             ;Ferme fichier
                    int     21h
                    jc      CloseError
```

Il n'y a pas grand chose dans ce programme. Considérez maintenant le même exemple réécrit pour utiliser les E/S bloquées:

Exemple: Cet exemple ouvre un fichier et le lit jusqu'à l'EOF en utilisant les E/S bloquées

```
                    mov     ah, 3dh             ;Ouvre le fichier
                    mov     al, 0         ;Ouvre en lecture
                    lea     dx, Filename  ;Suppose que DS pointe sur
                    int     21h                 ; le segment du nom de fichier
                    jc      BadOpen
                    mov     FHndl, ax           ;Sauve la handle du fichier

LP:                 mov     ah,3fh              ;Lit données du fichier
                    lea     dx, Buffer    ;Adresse du buffer de données
                    mov     cx, 256             ;Lit 256 bytes
                    mov     bx, FHndl           ;Obtient valeur handle fichier
                    int     21h
                    jc      ReadError
                    cmp     ax, cx              ;EOF atteinte?
                    jne     EOF
                    mov     si, 0         ;Note: CX=256 à cet endroit.
PrtLp:              mov al, Buffer[si]  ;Obtient caractère lu
                    putc                        ;L'affiche
                    inc     si
                    loop    PrtLp
                    jmp     LP                  ;Lit bloc suivant

; Note, ce n'est pas parce que le nombre de bytes lus n'égale pas
; 256, que nous avons pour autant fini, il pourrait y avoir jusqu'à
; 255 bytes dans le buffer en attente de traitement.

EOF:                mov     cx, ax
                    jcxz    EOF2          ;Si CX est zéro, nous avons fini.
                    mov     si, 0  ;Traite le dernier bloc de données
Finis:              mov     al, Buffer[si]      ; lu du fichier qui contient
                    putc                ; 1..255 bytes de données valides.
                    inc     si
                    loop    Finis

EOF2:       mov     bx, FHndl
                    mov     ah, 3eh ; ;Ferme fichier.
                    int     21h
                    jc      CloseError
```

Cet exemple démontre un défaut majeur des E/S bloquées - quand vous atteignez la fin du fichier, vous n'avez pas nécessairement traité toutes les données dans le fichier. Si la longueur de bloc est 256 et qu'il y a 255 bytes qui restent dans le fichier, le DOS renverra la condition EOF (le nombre de bytes lus ne correspond pas à la demande). Dans ce cas, il nous reste toujours à traiter les caractères qui ont été lus. Le code ci-dessus fait ceci d'une façon plutôt directe, en utilisant une deuxième boucle pour finaliser quand EOF est atteinte. Vous avez probablement remarqué que les deux boucles d'impression sont pratiquement identiques. Ce programme peut être légèrement réduit en taille en employant le code suivant qui est seulement un peu plus complexe:

Exemple: Cet exemple ouvre un fichier et le lit jusqu'à l'EOF en utilisant les E/S bloquées

```
                    mov     ah, 3dh             ;Ouvre le fichier
                    mov     al, 0         ;Ouvre en lecture
                    lea     dx, Filename  ;Suppose que DS pointe sur
                    int     21h                 ; le segment du nom de fichier
                    jc      BadOpen
                    mov     FHndl, ax           ;Sauve la handle du fichier
```

```
LP:                 mov    ah,3fh              ;Lit données du fichier
                    lea    dx, Buffer    ;Adresse du buffer de données
                    mov    cx, 256             ;Lit 256 bytes
                    mov    bx, FHndl           ;Obtient valeur handle fichier
                    int    21h
                    jc     ReadError
                    mov    bx, ax              ;Sauve pour plus tard
                    mov    cx, ax
                    jcxz   EOF
                    mov    si, 0         ;Note: CX=256 à cet endroit.
PrtLp:              mov al, Buffer[si]  ;Obtient caractère lu
                    putc                       ;L'affiche
                    inc    si
                    loop   PrtLp
                    cmp    bx, 256             ;EOF déjà atteinte?
                    je     LP
EOF:                mov    bx, FHndl
                    mov    ah, 3eh ; ;Ferme fichier.
                    int    21h
                    jc     CloseError
```

Les E/S bloquées marchent mieux sur les fichiers séquentiels. C'est-à-dire, les fichiers ouverts seulement pour la lecture ou l'écriture (sans recherches). pour traiter les fichiers à accès aléatoire, vous devriez lire ou écrire des enregistrements entiers à la fois à l'aide des commandes DOS lire/écrire pour traiter l'enregistrement en totalité. Ce sera toujours beaucoup plus rapide que manipuler les données un byte à la fois.

13.3.11 Le préfixe de segment de programme (PSP)

Quand un programme est chargé en la mémoire pour exécution, DOS construit d'abord un préfixe de segment de programme immédiatement avant de charger le programme en la mémoire. Ce PSP contient un bon nombre d'informations, en partie utiles, en partie obsolètes. La connaissance de la disposition du PSP est essentielle pour des programmeurs concevant des programmes en assembleur.

Le PSP a 256 bytes de long et contient les informations suivantes :

Offset	Longueur	Description
0	2	Une instruction INT 20h est stockée ici
2	2	Adresse de fin de programmes
4	1	Inutilisé, réservé par le DOS
5	5	Appel au gestionnaire de fonction du DOS
0Ah	4	Adresse du code de terminaison du programme
0Eh	4	Adresse de la routine du gestionnaire d'interruption de programme
12h	4	Adresse de la routine du gestionnaire d'erreur critique
16h	22	Réservé à l'usage du DOS
2Ch	2	Adresse du segment de la zone d'environnement
2Eh	34	Réservé par le DOS
50h	3	Instructions INT 21h, RETF
53h	9	Réservé par le DOS
5Ch	16	FCB #1 par défaut
6Ch	20	FCB #2 par défaut
80h	1	Longueur de la chaîne de la ligne de commande
81h	127	Chaîne de la ligne de commande

Note: les emplacements 80h..FFh sont utilisés pour le DTA par défaut.

La majeure partie des information dans le PSP est peu utile à un programme moderne MS-DOS en assembleur. Enterrées dans le PSP, cependant, se trouvent quelques gemmes qu'il vaut la peine de connaître. Pour ne rien ommettre, cependant, nous passerons en revue tous les champs du PSP.

Le premier champ dans le PSP contient une instruction **int 20h**. **int 20h** est un mécanisme obsolète utilisé pour terminer l'exécution du programme. A l'époque des premiers jours du DOS v1.0, votre programme exécutait un jmp à

cet emplacement afin de se terminer. De nos jours, naturellement, nous avons la fonction 4Ch du DOS qui est beaucoup plus facile (et plus sûre) que sauter à l'emplacement zéro du PSP. Par conséquent, ce champ est obsolète.

Le champ le numéro deux contient une valeur qui pointe sur le dernier paragraphe alloué à votre programme. En soustrayant l'adresse du PSP de cette valeur, vous pouvez déterminer la quantité de mémoire allouée à votre programme (et quitter s'il n'y a pas suffisamment de mémoire disponible).

Le troisième champ est le premier des beaucoup de "trous" laissés dans le PSP par Microsoft. Pourquoi ils se trouvent ici est laissé à votre imagination.

Le quatrième champ est un appel au gestionnaire de fonction du DOS. La fonction de ce mécanisme d'appel du DOS (maintenant) obsolète était de permettre quelque compatibilité supplémentaire avec les programmes CP/M-80. Pour des programmes DOS modernes, il n'y a absolument aucun besoin de s'occuper de ce champ.

Les trois champs suivants sont utilisés pour stocker des adresses spéciales pendant l'exécution d'un programme. Ces champs contiennent le vecteur de terminaison par défaut, le vecteur d'interruption de programme et le vecteur du gestionnaire d'erreur critique. Ce sont les valeurs normalement stockées dans les vecteurs d'interruption pour **int 22h**, **int 23h** et **int 24h**. En stockant une copie des valeurs dans les vecteurs pour ces interruptions, vous pouvez changer ces vecteurs de sorte qu'ils pointent sur votre propre code. Quand votre programme se termine, le DOS reconstitue ces trois vecteurs à partir de ces trois champs dans le PSP. Pour plus de détails sur ces vecteurs d'interruption, consultez svp le manuel de référence technique du DOS.

Le huitième champ dans l'enregistrement du PSP est un autre champ réservé, actuellement indisponible à l'usage de vos programmes.

Le neuvième champ est un autre vrai joyau. C'est l'adresse de la zone des chaînes d'environnement. C'est un pointeur à deux octets qui contient l'adresse de segment de la zone de stockage de l'environnement. Les chaînes d'environnement commencent toujours à un offset zéro dans ce segment. Le secteur de chaîne d'environnement se compose d'une suite de chaînes terminées par zéro. Il emploie le format suivant :

$chaîne_1$ 0 $chaîne_2$ 0 $chaîne_3$ 0... 0 $chaîne_n$ 0 0

C'est-à-dire, le secteur d'environnement se compose d'une liste de chaînes terminées zéro, la liste elle-même terminée par une chaîne de longueur zéro (c.-à-d., un zéro tout seul, ou deux zéros de suite, selon le point de vue). Les chaînes (habituellement) sont placées dans le secteur d'environnement par l'intermédiaire de commandes DOS comme PATH, SET, etc.. Généralement, une chaîne dans le secteur d'environnement prend la forme

nom = paramètres

Par exemple, la commande "SET IPATH=C:\ASSEMBLY\INCLUDE" copie la chaîne "IPATH=C:\ASSEMBLY\ INCLUDE" dans la zone de stockage de chaîne d'environnement.

Beaucoup de langues balayent la zone de stockage d'environnement pour trouver des noms de chemins de fichier par défaut et d'autres informations par défaut installées par le DOS. Vos programmes peuvent aussi tirer profit de ceci.

Le champ suivant du PSP est un autre bloc de stockage réservé, actuellement non défini par le DOS.

Le 11ème champ dans le PSP est un autre appel au gestionnaire de fonctions du DOS. Pourquoi cet appel existe (alors que celui à l'emplacement 5 dans le PSP existe déjà et que personne n'emploie vraiment l'un ou l'autre mécanisme pour appeler le DOS) est une question intéressante. En général, ce champ devrait être ignoré par vos programmes.

Le 12ème champ dans le PSP est un autre bloc de bytes inutilisés qui devrait être ignoré.

Les 13ème et 14ème champs dans le PSP sont les FCBs (blocs de commande de fichier) par défaut. Les blocs de commande de fichier sont une autre structure de données archaïque importée de CP/M-80. Les FCBs sont utilisés seulement avec les routines de gestions de fichiers obsolètes de DOS v1.0, ainsi ils sont de peu d'intérêt pour nous. Nous ignorerons ces FCBs dans le PSP.

Les emplacements 80h jusqu'à la fin du PSP contiennent une information très importante - les paramètres de ligne de commande tapés sur la ligne de commande du DOS avec le nom de votre programme. Si ce qui suit est tapé sur la ligne de commande du DOS :

```
MYPGM parameter1, parameter2
```

ce qui suit est stocké dans le champ des paramètres de ligne de commande :

