Conception d'un système d'exploitation : le cas de MS-DOS 1.25

Patrick Cégielski

Mars 2019

Pour Irène et Marie

Legal Notice

Université Paris XII – IUT de Sénart-Fontainebleau
Route forestière Hurtault
F-77300 Fontainebleau
cegielski@u-pec.fr

Table des matières

Table des figures

Préface

On peut distinguer cinq niveaux en ce qui concerne les rapports avec un système d'exploitation :

- le *niveau utilisateur*, dans lequel le but principal consiste à charger les logiciels dont on a besoin et de les faire exécuter ; il faut savoir utiliser et manipuler les fichiers pour cela ; on se sert de l'*interpréteur de commandes* et de *commandes* telles que `copy`, `rename`... ;

- le *niveau administrateur*, consistant à paramétrer le système et à le tenir à jour ; c'est évidemment très simple pour MS-DOS (par rapport à Unix, par exemple) : il suffit de remplir correctement les deux fichiers `autoexec.bat` et `config.sys` ;

- le *niveau écriture de scripts* pour automatiser certaines séquences répétitives de commandes ; ceci est réalisé en MS-DOS avec des fichiers de *traitement par lot* (*batch* en anglais) d'extension `.bat` ;

- le *niveau programmation système*, en utilisant les *appels systèmes* ; cette programmation se fait pour MS-DOS en langage d'assemblage en utilisant des *interruptions logicielles*, nouvelles par rapport à celles du BIOS, et éventuellement en langage évolué tel que le langage C, toujours en utilisant ces mêmes interruptions ;

- le *niveau conception du système* consiste à concevoir un tel système d'exploitation.

Nous allons nous intéresser ici à la conception d'un système d'exploitation relativement simple, à savoir MS-DOS 1.25.

Il s'agit d'un système d'exploitation simple, certains disent même rudimentaire, puisqu'il s'agit d'un système en ligne de commande, mono-utilisateur et mono-tâche. Mais c'est la raison pour laquelle nous commençons à l'étudier du point de vue de la conception, ainsi que les concepts qui lui sont associés.

Les sources de MS-DOS 1.25

Le code source de MS-DOS est resté longtemps inédit, contrairement à celui du BIOS. Mais le 25 mars 2014, Microsoft décide de publier le code source original des versions 1.25 (équivalent à PC-DOS 1.1) et 2.00 sur le site *Computer History*, sous une licence non libre n'autorisant ni les usages commerciaux, ni le partage :

`http://www.os2museum.com/wp/dos/dos-3-3/`

En fait le code source n'est pas complet : celui du secteur d'amorçage n'est pas publié ; seuls sont publiés IO.ASM, MSDOS.ASM et COMMAND.ASM pour MS-DOS 1.25 ; d'autres fichiers source sont publiés, mais pas ceux-ci, pour MS-DOS 2.0.

0.1 Bibliographie

[Tan-87] TANENBAUM, Andrew, **Operating Systems. Design and Implementation**, Prentice-Hall, 1987. Tr. fr. **Les systèmes d'exploitation : conception et mise en œuvre**, InterÉditions, 1989, XI + 756 p.

[Les principes des systèmes d'exploitation sont illustrés par *Minix*, version de Unix pour IBM PC créée à cette occasion par Tanenbaum. Le livre contient le source de la première version de *Minix*.]

[Tan-92] TANENBAUM, Andrew, **Modern Operating Systems**, Prentice-Hall, 1992. Tr. fr. **Systèmes d'exploitation : systèmes centralisés, systèmes distribués**, InterÉditions, 1994, XII + 795 p.

[La différence avec le livre précédent (dont il reprend une partie) est expliquée dans la préface : « *Pour être franc, je dois dire qu'à l'origine je souhaitais réviser l'un de mes ouvrages précédents (*Les systèmes d'exploitation*) duquel je souhaitais supprimer tout ce qui concernait MINIX, pour que ce livre soit plus proche d'un cours « théorique ». Au cours de cette révision, j'ai pris conscience de l'importance des systèmes répartis au point que j'ai ajouté sept chapitres consacrés à ce sujet.* »

Il n'y a donc plus MINIX (ni le source, bien sûr) mais une étude des cas de Unix et de MS-DOS.]