```
23, " parameter1, parameter2", 0Dh
```

L'emplacement 80h contient 23_{10}, la longueur des paramètres suivant le nom du programme. Les emplacements 81h à 97h contiennent les caractères composant la chaîne des paramètres. L'emplacement 98h contient un retour de chariot. Notez que le caractère de retour de chariot n'est pas compté dans la longueur de la chaîne de ligne de commande.

Le traitement de la de ligne de commande est une facette si importante de la programmation assembleur que ce processus sera discuté en détail dans la prochaine section.

Les emplacements 80h..FFh dans le PSP comportent également le DTA par défaut. Par conséquent, si vous n'utilisez pas la fonction 1Ah du DOS pour changer le DTA et si vous exécutez Trouver Premier Fichier, l'information sur le nom de fichier sera stockée à partir de l'emplacement 80h dans le PSP.

Un détail important que nous avons omis jusqu'ici est la manière dont vous accédez exactement à des données dans le PSP. Bien que le PSP soit chargé en mémoire juste avant votre programme, cela ne veut pas dire pour autant qu'il apparaît 100h bytes avant votre code. Vos segments de données ont pu avoir été chargés en mémoire avant vos segments de code, annulant de ce fait cette méthode pour localiser le PSP. L'adresse de segment du PSP est passée à votre programme dans le registre **ds**. Pour stocker l'adresse du PSP dans votre segment de données, vos programmes devraient commencer par le code suivant:

```
push   ds                  ;Sauve la valeur du PSP
mov    ax, seg DSEG ;Fait pointer DS et ES sur
mov    ds, ax              ; notre segment de données.
mov    es, ax
pop    PSP                 ;Stocke la valeur du PSP dans
                           ; la variable "PSP".
.
.
.
```

Une autre manière d'obtenir l'adresse du PSP, dans DOS 5.0 et plus tard, est de faire un appel DOS. Si vous chargez **ah** avec 51h et exécutez une instruction **int 21h**, le MS-DOS renverra l'adresse de segment du PSP courant dans le registre **bx**.

Il y a un bon nombre de bidouilles que vous pouvez faire avec les données dans le PSP. Le Guide du Programmeur du PC d'IBM de Peter Norton énumère toutes sortes d'astuces. De telles opérations ne seront pas discutées ici parce qu'elles sortent un peu du cadre de ce livre.

13.3.12 Accès aux paramètres de la ligne de commande

La plupart des programmes comme MASM et LINK vous permettent d'indiquer des paramètres de ligne de commande quand le programme est exécuté. Par exemple, en tapant

```
ML MYPGM.ASM
```

vous pouvez demander à MASM d'assembler MYPGM sans autre intervention du clavier. "MYPGM.ASM;" est un bon exemple de paramètre de ligne de commande.

Quand l'interprèteur de commande du DOS, COMMAND.COM, analyse votre ligne de commande, il copie la majeure partie du texte après le nom du programme à l'emplacement 80h dans le PSP comme décrit dans la section précédente. Par exemple, la ligne de commande ci-dessus stockera ce qui suit à PSP:80h

```
11, " MYPGM.ASM", 0Dh
```

Le texte stocké dans la zone de stockage de queue de ligne de commande dans le PSP est habituellement une copie exacte des données apparaissant sur la ligne de commande. Il y a, cependant, deux exceptions. Tout d'abord, des paramètres de redirection d'E/S ne sont pas stockés dans le tampon d'entrée. Il en est de même des queues de commande suivant l'opérateur pipe ("|"). L'autre chose apparaissant sur la ligne de commande qui est absente des données à PSP:80h est le nom du programme. C'est plutôt malheureux, puisqu'avoir le nom du programme disponible vous permettrait de déterminer le répertoire contenant le programme. Néanmoins, il y a un bon nombre d'informations utiles présentes sur la ligne de commande.

Les informations sur la ligne de commande peuvent être utilisées pour à peu près tout ce que vous pouvez estimer utile. Cependant, la plupart des programmes attendent deux types de paramètres dans tampon des paramètres de la

ligne de commande - des noms de fichier et des commutateurs. La fonction d'un nom de fichier est plutôt évidente, cela permet à un programme d'accéder à un fichier sans devoir demander à l'utilisateur le nom du fichier. Les commutateurs, d'autre part, sont des paramètres arbitraires du programme. Par convention, des commutateurs sont précédés par une barre oblique ou un trait d'union dans la ligne de commande.

Trouver ce qu'il y a à faire avec les informations sur la ligne de commande s'appelle *parsing* (analyse) de la ligne de commande. Clairement, si vos programmes doivent manipuler des données sur la ligne de commande, vous devrez analyser la ligne de commande dans votre code.

Avant qu'une ligne de commande puisse être analysée, chaque élément de la ligne de commande doit être isolé des autres. C'est-à-dire, chaque mot (ou plus correctement, chaque lexème[7]) doit être identifié dans la ligne de commande. La séparation des lexèmes sur une ligne de commande est relativement facile, tout ce qu'il vous faut faire, c'est rechercher des séquences de délimiteurs sur la ligne de commande. Les délimiteurs sont des symboles spéciaux utilisés pour séparer les éléments sur la ligne de commande. Le DOS supporte six caractères différents comme délimiteurs: l'espace, la virgule, le point-virgule, le signe égal, la tabulation ou le retour de chariot.

Habituellement, n'importe quel nombre de caractères délimiteur peuventt apparaître entre deux éléments sur une ligne de commande. Par conséquent, de telles occurrences doivent toutes être sautées lors du balayage de la ligne de commande. Le code suivant en assembleur balaye la ligne de commande toute entière et imprime tous les éléments qui y apparaissent:

```
                    include      stdlib.a
                    includelib stdlib.lib

cseg                segment      byte public 'CODE'
                    assume       cs:cseg, ds:dseg, es:dseg, ss:sseg

; Equates dans la ligne de commande-

CmdLnLen            equ    byte ptr es:[80h]   ;Longueur ligne commande
CmdLn        equ    byte ptr es:[81h]   ;Données ligne commande

tab                 equ    09h

MainPgm             proc   far

; Met en place les registres de segment:

                    push   ds                        ;Sauve le PSP
                    mov    ax, seg dseg
                    mov    ds, ax
                    pop    PSP

;-----------------------------------------------------------------

                    print
                    byte   cr,lf
                    byte   'Items on this line:',cr,lf,lf,0

                    mov    es, PSP             ;Fait pointer ES sur PSP
                    lea    bx, CmdLn           ;Pointe sur ligne de commande
PrintLoop:   print
                    byte   cr,lf,'Item: ',0
                    call   SkipDelimiters      ;Saute les délimiteurs de tête
PrtLoop2:           mov    al, es:[bx]  ;Obtient caractère suivant
                    call   TestDelimiter       ;Est-ce un délimiteur?
                    jz     EndOfToken   ;Quitte cette boucle si oui
                    putc                       ;Affiche caractère si non.
                    inc    bx                  ;Passe au caractère suivant
                    jmp    PrtLoop2

EndOfToken:  cmp    al, cr              ;Retour de chariot?
                    jne    PrintLoop           ;Recommence si pas fin de ligne

                    print
                    byte   cr,lf,lf
                    byte   'End of command line',cr,lf,lf,0
                    ExitPgm
MainPgm             endp
```

[7] Beaucoup de programmeurs emploient le mot "token" ("jeton") plutôt que lexème. Techniquement, un token est un élément différent.

```
; La routine suivante met le flag zéro à un si le caractère dans
; le registre AL est un des six caractères délimiteurs du DOS,
; autrement le flag zéro est renvoyé à zéro. Ceci permet d'utiliser
; par la suite les instructions JE/JNE pour tester la présence
; d'un délimiteur.

TestDelimiter       proc   near
                    cmp    al, ' '
                    jz     ItsOne
                    cmp    al,',
                    jz     ItsOne
                    cmp    al,Tab
                    jz     ItsOne
                    cmp    al,';'
                    jz     ItsOne
                    cmp    al,'='
                    jz     ItsOne
                    cmp    al, cr
ItsOne:             ret
TestDelimiter       endp

; SkipDelimiters saute les délimiteurs de tête sur la ligne de
; commande. Cependant, elle ne saute pas le retour de chariot à la
; fin d'une ligne car ce caractère est utilisé comme terminateur
; dans le programme principal.

SkipDelimiters      proc   near
                    dec    bx                ;Pour annuler INC BX ci-dessous
SDLoop:             inc    bx                ;Passe au caractère suivant.
                    mov    al, es:[bx]  ;Obtient le caractère suivant
                    cmp    al, 0dh           ;Ne saute pas si CR.
                    jz     QuitSD
                    call   TestDelimiter     ;Voit si il y a un autre
                    jz     SDLoop            ; délimiteur et recommence.
QuitSD:             ret
SkipDelimiters      endp

cseg                ends

dseg                segment      byte public 'data'

PSP                 word         ?            ;Program segment prefix
dseg                ends

sseg                segment      byte stack 'stack'
stk                 word         0ffh dup (?)
sseg                ends

zzzzzzseg           segment      para public 'zzzzzz'
LastBytes           byte         16 dup (?)
zzzzzzseg           ends
                    end          MainPgm
```

Une fois que vous pouvez balayer la ligne de commande (c'est-à-dire, séparer les lexèmes), l'étape suivante est de l'analyser. Pour la plupart des programmes, l'analyse de la ligne de commande ne pose aucun problème. Si le programme accepte seulement un nom de fichier unique, tout ce qu'il vous reste à faire est de saisir le premier lexème sur la ligne de commande, coller un octet zéro à la fin (peut-être le déplacer dans votre segment de données), et l'utiliser comme nom de fichier. L'exemple suivant en assembleur modifie la routine de vidage hexadécimal présentée plus tôt de sorte qu'elle obtienne son nom de fichier par la ligne de commande au lieu de coder en dur le nom de fichier dans le programme:

```
                    include      stdlib.a
                    includelib stdlib.lib
cseg                segment      byte public 'CODE'
                    assume       cs:cseg, ds:dseg, es:dseg, ss:sseg

; Notez que CR et LF sont déjà définis dans STDLIB.A

tab                 equ          09h

MainPgm             proc   far

; Met en place les registres de segment:
```

```
                    mov    ax, seg dseg
                    mov    es, ax              ;Laisse DS pointer sur PSP
;-------------------------------------------------------------
;
; D'abord, analyse la ligne de commande pour obtenir le nom de
; fichier:
                    mov    si, 81h             ;Pointeur sur ligne de commande
                    lea    di, FileName ;Pointeur sur tampon FileName
SkipDelimiters:
                    lodsb               ;Obtient caractère suivant
                    call   TestDelimiter
                    je     SkipDelimiters

; Suppose que ce qui suit est un véritable nom de fichier

                    dec    si                  ;Pointe sur 1er car du nom
GetFName:           lodsb
                    cmp    al, 0dh
                    je     GotName
                    call TestDelimiter
                    je     GotName
                    stosb               ;Sauve caractère
                    jmp    GetFName

; Nous sommes à la fin du nom de fichier, aussi terminons le par
; zéro comme le DOS le demande.

GotName:            mov    byte ptr es:[di], 0
                    mov    ax, es              ;Fait pointer DS sur DSEG
                    mov    ds, ax

; Maintenant traite le fichier

                    mov    ah, 3dh
                    mov    al, 0        ;Ouvre fichier en lecture
                    lea    dx, Filename ;Fichier à ouvrir
                    int    21h
                    jnc    GoodOpen
                    print
                    byte   'Cannot open fichier, aborting program...',cr,0
                    jmp    PgmExit
GoodOpen:           mov    FileHandle, ax      ;Sauve handle fichier
                    mov    Position, 0  ;Initialise position fichier
ReadFileLp:  mov    al, byte ptr Position
                    and    al, 0Fh             ;Calcule (Position MOD 16)
                    jnz    NotNewLn            ;Tous les 16 bytes début ligne
                    putcr
                    mov    ax, Position ;Imprime offset dans fichier
                    xchg   al, ah
                    puth
                    xchg   al, ah
                    puth
                    print
                    byte   ': ',0

NotNewLn:           inc    Position            ;Incremente compteur caractères
                    mov    bx, FileHandle
                    mov    cx, 1        ;Lit un byte
                    lea    dx, buffer   ;endroit où stocker ce byte
                    mov    ah, 3Fh             ;Opération de lecture
                    int    21h
                    jc     BadRead
                    cmp    ax, 1        ;EOF atteinte?
                    jnz    AtEOF
                    mov    al, Buffer   ;Obtient le caractère lu et
                    puth                       ; l'imprime en hexadécimal
                    mov    al, ' '             ;Imprime espace entre valeurs
                    putc
                    jmp    ReadFileLp
```

```
BadRead:            print
                    byte   cr, lf
                    byte   'Error reading data from fichier, aborting.'
                    byte   cr,lf,0

AtEOF:              mov    bx, FileHandle      ;Ferme le fichier
                    mov    ah, 3Eh
                    int    21h

;-----------------------------------------------------------------

PgmExit:            ExitPgm
MainPgm             endp

TestDelimiter       proc   near
                    cmp    al, ' '
                    je     xit
                    cmp    al, ','
                    je     xit
                    cmp    al, Tab
                    je     xit
                    cmp    al, ';'
                    je     xit
                    cmp    al, '='
xit:                ret
TestDelimiter       endp

cseg                ends

dseg                segment      byte public 'data'

PSP                 word         ?
Filename            byte         64 dup (0)    ;Nom fichier à vider
FileHandle   word          ?
Buffer              byte         ?
Position            word         0

dseg                ends

sseg                segment      byte stack 'stack'
stk                 word         0ffh dup (?)
sseg                ends

zzzzzzseg           segment      para public 'zzzzzz'
LastBytes           byte         16 dup (?)
zzzzzzseg           ends

                    end    MainPgm
```

L'exemple suivant démontre plusieurs concepts relatifs aux paramètres de la ligne de commande. Ce programme copie un fichier dans un autre. Si le commutateur "/U" est fourni (quelque part) sur la ligne de commande, toutes les minuscules dans le fichier sont converties en majuscules avant d'être écrites dans le fichier de destination. Un autre dispositif de ce code est qu'il demandera à l'utilisateur tout nom de fichier manquant, tout comme les programmes MASM et LINK vous demandent un nom de fichier si vous n'en avez pas fourni.

```
                    include      stdlib.a
                    includelib stdlib.lib
cseg                segment      byte public 'CODE'
                    assume       cs:cseg, ds:dseg, es:dseg, ss:sseg

; Notez que CR et LF sont déjà définis dans le fichier include
; stdlib.a

tab                 equ          09h

MainPgm             proc   far

; Met en place les registres de segment:

                    mov    ax, seg dseg
                    mov    es, ax              ;Laisse DS pointer sur PSP

;-----------------------------------------------------------------
;
```

```
; D'abord, analyse la ligne de commande pour obtenir le nom de
; fichier:

                    mov    es:GotName1, 0      ; Flags Init qui nous disent si
                    mov    es:GotName2, 0      ; nous avons analysé nom de
                    mov    es:ConvertLC,0      ; fichier et commutateur "/U"

; OK, commence à balayer et analyser la ligne de commande

                    mov    si, 81h             ;Pointeur sur ligne de commande

SkipDelimiters:

              lodsb                       ;Obtient caractère suivant
              call   TestDelimiter
              je     SkipDelimiters

; Détermine s'il s'agit d'un nom fichier ou du commutateur /U

              cmp    al, '/'
              jnz    MustBeFN

; Voit si c'est "/U" ici-

              lodsb
              and    al, 5fh                  ;Convertit "u" en "U"
              cmp    al, 'U'
              jnz    NotGoodSwitch
              lodsb                       ;Vérifie si caractère est
              cmp    al, cr                   ; un délimiteur quelconque
              jz     GoodSwitch
              call   TestDelimiter
              jne    NotGoodSwitch

; OK, c'est "/U" ici.

GoodSwitch:   mov    es:ConvertLC, 1     ;Convertit min en maj
                     dec    si                  ;Recule s'il s'agit de CR
                     jmp    SkipDelimiters      ;Passe à l'item suivant.

; Si un mauvais commutateur a été trouvé sur la ligne de commande,
; imprime un message d'erreur et termine-

NotGoodSwitch:
                    print
                    byte   cr,lf
                    byte   'Illegal switch, only "/U" is allowed!',cr,lf
                    byte   'Aborting program execution.',cr,lf,0
                    jmp    PgmExit

; Si ce n'est pas un commutateur, suppose qu'il s'agit d'un nom de
;fichier valide et passe le ici-

MustBeFN:           cmp    al, cr       ;Voit si fin de ligne de cmd
                    je     EndOfCmdLn

; Voit s'il s'agit d'un nom de fichier, de deux, ou si trop de noms
; de fichier  ont été spécifiés-

                    cmp    es:GotName1, 0
                    jz     Is1stName
                    cmp    es:GotName2, 0
                    jz     Is2ndName
; Plus de deux noms de fichiers ont été entrés, affiche un message
; d'erreur et termine.

                    print
                    byte   cr,lf
                    byte   'Too many filenames specified.',cr,lf
                    byte   'Program aborting...',cr,lf,lf,0
                    jmp    PgmExit

; Saute ici bas si c'est le premier fichier à être traité-

Is1stName:          lea    di, FileName1
                    mov    es:GotName1, 1
```

```
                    jmp    ProcessName

Is2ndName:   lea    di, FileName2
                    mov    es:GotName2, 1
ProcessName:

                    stosb               ;Stocke caractère dans le nom
                    lodsb               ;Obtient car suivant ligne cmd
                    cmp    al, cr
                    je     NameIsDone
                    call   TestDelimiter
                    jne    ProcessName

NameIsDone:  mov al, 0                  ;Nom fichier terminé par zéro
                    stosb
                    dec    si                   ;Pointe sur car précédent
                    jmp    SkipDelimiters       ;Réessaie.
; Quand la fin de la ligne de commande est atteinte, descendre ici
; et voir si les deux noms de fichier étaient indiqués.

                    assume       ds:dseg

EndOfCmdLn:  mov    ax, es              ;Fait pointer DS sur DSEG
                    mov    ds, ax
; Nous sommes à la fin du nom de fichier, aussi terminons le par
; zéro comme le DOS le demande.

GotName:            mov    ax, es              ;Fait pointer DS sur DSEG
                    mov    ds, ax

; Voit si les noms names ont été forurnis sur la ligne de commande.
; Si non, les demande à l'utilisateur et les lit sur le clavier

                    cmp    GotName1, 0  ;Nom fichier #1 fourni?
                    jnz    HasName1
                    mov    al, '1'              ;Filename #1
                    lea    si, Filename1
                    call   GetName              ;Obtient nom fichier #1

HasName1:           cmp    GotName2, 0  ; Nom fichier #2 fourni?
                    jnz    HasName2
                    mov    al, '2'              ;Si non, le lire sur clavier.
                    lea    si, FileName2
                    call   GetName

; OK, nous avons les noms de fichiers, maintenant ouvre les fichiers
; et copie le fichier source dans le fichier destination.

HasName2            mov    ah, 3dh
                    mov    al, 0        ;Ouvre fichier en lecture
                    lea    dx, Filename1       ;Fichier à ouvrir
                    int    21h
                    jnc    GoodOpen1

                    print
                    byte   'Cannot open fichier, aborting program...',cr,lf,0
                    jmp    PgmExit

; Si le fichiers source a été" ouvert avec succès, sauve la handle.

GoodOpen1:   mov    FileHandle1, ax           ;Sauve handle de fichier

; Ouvre (Crée, en fait) le second fichier ici.

                    mov    ah, 3ch                    ;Crée fichier
                    mov    cx, 0               ;Attributs standards
                    lea    dx, Filename2              ;Fichier à ouvrir
                    int    21h
                    jnc    GoodCreate

; Note: le code d'erreur code suivant se base sur le fait que le DOS
; ferme automatiquement tout fichier source ouvert quand le
; programme  se termine.

                    print
                    byte   cr,lf
```

```
                    byte   'Cannot create new fichier, aborting operation'
                    byte   cr,lf,lf,0
                    jmp    PgmExit

GoodCreate:  mov    FileHandle2, ax     ;Sauve handle de fichier

; Maintenant traite les fichiers

CopyLoop:           mov    ah, 3Fh             ;opcode de lecture du DOS
                    mov    bx, FileHandle1     ;Lit depuis fichier #1
                    mov    cx, 512             ;Lit 512 bytes
                    lea    dx, buffer   ;Tampon de stockage
                    int    21h
                    jc     BadRead
                    mov    bp, ax              ;Sauve # bytes lus

                    cmp    ConvertLC,0  ;Option conversion active?
                    jz     NoConversion

; Convertit toutes les minuscules du tampon en majuscules -

                    mov    cx, 512
                    lea    si, Buffer
                    mov    di, si
ConvertLC2UC:
                    lodsb
                    cmp    al, 'a'
                    jb     NoConv
                    cmp    al, 'z'
                    ja     NoConv
                    and    al, 5fh
NoConv:             stosb
                    loop   ConvertLC2UC

NoConversion:
                    mov    ah, 40h             ;Opcode d'écriture du DOS
                    mov    bx, FileHandle2     ;Écrit dans fichier #2
                    mov    cx, bp              ;Écrit tant de bytes
                    lea    dx, buffer   ;Tampon pour stockage
                    int    21h
                    jc     BadWrite
                    cmp    ax, bp              ;Avons-nous écrit tous les
                    jnz    jDiskFull           ; bytes?
                    cmp    bp, 512             ;A-t-on lu 512 bytes?
                    jz     CopyLoop
                    jmp    AtEOF
jDiskFull:   jmp    DiskFull

; Messages d'erreur variés:

BadRead:            print
                    byte   cr,lf
                    byte   'Error while reading source fichier, aborting '
                    byte   'operation.',cr,lf,0
                    jmp    AtEOF

BadWrite:           print
                    byte   cr,lf
                    byte   'Error writing destination fichier, aborting'
                    byte   ' operation.',cr,lf,0
                    jmp    AtEOF

DiskFull:           print
                    byte   cr,lf
                    byte   'Error, disk full. Aborting operation.',cr,lf,0

AtEOF:              mov bx, FileHandle1       ;Ferme le premier fichier
                    mov    ah, 3Eh
                    int    21h
                    mov    bx, FileHandle2 ;Ferme le second fichier
                    mov    ah, 3Eh
                    int    21h

PgmExit:            ExitPgm
```

```
MainPgm             endp

TestDelimiter       proc   near
                    cmp    al, ' '
                    je     xit
                    cmp    al, ','
                    je     xit
                    cmp    al, Tab
                    je     xit
                    cmp    al, ';'
                    je     xit
                    cmp    al, '='
xit:                ret
TestDelimiter endp

; GetName- Lit un nom de fichier sur le clavier. A l'entrée, AL
; contient le numéro de nom de fichier et DI pointe sur le tampon
; dans ES où le nom de fichier terminé par zéro doit être stocké.
; contains the filename number and DI points at the buffer in ES