[TW-97] TANENBAUM, Andrew & WOODHULL, Albert, **Operating Systems. Design and Implementation, second edition**, Prentice-Hall, 1997, xviii + 940 p + CD-ROM.

[Le livre contient le source de la seconde version de *Minix*, dont TANENBAUM regrette qu'elle ait trop évoluée pour un cours. Le CD-ROM en contient le source et une version exécutable.]

[SG-98] SILBERSCHATZ, Abraham & GALVIN, Peter Baer, **Operating System Concepts**, Addison-Wesley, fifth edition, 1998, xvii + 888 p. Tr. fr. de la quatrième édition.

Il n'existe pas de description officielle de l'implémentation de MS-DOS, mais seulement une description des fonctions dans les manuels techniques d'IBM puis de Microsoft.

On dispose de références sur des produits analogues :

[Pod-95] PODANOFFSKY, Michael, **Disecting DOS : A Co-de-Level Look At The Dos Operating System**, Addison-Wesley, 1995, x + 502 p. + diskette.

[Commentaire de RXdos, dont le source est complet dans la disquette.]

Chapitre 1

Le démarrage du système d'exploitation

Nous allons voir comment charger et exécuter une système d'exploitation au démarrage de l'ordinateur. Nous avons déjà vu que le BIOS initialise les périphériques puis charge un programme.

1.1 Principe du chargement de l'enregistrement de démarrage

1.1.1 Rappels sur le chargement d'un système d'exploitation

Les étapes de la **procédure de démarrage** (en anglais *System Startup Procedure*) d'un ordinateur muni d'un microprocesseur *Intel* et du BIOS de l'IBM-PC sont les suivantes :

— Lors de la mise sous tension de la machine, nous avons vu, lors de l'étude des microprocesseurs, que le microprocesseur *Intel* exécute l'instruction située à l'adresse `FFFF0h`.

Cette adresse doit donc toujours être située en ROM.

L'instruction qui s'y trouve, située en fin de mémoire, ne peut être qu'un saut vers une adresse antérieure, début d'un code, situé en ROM BIOS également, d'initialisation de l'ordinateur.

— Comme nous l'avons vu, lors de l'étude du BIOS (de l'IBM-PC), celui-ci effectue quelques tests sur le matériel (microprocesseur, ROM, RAM, coupleurs), initialise les périphériques, établit deux emplacements de données du BIOS, à savoir la table des vecteurs d'interruption et la zone de communication du BIOS, et, si tout s'est bien passé, essaie de lire le secteur de démarrage de la disquette `A:`. S'il n'y parvient pas, il donne la main à l'interpréteur BASIC, également situé en ROM.

À partir de MS-DOS 2.0, s'il n'y a pas de disquette dans le lecteur `A:`, le BIOS essaie de lire le premier secteur de démarrage du disque dur. La *table de partition* du disque dur, située sur le premier secteur de celui-ci, lui indique où sont les *partitions* et quelle est la *partition active*. Celle-ci est alors choisie et son premier secteur (appelé *deuxième secteur de démarrage*) est lu et exécuté. Cette procédure en deux étapes, avec partitions, est utilisée uniquement pour les disques durs : elle permet de lancer automatiquement soit MS-DOS, soit un autre système d'exploitation.

— L'interruption du BIOS appelée **chargeur de démarrage** (en anglais *bootstrap loader*) charge à l'adresse 7C00h l'**enregistrement de démarrage** (en anglais *boot record*) du disque ayant été désigné comme **disque de démarrage** (en anglais *startup drive*) et lui donne la main.

Le BIOS permet, comme son nom l'indique, les entrées-sorties de base, en particulier la lecture au clavier, l'affichage à l'écran et la lecture et l'écriture de secteurs sur disquette et sur disque dur. Ceci est suffisant pour charger le système d'exploitation. Ce dernier se trouve, en plusieurs parties (fichiers), sur une disquette ou, de nos jours, sur le disque dur, un CD-ROM, un disque SSD ou une clé USB. Ce qui est important, c'est que les interruptions logicielles du BIOS permettent de charger le secteur de démarrage.

Lors de l'appel de l'interruption logicielle de chargement, à savoir l'interruption `int 19h`, le BIOS lit le premier secteur de la disquette située dans le premier lecteur de disquettes (du disque dur, du CD-ROM, du disque SSD ou de la clé USB dans les BIOS ultérieurs) et vérifie que les deux derniers octets comportent le **nombre magique** `55 AAh`, signature d'un secteur de démarrage.