ObtientName  proc   near
                    print
                    byte   'Enter filename #',0
                    putc
                    mov    al, ':'
                    putc
                    gets
                    ret
GetName             endp
cseg                ends

dseg                segment      byte public 'data'
PSP                 word         ?
Filename1           byte         128 dup (?);Source filename
Filename2           byte         128 dup (?);Destination filename
FileHandle1  word          ?
FileHandle2  word          ?
GotName1            byte         ?
GotName2            byte         ?
ConvertLC           byte         ?
Buffer              byte         512 dup (?)
dseg                ends

sseg                segment      byte stack 'stack'
stk                 word         0ffh dup (?)
sseg                ends

zzzzzzseg           segment      para public 'zzzzzz'
LastBytes           byte         16 dup (?)
zzzzzzseg           ends
                    end          MainPgm
```

Comme vous pouvez le voir, cela prend plus de travail de traiter les paramètres de la ligne de commande que de copier réellement les fichiers !

13.3.13 ARGC et ARGV

La Bibliothèque standard UCR fournit deux routines, **argc** et **argv**, qui fournissent un accès facile aux paramètres de la ligne de commande. **Argc** (compte d'arguments) renvoie le nombre d'éléments sur la ligne de commande. **Argv** (vecteur d'arguments) renvoie un pointeur sur un élément spécifique dans la ligne de commande.

Ces routines décomposent la ligne de commande en lexèmes en utilisant les délimiteurs standards. Selon les conventions de MS-DOS, **argc** et **argv** traitent toute chaîne sur la ligne de commande entourée par des guillemets comme élément unique de ligne de commande.

Argc retournera dans cx le nombre d'éléments de la ligne de commande. Puisque MS-DOS n'inclut pas le nom du programme dans la ligne de commande, ce compte n'inclut pas non plus ce nom. En outre, les opérandes de

redirection (">filename" et "<filename") et les items à la droite d'un pipe ("| command") n'apparaissent pas non plus sur la ligne de commande. Logiquement, **argc** ne les compte pas non plus.

Argv renvoie un pointeur sur une chaîne (allouée sur le tas) pour un élément spécifié d'une ligne de commande. Pour employer **argv** vous chargez simplement **ax** avec une valeur entre un et le nombre retourné par **argc** et exécutez la routine **argv**. Au retour, **es:di** pointe sur une chaîne contenant l'option indiquée de la ligne de commande. Si le nombre dans **ax** est plus grand que le nombre d'arguments de la ligne de commande, alors **argv** renvoie un pointeur sur une chaîne vide (c.-à-d., un byte zéro). Puisque **argv** appelle **malloc** pour assigner le stockage sur le tas, il est possible qu'une erreur d'attribution de mémoire se produise. **Argv** renvoie le flag de retenue à un si une erreur d'attribution de mémoire se produit. Pensez à libérer le stockage alloué à un paramètre de ligne de commande dès que vous avez terminé avec lui.

Exemple : Le code suivant renvoie un écho des paramètres de la ligne de commande à l'écran.

```
                    include     stdlib.a
                    includelib stdlib.lib

dseg                segment     para public 'data'

ArgCnt              word        0

dseg                ends

cseg                segment     para public 'code'
                    assume      cs:cseg, ds:dseg

Main                proc
                    mov   ax, dseg
                    mov   ds, ax
                    mov   es, ax

; Il faut appeler la routine d'initialization du gestionnaire de
; mémoire si vous utilisez des routines qui appellent malloc!
; ARGV est un bon exemple d'une routine qui appelle malloc.

                    meminit

                    argc                    ;Obtient le compte d'args.
                    jcxz Quit               ;Quit si pas d'args dans LdC.
                    mov ArgCnt, 1           ;Init compteur de la LdC.
PrintCmds:   printf                    ;Affiche l'élément.
                    byte   "\n%2d: ",0
                    dword ArgCnt

                    mov   ax, ArgCnt   ;Obtient le suivant dans la LdC
                    argv
                    puts
                    inc   ArgCnt            ;Passe à l'arg suivant.
                    loop  PrintCmds         ;Recommence pour chaque arg.
                    putcr

Quit:        ExitPgm                   ;Macro DOS pour quitter prog
Main                endp
cseg                ends

sseg                segment     para stack 'stack'
stk                 byte        1024 dup ("stack ")
sseg                ends

;zzzzzzseg est nécessaire pour les routines de la bib standard.

zzzzzzseg           segment     para public 'zzzzzz'
LastBytes           byte        16 dup (?)
zzzzzzseg           ends
                    end         Main
```

13.4 Routine d'E/S de fichiers de la bibliothèque standard UCR

Bien que les routines d'E/S de fichier de MS-DOS ne soient pas trop mauvaises, la Bibliothèque standard UCR fournit une panoplie d'E/S de fichier qui rend les E/S séquentielles bloquées aussi faciles que les E/S un caractère à la fois. En outre, avec très peu d'effort, vous pouvez employer toutes les routines de la StdLib comme **printf**, **print**, **puti**,

puth, **putc**, **getc**, **get**, etc., en exécutant des E/S de fichier. Ceci simplifie considérablement les opérations de fichier texte en assembleur.

Notez qu'il vaut mieux laisser au DOS pur les E/S orientées enregistrement, ou binaires, quand vous voulez faire des accès aléatoires dans un fichier. Les routines de la Bibliothèque standard soutiennent en fait seulement les E/S texte séquentielles. Néanmoins, c'est la forme la plus commune d'E/S de fichiers, aussi les routines de la Bibliothèque standard sont tout à fait utiles.

La Bibliothèque standard UCR fournit huit routines d'E/S de fichiers : **fopen**, **fcreate**, **fclose**, **fgetc**, **fread**, **fputc** et **fwrite**. **Fgetc** et **fputc** exécutent des E/S un caractère à la fois, **fread** et **fwrite** vous permettent de lire et d'écrire des blocs de données, les quatre autres fonctions effectuent les opérations correspondantes du DOS.

La Bibliothèque standard UCR emploie une *variable de fichier* spéciale pour garder trace des opérations sur les fichiers. Il y a un type d'enregistrement spécial, *FileVar*, déclaré dans stdlib.a[8]. En utilisant les routines d'E/S de fichier de StdLib, vous devez créer une variable de type **FileVar** pour chaque fichier que vous avez besoin d'ouvrir en même temps. C'est très facile, utilisez juste une définition de la forme :

```
MyFileVar  FileVar      {}
```

Notez svp qu'une variable de fichier de la Bibliothèque standard n'est pas la même chose qu'une handle de fichier du DOS. C'est une structure qui contient la handle de fichier du DOS, un tampon (pour les E/S bloquées) et diverses variables d'index et de statut. La structure interne de ce type est sans intérêt (rappelez-vous l'encapsulation des données !), sauf pour la programmeur des routines de fichier. Vous passerez l'adresse de cette variable de fichier aux diverses routines d'E/S de fichiers de la Bibliothèque standard.

13.4.1 Fopen

Paramètres d'entrée: **ax**- Mode d'ouverture de fichier
0- Fichier ouvert en lecture
1- Fichier ouvert en écriture
dx:si- Pointe sur une chaîne terminée par zéro contenant le nom du fichier
es:di- Pointe sur la variable de fichier StdLib
Paramètres de sortie: Si la retenue est à 1, **ax**. contient le code d'erreur du DOS (voir la fonction Ouvrir du DOS)

Fopen ouvre un fichier séquentiel texte pour la lecture *ou* l'écriture. À la différence du DOS, vous ne pouvez pas ouvrir un fichier pour la lecture et l'écriture. En outre, c'est un fichier séquentiel texte qui ne supporte pas l'accès sélectif (aléatoire). Notez que le fichier doit exister sinon **fopen** renverra une erreur. C'est vrai même si vous ouvrez le fichier en écriture.

Notez que si vous ouvrez un fichier en écriture et que ce fichier existe déjà, toutes les données écrites dans le fichier écraseront les données existantes. Quand vous fermez le fichier, toutes les données apparaissant dans le fichier après les données que vous avez écrites seront toujours là. Si vous voulez effacer le fichier existant avant d'y écrire des données, employez la fonction **fcreate**.

13.4.2 Fcreate

Paramètres d'entrée: **dx:si**- Pointe sur une chaîne terminée par zéro contenant le nom du fichier
es:di- Pointe sur une variable de fichier StdLib
Paramètres de sortie: Si la retenue est à 1, **ax**. contient le code d'erreur du DOS (voir la fonction Ouvrir du DOS)

Fcreate crée un nouveau fichier et l'ouvre en écriture. Si le fichier existe déjà, le **fcreate** supprime le fichier existant et crée un nouveau. Il initialise la variable de fichier pour les sorties mais, sinon, est identique à l'appel **fopen**.

13.4.3 Fclose

Paramètres d'entrée: **es:di**- Pointe sur une variable de fichier StdLib
Paramètres de sortie: Si la retenue est à 1, **ax**. contient le code d'erreur du DOS (voir la fonction Ouvrir du DOS)

[8] En fait, on le déclare dans file.a. Stdlib.a inclut file.a, ainsi cette définition apparaît aussi à l'intérieur de stdlib.a.

Fclose ferme un fichier et met à jour toutes les informations internes pour faire le ménage. Il est très important que vous fermiez tous les fichiers ouverts avec **fopen** ou **fcreate** en utilisant cet appel. Quand vous utilisez des fonctions de fichier du DOS, si vous oubliez de fermer un fichier, DOS le fera automatiquement pour vous quand votre programme se termine. Cependant, les routines de StdLib cachent des données dans des tampons internes. L'appel à **fclose** vide automatiquement ces tampons sur le disque. Si vous quuittez votre programme sans appeler **fclose**, vous pouvez perdre des données écrites dans le fichier mais pas encore transférées à partir depuis le tampon interne sur le disque.

Si vous êtes dans un environnement où il est possible que quelqu'un interrompe le programme sans vous laisser le temps de fermer le fichier, vous devriez appeler les routines **fflush** (voyez la prochaine section) de façon régulière pour éviter de perdre trop de données.

13.4.4 Fflush

Paramètres d'entrée: **es:di**- Pointe sur la variable de fichier StdLib
Paramètres de sortie: Si la retenue est à 1, **ax**. contient le code d'erreur du DOS (voir la fonction Ouvrir du DOS)

Cette routine écrit immédiatement toutes les données du tampon interne de fichier sur le disque. Notez que vous devriez seulement employer cette routine à l'intention des fichiers ouverts en écriture (ou ouvert avec **fcreate**). Si vous écrivez des données dans un fichier et si vous avez besoin laisser le fichier ouvert par la suite, tout en restant inactif pendant un certain temps, vous devriez effectuer une opération de vidage au cas où le programme se terminerait anormalement.

13.4.5 Fgetc

Paramètres d'entrée: **es:di**- Pointe sur la variable de fichier StdLib
Paramètres de sortie: Si la retenue est à 0, **al**. contient le caractère lu dans le fichier
Si la retenue est à 1, **ax**. contient le code d'erreur du DOS (voir la fonction Ouvrir du DOS)
ax. contiendra 0 si vous essayez de lire au delà de la fin du fichier.

Fgetc lit un caractère unique à partir du fichier et renvoie ce caractère dans le registre **al**. A la différence des appels au DOS, les E/S un caractère à la fois de **fgetc** sont relativement rapides puisque les routines de StdLib utilisent les E/S bloquées. Naturellement, les appels multiples à **fgetc** ne seront jamais plus rapides qu'un appel à **fread** (voyez la prochaine section), mais les performances ne sont pas trop mauvaises.

Fgetc est très flexible. Comme vous le verrez bientôt, vous pouvez réorienter les routines d'entrée de StdLib pour lire leurs données à partir d'un fichier en utilisant **fgetc**. Ceci vous permet d'employer les routines de niveau plus élevé comme **gets** et **getsm** pour lire des données à partir d'un fichier.

13.4.1 Fread

Paramètres d'entrée: **es:di**- Pointe sur la variable de fichier StdLib
dx:si- Pointe sur un tampon d'importation de données
cx- Contient le compte des octets
Paramètres de sortie: Si la retenue est à 0, **ax**. contient le nombre effectif d'octets lus dans le fichier.
Si la retenue est à 1, **ax**. contient le code d'erreur du DOS (voir la fonction Ouvrir du DOS)

Fread est très semblable à la commande Lire du DOS. Il vous permet de lire un bloc de bytes, au lieu d'un seul byte, à partir d'un fichier. Notez que vous ne faites que lire un bloc de bytes à partir d'un fichier, l'appel au DOS est légèrement plus efficace que **fread**. Cependant, si vous avez un mélange de lectures de byte unique et de multi-bytes, la combinaison de **fread** et de **fgetc** marche très bien.

Comme avec l'opération Lire du DOS, si le nombre d'octets retourné dans **ax** ne correspond pas à la valeur passée dans le registre **cx**, alors vous avez lu les bytes qui restaients dans le fichier. Quand ceci se produit, l'appel suivant à **fread** ou à **fgetc** renverra une erreur d'EOF (la retenue sera à un et **ax** contiendra zéro). Notez que **fread** ne renvoie pas EOF sans qu'il n'y ait eu zéro bytes lus à partir du fichier.

13.4.7 Fputc

Paramètres d'entrée: **es:di**- Pointe sur la variable de fichier StdLib
al- Contient le caractère à écrire dans le fichier
Paramètres de sortie: Si la retenue est à 1, **ax**. contient le code d'erreur du DOS (voir la fonction Ouvrir du DOS)

Fputc écrit un caractère unique (dans **al**) dans le fichier indiqué par la variable de fichier dont l'adresse est dans **es:di**. Cet appel ajoute uniquement le caractère dans **al** à un tampon interne (qui fait partie de la variable de fichier) jusqu'à ce que le tampon soit plein. Lorsque le tampon est rempli ou que vous appelez **fflush** (ou que vous fermez le fichier avec **fclose**), les routines d'E/S de fichier écrivent les données sur le disque.

13.4.8 Fwrite

Paramètres d'entrée: **es:di**- Pointe sur la variable de fichier StdLib
dx:si- Pointe sur un tampon d'exportation de données
cx- Contient le compte des octets
Paramètres de sortie: Si la retenue est à 0, **ax**. contient le nombre effectif d'octets écrits dans le fichier.
Si la retenue est à 1, **ax**. contient le code d'erreur du DOS (voir la fonction Ouvrir du DOS)

Comme **fread**, **fwrite** travaille sur des blocs de bytes. Il vous permet d'écrire un bloc de bytes dans un fichier ouvert en écriture avec **fopen** ou **fcreate**.

13.4.9 Redirection des E/S avec les routines d'E/S de fichier de stdlib

La Bibliothèque standard fournit très peu de routines d'E/S de fichier. **Fputc** et **fwrite** sont les seules deux routines de sortie, par exemple. La bibliothèque du langage C par contre (sur laquelle la Bibliothèque standard UCR est basée), fournit beaucoup plus de routines comme *fprintf*, *fputs*, *fscanf*, etc... Aucune de celles-ci n'est nécessaire dans notre version, car cette dernière fournit un mécanisme de redirection d'E/S qui vous permet de réutiliser toutes les routines d'E/S existantes pour exécuter des E/S.

La routine **putc** de la Bibliothèque standard UCR se compose d'une instruction **jmp** unique. Cette instruction transfère le contrôle à une routine de sortie effective via une adresse indirecte interne au code de **putc**. Normalement, cette variable pointeur pointe sur un morceau de code qui écrit le caractère dans le registre **al** dans le périphérique de sortie standard du DOS. Cependant, la Bibliothèque standard fournit également quatre routines qui vous permettent de manipuler ce pointeur indirect. En changeant ce dernier, vous pouvez réorienter la sortie depuis sa routine courante vers une routine de votre choix. *Toutes* les routines de sortie de la Bibliothèque standard (par exemple, **printf**, **puti**, **puth**, **puts**) appellent **putc** pour sortir différents caractères. Par conséquent, la réorientation de la routine **putc** affecte toutes les routines de sortie.

De même, la routine **getc** n'est rien d'autre qu'un **jmp** indirect dont la variable pointeur pointe normalement sur un morceau de code qui lit des données sur l'entrée standard du DOS. Puisque toutes les routines d'entrée de la Bibliothèque standard appellent la fonction **getc** pour lire chaque caractère vous pouvez réorienter les entrées de fichier de manière identique aux sorties de fichier.

Les routines ***GetOutAdrs***, ***SetOutAdrs***, ***PushOutAdrs*** et ***PopOutAdrs*** de la Bibliothèque Standard sont les quatre routines principales qui manipulent le pointeur de redirection de sortie. **GetOutAdrs** renvoie l'adresse de la routine de sortie courante dans les registres **es:di**. Réciproquement, **SetOutAdrs** attend que vous passiez l'adresse d'une nouvelle routine de sortie dans les registres **es:di** et stocke cette adresse dans le pointeur de sortie. **PushOutAdrs** et **PopOutAdrs** poussent et chargent le pointeur sur une pile interne. Celles-ci n'emploient pas la pile hardware du 80x86. Vous êtes limité à un nombre restreint de push et pop. Généralement, vous ne devriez pas envisager de pouvoir pousser plus de quatre de ces adresses sur la pile interne sans débordement.

GetInAdrs, ***SetInAdrs***, ***PushInAdrs***, et ***PopInAdrs*** sont les routines complémentaires pour le vecteur d'entrée. elles vous permettent de manipuler le pointeur de routine d'entrée. Notez que la pile pour **PushInAdrs**/**PopInAdrs** n'est pas la même que la pile pour **PushOutAdrs**/**PopOutAdrs**. Les push et les pop de ces deux piles sont indépendant les uns des autres.

Normalement, le pointeur de sortie (nous nous y référerons dorénavant comme *hook* (crochet) *de sortie*) pointe sur la routine PutcStdOut de la Bibliothèque Standard[9]. Par conséquent, vous pouvez renvoyer le hook de sortie à son état normal d'initialisation à tout moment en exécutant les instructions[10] :

```
            mov    di, seg SL_PutcStdOut
            mov    es, di
            mov    di, offset SL_PutcStdOut
            SetOutAdrs
```

La routine de PutcStdOut écrit le caractère dans le registre **al** sur la sortie standard du DOS, qui elle-même pourrait être réorientée sur un fichier ou périphérique quelconque (en employant l'opérateur DOS ">" de redirection). Si vous voulez vous assurer que votre sortie va à l'affichage vidéo, vous pouvez toujours appeler la routine **PutcBIOS** qui appelle le BIOS directement pour afficher un caractère[1]. Vous pouvez forcer toutes les sorties de la Bibliothèque Standard sur *périphérique d'erreur standard* en utilisant une séquence de code comme:

```
            mov    di, seg SL_PutcBIOS
            mov    es, di
            mov    di, offset SL_PutcBIOS
            SetOutAdrs
```

Généralement, vous n'écraserez pas purement et simplement le hook de sortie en stockant un pointeur sur votre routine par dessus le pointeur qui se trouvait là et en reconstituant le hook à partir de **PutcStdOut** après avoir terminé. Qui sait si le hook pointait sur **PutcStdOut** en premier lieu ? La meilleure solution est d'utiliser les routines de la Bibliothèque Standard **PushOutAdrs** et **PopOutAdrs** pour préserver et reconstituer le hook précédent. Le code suivant démontre une manière plus *polie* de modifier le hook de sortie:

```
            PushOutAdrs         ;Sauve routine de sortie courante.
            mov    di, seg Output_Routine
            mov    es, di
            mov    di, offset Output_Routine
            SetOutAdrs
<Fait toutes les sorties sur Output_Routine ici>
            PopOutAdrs          ;Restaure routine sortie précédente.
```

Manipulez les entrées d'une façon identique en utilisant les routines correspondantes d'accès au hook d'entrée et les routines **SL_GetcStdOut** et **SL_GetcBIOS**. Gardez toujours à l'esprit qu'il y a un nombre limité d'entrées sur les piles des hooks d'entrée et de sortie, aussi faites attention à combien d'éléments vous poussez sur ces piles sans les récupérer.

Pour réorienter les sorties sur un fichier (ou réorienter les entrées à partir d'un fichier), vous devez d'abord écrire une courte routine qui écrit (lit) un caractère unique sur (à partir d')un fichier. C'est très facile. Le code pour qu'une routine exporte des données dans un fichier décrit par la variable fichier ***OutputFile*** est

```
ToOutput            proc   far
                    push   es
                    push   di

; Charge ES:DI avec l'adresse de la variable OutputFile. Ce code
; suppose qu'OutputFile est de type FileVar, et non un pointeur sur
; une variable de type FileVar.

                    mov    di, seg OutputFile
                    mov    es, di
                    mov    di, offset OutputFile

; Sort le caractère dans AL sur le fichier décrit par "OutputFile"

                    fputc

                    pop    di
                    pop    es
                    ret
ToOutput            endp
```

---

[9] . En fait, la routine est *SL_PutcStdOut*. La macro de la Bbibliothèque standard par laquelle vous appellez normalement cette routine est PutcStdOut.

[10] Si vous n'avez aucun appel à PutcStdOut dans votre programme, vous devrez également ajouter l'instruction "externdef SL_PutcStdOut:far" à votre programme.

[11] Il est possible de réorienter la sortie du BIOS elle-même, mais ceci est rarement fait et pas facile à faire depuis le DOS.

Maintenant avec un seul morceau supplémentaire de code, vous pouvez commencer à écrire des données sur un fichier de sortie en utilisant toutes les routines de sortie de la Bibliothèque standard. C'est un court morceau de code qui réoriente le hook de sortie sur la routine "ToOutput" ci-dessus:

```
SetOutFile   proc
                    push   es
                    push   di

                    PushOutAdrs         ;Sauve hook de sortie courant.
                    mov    di, seg ToOutput
                    mov    es, di
                    mov    di, offset ToOutput
                    SetOutAdrs

                    pop    di
                    pop    es
                    ret
SetOutFile   endp
```

Il n'y a besoin d'aucune routine spécifique pour reconstituer le hook de sortie à sa valeur précédente; PopOutAdrs se chargera de cette tâche tout seul.

13.4.10 Un exemple d'E/S de fichier

Le morceau de code suivant assemble tous les éléments des dernières sections. C'est un court programme qui ajoute des numéros de ligne à un fichier texte. Ce programme attend deux paramètres sur la ligne de commande : un fichier d'entrée et un fichier de sortie. Il copie le fichier d'entrée dans le fichier de sortie tout en ajoutant des numéros de ligne au commencement de chaque ligne dans le fichier de sortie. Ce code montre l'utilisation des routines d'E/S de fichier de la Bibliothèque Standard **argc**, **argv** et de la redirection d'E/S.

```
;Ce programme copie le fichier d'entrée dans fichier de sortie et

; ajoute des numéros de ligne pendant qu'il copie le fichier

                    include      stdlib.a

                    includelib stdlib.lib

dseg                segment      para public 'data'

ArgCnt              word         0
LineNumber   word         0
DOSErrorCode word         0
InFile              dword  ?            ;Ptr sur nom fichier Entrée.
OutFile             dword  ?            ;Ptr sur nom fichier Sortie
InputLine           byte         1024 dup (0) ;Tampon données E/S.
OutputFile   FileVar      {}
InputFile           FileVar      {}

dseg                ends

cseg                segment      para public 'code'
                    assume       cs:cseg, ds:dseg

; ReadLn- Lit une ligne de texte du fichier d'entrée et stocke les
; données dans le tampon InputLine:

ReadLn              proc
                    push   ds
                    push   es
                    push   di
                    push   si
                    push   ax

                    mov    si, dseg
                    mov    ds, si
                    mov    si, offset InputLine
                    lesi   InputFile
```

```
GetLnLp:
                    fgetc
                    jc      RdLnDone            ;En cas d' erreur bizarre.
                    cmp     ah, 0         ;Vérifie EOF.
                    je      RdLnDone            ;Note:retenue est à 1.
                    mov     ds:[si], al
                    inc     si
                    cmp     al, lf              ;à EOLN?
                    jne     GetLnLp
                    dec     si                  ;Revient avant LF.
                    cmp     byte ptr ds:[si-1], cr    ;CR avant LF?
                    jne     RdLnDone
                    dec     si                          ;Si oui, saute le aussi.

RdLnDone:           mov byte ptr ds:[si], 0   ;Termine par zéro.
                    pop     ax
                    pop     si
                    pop     di
                    pop     es
                    pop     ds
                    ret
ReadLn              endp

; MyOutput- Écrit le caractère dans AL dans le fichier de sortie.

MyOutput            proc    far
                    push    es
                    push    di
                    lesi    OutputFile
                    fputc
                    pop     di
                    pop     es
                    ret
MyOutput            endp

; Le programme principal qui fait tout le travail:

Main                proc
                    mov     ax, dseg
                    mov     ds, ax
                    mov     es, ax

; Il faut appeler la routine d'initialization du gestionnaire de
; mémoire si vous utilisez des routines qui appellent malloc !
; ARGV est un bon exemple d'une routine qui appelle malloc.

                    meminit

; Ce programme doit être appelé comme suit:
; fileio file1, file2
; autrment il y a une erreur.

                    argc
                    cmp     cx, 2         ;Doit avoir deux paramètres.
                    je      Got2Parms
BadParms:           print
                    byte    "Usage: FILEIO infile, outfile",cr,lf,0
                    jmp     Quit

; OK, nous avons deux paramètres, sans doute des noms valides.
; Fait des copies des noms de fichier et stocke leurs pointeurs.

Got2Parms:   mov    ax, 1         ;Obtient nom fichier entrée
                    argv
                    mov     word ptr InFile, di
                    mov     word ptr InFile+2, es
                    mov     ax, 2         ;Obtient nom fichier sortie
                    argv
                    mov     word ptr OutFile, di
                    mov     word ptr OutFile+2, es

; Envoie les noms de fichier au périphérique de sortie standard

                    printf
```

```
                    byte   "Input fichier: %^s\n"
                    byte   "Output fichier: %^s\n",0
                    dword InFile, OutFile

; Ouvre le fichier d'entrée:

                    lesi   InputFile
                    mov    dx, word ptr InFile+2
                    mov    si, word ptr InFile
                    mov    ax, 0
                    fopen
                    jnc    GoodOpen
                    mov    DOSErrorCode, ax
                    printf
                    byte   "Could not open input fichier, DOS: %d\n",0
                    dword DOSErrorCode
                    jmp    Quit

; Crée un nouveau fichier pour la sortie:

GoodOpen:           lesi   OutputFile
                    mov    dx, word ptr OutFile+2
                    mov    si, word ptr OutFile
                    fcreate
                    jnc    GoodCreate
                    mov    DOSErrorCode, AX
                    printf
                    byte   "Could not open output fichier, DOS: %d\n",0
                    dword DOSErrorCode
                    jmp    Quit

; OK, sauve le hook de sortie et redirige les sorties.

GoodCreate:  PushOutAdrs
                    lesi   MyOutput
                    SetOutAdrs

WhlNotEOF:   inc    LineNumber

; OK, lit la ligne en entrée fournie par l'utilisateur

                    call   ReadLn
                    jc     BadInput

; OK, redirige les sorties sur notre fichier de sortie et écrit la
; dernière ligne lue préfixée par un numéro de ligne:

                    printf
                    byte   "%4d: %s\n",0
                    dword LineNumber, InputLine
                    jmp    WhlNotEOF

BadInput:           push   ax                  ;Sauve code d'erreur.
                    PopOutAdrs          ;Restaure hook de sortie.
                    pop    ax                  ;Récupère code erreur.
                    test   ax, ax              ;Erreur d'EOF? (AX = 0)
                    jz     CloseFiles
                    mov    DOSErrorCode, ax
                    printf
                    byte   "Input error, DOS: %d\n",0
                    dword LineNumber

; OK, ferme les fichiers et quitte:

CloseFiles:  lesi   OutputFile
                    fclose
                    lesi   InputFile
                    fclose

Quit:        ExitPgm                   ;Macro DOS pour quitter prog.
Main                endp
cseg                ends

sseg                segment      para stack 'stack'
stk                 byte         1024 dup ("stack ")
```

```
sseg                ends

zzzzzzseg           segment     para public 'zzzzzz'
LastBytes           byte        16 dup (?)
zzzzzzseg           ends
                    end         Main
```

13.5 Exemple de programme

Si vous voulez utiliser les routines de sortie de la Bibliothèque standard (**putc**, **print**, **printf**, etc...) pour exporter des données dans un fichier, vous pouvez le faire en réorientant manuellement les sorties avant et après chaque appel à ces routines. Malheureusement, celà peut demander beaucoup de travail si vous mélangez les E/S interactives et les E/S de fichier. Le programme suivant présente plusieurs macros qui vous simplifient la tâche.

```
; FileMacs.asm
;
; Ce programme présente un éventail de macros qui rendent les E/S de
; fichiers de la Standard Library encore plus faciles.
;
; Le programme principal écrit une table de multiplication dans le
; fichier "MyFile.txt".
                    .xlist
                    include     stdlib.a
                    includelib stdlib.lib
                    .list


dseg                segment     para public 'data'

CurOutput           dword  ?

Filename            byte        "MyFile.txt",0

i                   word        ?
j                   word

TheFile             filevar     {}

dseg                ends

cseg                segment     para public 'code'
                    assume      cs:cseg, ds:dseg

; Macros For-Next du Chapitre Huit.
; Voir Chapitre Huit pour les détails.

ForLp       macro   LCV, Start, Stop
                    local  ForLoop

                    ifndef      $$For&LCV&
                          $$For&LCV&=  0
                    else
                          $$For&LCV&=  $$For&LCV& + 1
                    endif

                    mov    ax, Start
                    mov    LCV, ax

ForLoop             textequ     @catstr($$For&LCV&, %$$For&LCV&)
&ForLoop&:
                    mov    ax, LCV
                    cmp    ax, Stop
                    jg     @catstr($$Next&LCV&, %$$For&LCV&)
                    endm

Next                macro  LCV
                    local  NextLbl
                    inc         LCV
                    jmp         @catstr($$For&LCV&, %$$For&LCV&)
NextLbl             textequ     @catstr($$Next&LCV&, %$$For&LCV&)
```

```
&NextLbl&:
                    endm

; Macros d'E/S de fichier:
;
;
; SetPtr définit la variable pointeur CurOutput. Cette macro est
; appelée par d'autres macros, ce n'est pas quelque chose que vous
; appelleriez  normalement. Sa seule fonction est de raccourcir les
; autres macros et épargner quelques frappes.

SetPtr              macro  fvar
                    push         es
                    push         di

                    mov          di, offset fvar
                    mov          word ptr CurOutput, di
                    mov          di, seg fvar
                    mov          word ptr CurOutput+2, di

                    PushOutAdrs
                    lesi         FileOutput
                    SetOutAdrs
                    pop          di
                    pop          es
                    endm
;
;
;
; fprint- Imprime une chaîne à l'écran
;
; Usage:
; fprint filevar,"String or bytes to print"
;
; Note: vous pouvez fournir des données byte ou chaîne optionnelles
; après la chaîne ci-dessus en mettant les données entre crochets,
;
; p. ex.             fprint filevar,<"string to print",cr,lf>
;
; Ne mettez *PAS* de byte zéro de terminaison à la fin de la chaîne,
; la macro fprint le fera automatiquement.

fprint              macro  fvar:req, string:req
                    SetPtr       fvar

                    print
                    byte         string
                    byte         0
                    PopOutAdrs
                    endm

; fprintf- Imprime une chaîne formatée à l'écran.
; fprintff- Comme fprintf, mais traite les nombres en virgule
; flottante en plus des autres items.
;
; Usage:
; fprintf filevar,"format string", optional data values
; fprintff filevar,"format string", optional data values
; Exemples:
;
;      fprintf      FileVariable,"i=%d, j=%d\n", i, j
;      fprintff     FileVariable,"f=%8.2f, i=%d\n", f, i
;
; Note: si vous voulez specifier une liste de chaînes et de bytes
; pour la chaîne format, mettez les items entre crochets, p.ex.,
;
;      fprintf      FileVariable, <"i=%d, j=%d",cr,lf>, i, j
;
;

fprintf             macro  fvar:req, FmtStr:req, Operands:vararg
```

```
                    setptr      fvar

                    printf
                    byte        FmtStr
                    byte        0

                    for         ThisVal, <Operands>
                    dword ThisVal
                    endm

                    PopOutAdrs
                    endm

fprintff            macro fvar:req, FmtStr:req, Operands:vararg
                    setptr      fvar

                    printff
                    byte        FmtStr
                    byte        0

                    for         ThisVal, <Operands>
                    dword ThisVal
                    endm

                    PopOutAdrs
                    endm

; F- Il s'agit d'une macro generique qui convertit des fonctions
; isolées (pas de parms dans le flux de code) de la stdlib en
; routines de sortie de fichier. À utiliser avec putc, puts,
; puti, putu, putl, putisize, putusize, putlsize, putcr, etc.
;
; Usage:
;
;     F     StdLibFunction, FileVariable
;
; Examples:
;
;     mov   al, 'A'
;     F     putc, TheFile
;     mov   ax, I
;     mov   cx, 4
;     F     putisize, TheFile

F                   macro func:req, fvar:req
                    setptr      fvar
                    func
                    PopOutAdrs
                    endm

; WriteLn- Courte macro pour gérer l'opération putcr (car ce code
; appelle très souvent putcr).

WriteLn             macro fvar:req
                    F           putcr, fvar
                    endm

; FileOutput- Écrit le caractère dans AL dans un fichier en sortie.
; Les macros ci-dessus redirigent la sortie standard sur cette
; routine pour imprimer des données dans un fichier.

FileOutput  proc    far
                    push  es
                    push  di
                    push  ds
                    mov   di, dseg
                    mov   ds, di
```

```
                    les    di, CurOutput
                    fputc

                    pop    ds
                    pop    di
                    pop    es
                    ret
FileOutput   endp

; Un programme principal tout simple qui teste le code ci-dessus.
; Ce programme écrit une table de multiplication dans le fichier
; "MyFile.txt"

Main                proc
                    mov    ax, dseg
                    mov    ds, ax
                    mov    es, ax
                    meminit

; Rewrite(TheFile, FileName);

                    ldxi   FileName
                    lesi   TheFile
                    fcreate

; writeln(TheFile);
; writeln(TheFile,'    ');
; for i := 0 to 5 do write(TheFile,'|',i:4,' ');
; writeln(TheFile);.Chapter 13

                    WriteLn TheFile
                    fprint TheFile,"    "

                    forlp i,0,5
                    fprintf TheFile, "|%4d ", i
                    next   i
                    WriteLn TheFile

; for j := -5 to 5 do begin
;
;      write(TheFile,'----');
;      for i := 0 to 5 do write(TheFile, '+-----');
;      writeln(TheFile);
;
;      write(j:3, ' |');
;      for i := 0 to 5 do write(i*j:4, ' |);
;      writeln(TheFile);
;
; end;

                    forlp j,-5,5

                    fprint TheFile,"----"
                    forlp i,0,5
                    fprintf TheFile,"+-----"
                    next i
                    fprint TheFile,<"+",cr,lf>

                    fprintf TheFile, "%3d |", j

                    forlp i,0,5

                    mov    ax, i
                    imul   j
                    mov    cx, 4
                    F      putisize, TheFile
                    fprint TheFile, " |"
```

```
                    next i
                    Writeln TheFile

                    next j
                    WriteLn TheFile

; Close(TheFile);

                    lesi TheFile
                    fclose

Quit:       ExitPgm                     ;Macro DOS pour quitter prog.
Main                endp

cseg                ends

sseg                segment     para stack 'stack'
stk                 db          1024 dup ("stack ")
sseg                ends

zzzzzzseg           segment     para public 'zzzzzz'
LastBytes           db          16 dup (?)
zzzzzzseg           ends
                    end         Main
```

13.6 Exercices de laboratoire

Les trois programmes suivants font tous la même chose: ils copient le fichier "ex13_1.in" dans le fichier "ex13_1.out". La différence est la manière dont ils copient les fichiers. Le premier programme, ex13_1a, copie les données à partir du fichier d'entrée dans le fichier de sortie en utilisant les E/S un caractère à la fois sous DOS. Le deuxième programme, ex13_1b, emploie les E/S bloquées sous DOS. Le troisième programme, ex13_1c, emploie les routines d'E/S de fichier de la Bibliothèque Standard pour copier les données.

Exécutez ces trois programmes et mesurez la quantité de temps qu'ils prennent pour s'exécuter[1]. **Pour votre rapport de laboratoire:** relevez les temps de fonctionnement et commentez les performances relatives de ces méthodes de transfert de données. La perte de performance des routines de la Bibliothèque standard (comparées aux E/S bloquées) est-elle justifiée en termes de facilité d'utilisation de ces routines? Expliquez.

```
; EX13_1a.asm
;
; Ce programme copie un fichier dans un autre en utilisant les E/S
; un caractère à la fois. Il est facile à écrire, à lire et à
; comprendre, mais les E/S un caractère à la fois sont très lentes.
; Lancez ce programme et chronométrez son exécution. Ensuite
; lancez l'exercise correspondant sur les E/S bloquées et comparez
; les temps d'exécution des deux programmes.

            include     stdlib.a
            includelib stdlib.lib

dseg        segment     para public 'data'

FHndl  word         ?
FHndl2      word        ?
Buffer      byte        ?

FName  equ          this word
FNamePtr    dword  FileName

Filename    byte        "Ex13_1.in",0
Filename2   byte        "Ex13_1.out",0
```

[12] Si vous avez une machine vraiment rapide, vous pouvez aggrandir le fichier ex13_1.in (en copiant et en collant des données dans le fichier).

```
dseg        ends


cseg        segment     para public 'code'
            assume      cs:cseg, ds:dseg

Main        proc
            mov   ax, dseg
            mov   ds, ax
            mov   es, ax
            meminit

            mov   ah, 3dh            ;Ouvre le fichier d'entrée
            mov   al, 0        ; en lecture
            lea   dx, Filename ;DS pointe sur le segment
            int   21h                ; du nom de fichier
            jc    BadOpen
            mov   FHndl, ax          ;Sauve handle fichier

            mov   FName, offset Filename2   ;Met en place en cas
            mov   FName+2, seg FileName2    ; d'erreur d'ouverture.

            mov   ah, 3ch            ;Ouvre fichier sortie en écriture
            mov   cx, 0        ; avec attributs fichier normaux
            lea   dx, Filename2      ;Suppose DS pointe sur segment
            int   21h                ; du nom du fichier
            jc    BadOpen
            mov   FHndl2, ax   ;Sauve handle du fichier

LP:         mov   ah,3fh             ;Lit données dans fichier
            lea   dx, Buffer   ;Adresse tampon données
            mov   cx, 1        ;Lit un byte
            mov   bx, FHndl          ;Obtient valeur handle du fichier
            int 21h
            jc    ReadError
            cmp   ax, cx             ;EOF atteinte?
            jne   EOF

            mov   ah,40h             ;Ecrit données dans fichier
            lea   dx, Buffer   ;Adresse tampon données
            mov   cx, 1        ;Écrit un byte
            mov   bx, FHndl2   ;Obtient valeur handle du fichier int  21h
            jc    WriteError
            jmp   LP                 ;Lit byte suivant

EOF:        mov   bx, FHndl
            mov   ah, 3eh            ;Ferme fichier
            int   21h
            jmp   Quit

ReadError: printf
            byte  "Error while reading data from file '%s'.",cr,lf,0
            dword FileName
            jmp   Quit

WriteError: printf
            byte  "Error while writing data to file '%s'.",cr,lf,0
            dword FileName2
            jmp   Quit

BadOpen:    printf
            byte  "Could not open '%^s'. Make sure this file is "
            byte  "in the ",cr,lf
            byte  "current directory before attempting to run "
            byte  "this program again.", cr,lf,0
            dword FName
```

```
Quit:  ExitPgm                  ;Macro DOS pour quitter prog.
Main         endp

cseg         ends

sseg         segment      para stack 'stack'
stk          db           1024 dup ("stack ")
sseg         ends

zzzzzzseg    segment      para public 'zzzzzz'
LastBytes    db           16 dup (?)
zzzzzzseg    ends
             end          Main
```

```
; EX13_1b.asm
;
; Ce programme copie un fichier dans un autre en utilisant les E/S
; bloquées. Lancez ce programme et chronométrez son exécution.
; Comparez le temps d'exécution de ce programme avec celui de
; l'exemple E/S un caractère à la fois et de l'exemple
; E/S de fichiers de la Bibliothèque standard (ex13_1a and ex13_1c).

             include      stdlib.a
             includelib stdlib.lib

dseg         segment      para public 'data'

; Handles de fichier pour les fichiers que nous ouvrirons.

FHndl  word         ?                  ;Handle fichier Entrée
FHndl2       word         ?                  ;Handle fichier Sortie

Buffer       byte         256 dup (?)  ;Zone tampon fichier

FName  equ          this word          ;Ptr sur nom fichier courant
FNamePtr     dword  FileName

Filename     byte         "Ex13_1.in",0      ;Nom fichier Entrée
Filename2    byte         "Ex13_1.out",0     ;Nom fichier Sortie

dseg         ends

cseg         segment      para public 'code'
             assume       cs:cseg, ds:dseg

Main         proc
             mov    ax, dseg
             mov    ds, ax
             mov    es, ax
             meminit

             mov    ah, 3dh             ;Ouvre fichier Entrée
             mov    al, 0         ; en lecture
             lea    dx, Filename ;Suppose DS pointe sur
             int    21h                 ; segment nom fichier
             jc     BadOpen
             mov    FHndl, ax           ;Sauve handle fichier

             mov    FName, offset Filename2   ;Met en place en cas
             mov    FName+2, seg FileName2    ; d'erreur d'ouverture.

             mov    ah, 3ch             ;Ouvre fichier Sortie en écriture
             mov    cx, 0         ; avec attributs fichier normaux
             lea    dx, Filename2       ;Suppose DS pointe sur segment
             int    21h                 ; du nom de fichier
             jc     BadOpen
             mov    FHndl2, ax   ;Sauve handle fichier
```

```
; La boucle suivante lit 256 bytes à la fois dans le fichier et
; ensuite écrit ces 256 bytes dans le fichier de sortie.

LP:          mov    ah,3fh              ;Lit données du fichier
             lea    dx, Buffer    ;Adresse tampon données
             mov    cx, 256             ;Lit 256 bytes
             mov    bx, FHndl           ;Obtient valeur handle fichier
             int    21h
             jc     ReadError
             cmp    ax, cx              ;EOF atteinte?
             jne    EOF

             mov    ah, 40h             ;Écrit données dans fichier
             lea    dx, Buffer    ;Adresse du tampon sortie
             mov    cx, 256             ;Ecrit 256 bytes
             mov    bx, FHndl2    ;Handle sortie
             int    21h
             jc     WriteError
             jmp    LP                  ;LIt bloc suivant

; Notez, ce n'est pas parce que le nombre de bytes lus n'égale pas
; 256 que nous avons fini, il pourrait y avoir jusqu'à255 bytes
; dans le tampon en attente de traitement.

EOF:         mov    cx, ax              ;Met # de bytes à écrire dans CX.
             jcxz   EOF2                ;Si CX = zero, c'est vraiment fini.
             mov    ah, 40h             ;Écrit données dans fichier
             lea    dx, Buffer    ;Adresse du tampon de sortie
             mov    bx, FHndl2    ;Handle de sortie
             int    21h
             jc     WriteError

EOF2:  mov   bx, FHndl
             mov    ah, 3eh             ;Ferme fichier
             int    21h
             jmp    Quit

ReadError: printf
             byte   "Error while reading data from file '%s'.",cr,lf,0
             dword FileName
             jmp    Quit

WriteError: printf
             byte   "Error while writing data to file '%s'.",cr,lf,0
             dword FileName2
             jmp    Quit

BadOpen:     printf
             byte   "Could not open '%^s'. Make sure this file is in "
             byte   "the ",cr,lf
             byte   "current directory before attempting to run "
             byte   "this program again.", cr,lf,0
             dword FName

Quit:  ExitPgm                   ;Macro DOS pour quitter prog.
Main         endp

cseg         ends

sseg         segment      para stack 'stack'
stk          db           1024 dup ("stack ")
sseg         ends

zzzzzzseg    segment      para public 'zzzzzz'
LastBytes    db           16 dup (?)
zzzzzzseg    ends
             end          Main
```

```
------------------------------------------------
; EX13_1c.asm
;
; Ce programme copie un fichier dans un autre en utilisant les
; routines d'E/S de fichier de la Bibliothèque standard. Ces
; routines vous permettent des E/S un caractère à la fois,
; mais elles massifient les données à transférer pour améliorer
; les performances du système. Vous devriez trouver que le temps
; d'exécution de ce code est quelque part entre les E/S bloquées
; (ex13_1b) et les E/S un caractère à la fois (EX13_1a); il sera,
; quoi qu'il en soit, beaucoup plus proche du temps des  E/S bloquées
; (probablement environ deux fois plus long que le temps de ces dernières).

            include     stdlib.a
            includelib stdlib.lib

dseg        segment     para public 'data'

InFile      filevar     {}
OutFile     filevar     {}

Filename    byte        "Ex13_1.in",0;Input file name.
Filename2 byte          "Ex13_1.out",0;Output file name

dseg        ends

cseg        segment     para public 'code'
            assume      cs:cseg, ds:dseg

Main        proc
            mov   ax, dseg
            mov   ds, ax
            mov   es, ax
            meminit

; Ouvre le fichier d'entrée:
            mov   ax, 0       ;Ouvre en lecture
            ldxi  Filename
            lesi  InFile
            fopen
            jc    BadOpen

; Ouvre le fichier de sortie:

            mov   ax, 1       ;Ouvre en sortie
            ldxi  Filename2
            lesi  OutFile
            fcreate
            jc    BadCreate

; Copie le fichier entrée dans le fichier sortie:

CopyLp:     lesi  InFile
            fgetc
            jc    GetDone

            lesi  OutFile
            fputc
            jmp   CopyLp

BadOpen:    printf
            byte  "Error opening '%s'",cr,lf,0
            dword Filename
            jmp   Quit

BadCreate: printf
            byte  "Error creating '%s'",cr,lf,0
            dword Filename2
```

```
            jmp    CloseIn

GetDone:    cmp    ax, 0        ;Vérifie EOF
            je     AtEOF

            print
            byte   "Error copying files (read error)",cr,lf,0

AtEOF:      lesi   OutFile
            fclose
CloseIn:    lesi   InFile
            fclose

Quit:  ExitPgm                   ;Macro DOS pour quitter prog.
Main        endp

cseg        ends

sseg        segment      para stack 'stack'
stk         db           1024 dup ("stack ")
sseg        ends

zzzzzzseg   segment      para public 'zzzzzz'
LastBytes   db           16 dup (?)
zzzzzzseg   ends
            end          Main
```

13.7 Projets de programmation

1) Le programme exemple dans la section 13.5 réachemine les sorties standards via les routines d'E/S de fichier de la Bibliothèque standard vous permettant d'utiliser n'importe laquelle de ces routines de sortie pour écrire des données dans un fichier. Écrivez un ensemble semblable de routines et de macros qui vous permettent de lire des données à partir d'un fichier en utilisant les routines d'entrée de la Bibliothèque Standard (**getc**, **gets**, **getm, scanf**, etc.). Redirigez les entrées par les fonctions d'entrée de fichier de la Bibliothèque standard.

2) Le dernier programme exemple de la section 13.3.12 (copyuc.asm sur le CD-ROM d'accompagnement) copie un fichier dans un autre, convertissant au besoin les minuscules en majuscules. Ce programme analyse normalement la ligne de commande directement et emploie les E/S bloquées pour copier les données dans le fichier. Récrivez ce programme en utilisant argv/argc pour traiter la ligne de commande paramètres et utilisez les routines d'E/S de fichier de la Bibliothèque Standard pour traiter chaque caractère dans le fichier.

3) Écrivez un programme "compteur de mots" qui compte le nombre de caractères, de mots et de lignes dans un fichier. Supposez qu'un mot est n'importe quel suite de caractères entre des espaces, des tabulations, des retours de chariot, des retours à la ligne, le commencement d'un fichier et la fin d'un fichier (si vous voulez épargner vos efforts, vous pouvez présumer qu'un symbole "espace" est n'importe quel code ASCII inférieur ou égal à une espace).

4) Écrivez un programme qui imprime un fichier texte ASCII sur l'imprimante. Employez les services int 17h du BIOS pour imprimer les caractères dans le fichier.

5) Écrivez deux programmes, "xmit" et "rcv". Le programme xmit devrait chercher un nom de fichier sur la ligne de commande et transmettre ce fichier à travers le port série. Il devrait transmettre le nom de fichier et le nombre de bytes dans le fichier (conseil: employez la commande de recherche du DOS pour déterminer la longueur du fichier). Le programme rcv devrait lire le nom de fichier et la longueur de fichier depuis le port série, créer un fichier du nom indiqué, lire le nombre indiqué de bytes depuis le port série et enfin fermer le fichier.

13.8 Résumé

MS-DOS et le BIOS fournissent beaucoup de services de système qui contrôlent le matériel sur un PC. Ils fournissent une interface flexible et indépendante de la machine. Malheureusement, le PC a beaucoup évolué depuis les premiers jours du PC original 8088 de 5 mégahertz d'IBM. Beaucoup des appels du BIOS et de DOS sont maintenant obsolètes, après avoir été remplacés par de nouveaux appels. Pour assurer la compatibilité antérieure,

MS-DOS et le BIOS supportent généralement les plus vieux appels obsolètes comme les plus récents. Cependant, vos programmes ne devraient pas utiliser les appels obsolètes, ils ne sont là que pour la compatibilité antérieure.

Le BIOS fournit beaucoup de services destinés au contrôle des périphériques tels que l'affichage vidéo, le port d'imprimante, le clavier, le port série, l'horloge en temps réel, etc... Les descriptions des services du BIOS pour ces périphériques apparaissent dans les sections suivantes:

- "INT 5- Impression d'écran" à la section 13.2.1
- "INT 10h- Services vidéo" à la section 13.2.2
- "INT 11h- Matériel installé" à la section 13.2.3
- "INT 12h- Mémoire disponble" à la section 13.2.4
- "INT 13h- Services disque de bas niveau "à la section 13.2.5
- "INT 14h- E/S série" à la section 13.2.6
- "INT 15h- Services divers" à la section 13.2.7
- "INT 16h- Services clavier" à la section 13.2.8
- "INT 17h- Services Imprimante" à la section 13.2.9
- "INT 18h- Lancer le BASIC" à la section 13.2.10
- "INT 19h - Redémarrer l'ordinateur" à la section 13.2.11
- "INT 1Ah- Horloge temps réel" à la section 13.2.12

MS-DOS fournit plusieurs types de services. Ce chapitre s'est concentré sur les services d'E/S de fichier fournis par ce système. En particulier, il a traité de la mise en application d'opérations E/S de fichier efficaces en utilisant les E/S bloquées. Pour apprendre comment exécuter des E/S de fichier et effectuer d'autres opérations de MS-DOS, voyez les sections suivantes:

- "Séquence d'appel de MS-DOS " à la section 13.3.1
- "Fonctions orientées caractère de MS-DOS" à la section 13.3.2
- "Les fonctions de gestion de fichiers "obsolètes" de MS-DOS" à la section 13.3.4
- "Fonctions de gestion de date et d' heure de MS-DOS" à la section 13.3.5
- "Fonctions de gestion de la mémoire de MS-DOS" à la section 13.3.6
- " Fonctions de contrôle des processus de MS-DOS" à la section 13.3.7
- ""Nouvelles" Fonctions de gestion de fichiers de MS-DOS" à la section 13.3.8
- " Exemples d'E/S de fichiers " à la section 13.3.9
- " E/S de fichier bloquées" à la section 13.3.10

Accéder aux paramètres de ligne de commande est une opération importante pour des applications MS-DOS. Le PSP DOS (préfixe de segment de programme) contient la ligne de commande et plusieurs autres informations importantes. Pour connaître les divers champs dans le PSP et voir comment accéder aux paramètres de la ligne de commande, voyez les sections suivantes en ce chapitre:

- "Le Préfixe de segment de programme (PSP)" à la section 13.3.11
- "Accès aux paramètres de la ligne de commande" à la section 13.3.12
- "ARGC et ARGV" à la section 13.3.13

Naturellement, la Bibliothèque standard de l'UCR fournit aussi des routines d'E/S de fichier. Ce chapitre se termine par la description de certaines des routines d'E/S de fichier de StdLib avec leurs avantages et inconvénients. Voir

- "Fopen" à la section 13.4.1
- "Fcreate" à la section 13.4.2
- "Fclose" à la section 13.4.3
- "Fflush" à la section 13.4.5
- "Fgetc" à la section 13.4.6
- "Fread" à la section 13.4.7
- "Fputc" à la section 13.4.8
- "Fwrite" à la section 13.4.9
- "Redirection des E/S par les routines d'E/S de fichier de la stdLib" à la section 13.4.10
- " Example d'E/S de fichier " à la section 13.4.11

13.9 Questions

1) Comment appelle-t-on les routines du BIOS?
2) Quelle routine du BIOS est utilisée pour écrire à un caractère au:
   a) port d'imprimante b) port série c) affichage vidéo
3) Quand les services de transmission ou réception série reviennent à l'appelant, l'état d'erreur est retourné dans le registre AH. Cependant, il y a un problème avec la valeur retournée. Quel est ce problème ?
4) Expliquez comment vous pourriez examiner le clavier pour voir si une touche est disponible.
5) Quel est le problème avec la fonction d'état des majuscules du clavier ?
6) Comment les principaux codes spéciaux (ces frappes qui ne renvoyent pas de codes ASCII) sont-ils retournés par l'appel à la lecture du clavier ?
7) Comment enverriez-vous un caractère à l'imprimante ?
8) Comment lisez-vous l'horloge en temps réel ?
9) Étant donné que la RTC incrémente un compteur de 32 bits tous les 55ms, combien de temps le système fonctionne-t-il avant que le débordement de ce compteur se produise ?
10) Pourquoi devez-vous remettre à zéro l'horloge si, en lisant l'horloge, vous avez déterminé que le compteur a débordé ?
11) Comment les programmes assembleur appellent-ils MS-DOS ?
12) Où les paramètres sont-ils généralement passés à MS-DOS ?
13) Pourquoi y a-t-il deux ensembles de fonctions de gestion de fichiers dans MS-DOS ?
14) Où peut-on trouver la ligne de commande du DOS?
15) Quel est la fonction de la zone de la chaîne d'environnement?
16) Comment pouvez-vous déterminer la quantité de mémoire disponible à l'usage de votre programme?
17) Quel sont les plus efficaces: les E/S par caractère ou les E/S bloquées? Pourquoi?
18) Quel est une bonne taille de bloc pour les E/S bloquées ?
19) Pourquoi ne pouvez-vous pas employer des E/S bloquées avec des fichiers à accès aléatoire?
20) Expliquez comment utiliser la commande *Rechercher* pour déplacer le pointeur de fichier 128 bytes en arrière de la position actuelle dans fichier.
21) Où l'état d'erreur est-il normalement retourné après un appel au DOS ?
22) Pourquoi est-il difficile d'utiliser les E/S bloquées sur un fichier à accès aléatoire? Qu'est-ce qui serait le plus facile, un accès aléatoire sur un fichier en E/S bloquées ouvert pour les entrées ou un accès aléatoire sur un fichier en E/S bloquées ouvert en lecture et écriture?
23) Décrivez comment vous pourriez mettre en application les E/S bloquées sur des fichiers ouverts pour la lecture et l'écriture en accès aléatoire.
24) Quelles sont deux manières par lesquelles vous pouvez obtenir l'adresse du PSP ?
25) Comment déterminez-vous que vous avez a atteint la fin du fichier quand vous utilisez les appels d'E/S de fichier de MS-DOS? Quand quand vous utilisez les appels d'E/S de fichier de la Bibliothèque standard UCR?