S'il contient le nombre magique, l'interruption logicielle de chargement charge le premier secteur de ce disque en mémoire centrale à l'adresse `07C0:000` et effectue un saut à cette adresse. Le code situé sur ce premier secteur est donc exécuté.

1.1.2 Premier exemple

1.1.2.1 Grandes étapes de notre premier exemple

Nous allons :

— 1^o) Écrire un programme, en langage d'assemblage, n'utilisant que les instructions du microprocesseur et les interruptions logicielles du BIOS (sans utiliser aucune interruption de MS-DOS, ou de tout autre système d'exploitation), écrivant 120 fois la lettre 'b' en brun sur fond bleu.

— 2^o) Le compiler, à la main ou, mieux, en utilisant un assembleur et en récupérant la partie en langage machine correspondante.

— 3^o) Placer ce programme, sans oublier le nombre magique, sur le premier secteur d'une disquette formatée (à bas niveau, ce qui est indépendant du système d'exploitation).

Puisqu'il y a peu de chance que le lecteur possède encore un PC muni d'un lecteur de disquette, nous verrons également comment utiliser une clé USB à la place.

— 4^o) Démarrer l'ordinateur avec cette disquette placée dans le premier lecteur, ou la clé USB à l'emplacement adéquat.

Si tout fonctionne bien, on devrait voir apparaître les 120 'b' et tout s'arrête là. Il faudra ensuite redémarrer l'ordinateur (sans la disquette ou la clé USB).

1.1.2.2 Écriture du programme

Le programme en langage d'assemblage pour afficher 120 fois la lettre 'b' en brun sur fond bleu est le suivant, saisi par exemple en utilisant `debug` :

```
17A6:0100 MOV     AH,00
17A6:0102 MOV     AL,03
17A6:0104 INT     10
17A6:0106 MOV     AH,09
17A6:0108 MOV     AL,62
17A6:010A MOV     BL,16
17A6:010C MOV     BH,00
17A6:010E MOV     CX,0078
17A6:0111 INT     10
17A6:0113 INT     03
```

Les trois premières lignes permettent de se placer en mode graphique 3. Les six lignes suivantes permettent d'afficher (fonction `09h` de l'interruption logicielle `10h`) 120 fois (soit `78h` en hexadécimal, à placer dans le registre `cx`) la lettre 'b' minuscule (de code ASCII `62h` à placer dans le registre `al`) en brun sur fond bleu (`16h` à placer dans le registre `bl`) sur la page 0 (numéro à placer dans le registre `bh`). La dernière ligne dit au microprocesseur de s'arrêter là, sinon on risque un crash du système.

1.1.2.3 Programme en langage machine

On peut traduire ce programme, écrit en langage d'assemblage, en langage machine à la main. Le mieux est cependant d'utiliser `debug` ou un assembleur. Toujours est-il que le programme en langage machine est le suivant :

```
B4 00 B0 03 CD 10 B4 09 B0 62 B3 16
B7 00 B9 78 00 CD 10 CD 03
```

1.1.2.4 Placement du programme sur le premier secteur de la disquette

Pour placer le programme sur le premier secteur d'une disquette (formatée), on peut utiliser toutes les techniques que l'on veut. Voyons une technique relativement rudimentaire utilisant `debug`.

Première étape.- Cherchons un emplacement de mémoire vive de 512 octets inoccupé, en parcourant avec la fonction `d` de `debug`. Par exemple :

```
C:> debug
-d 2000:0000
```

Si on ne voit que des zéros alors cet emplacement de 128 octets est libre. Qu'en est-il de la suite ? Continuons à explorer la mémoire vive :

```
-d
```

Si on ne voit encore que des zéros alors l'emplacement précédent de 256 octets est libre. Qu'en est-il de la suite ? Continuons à explorer la mémoire vive :

```
-d
```

Si on ne voit encore que des zéros alors l'emplacement précédent de 384 octets est libre. Qu'en est-il de la suite ? Continuons à explorer la mémoire vive :

```
-d
```

Si, enfin, on ne voit encore que des zéros alors l'emplacement précédent de 512 octets est libre. On va pouvoir l'utiliser pour y placer momentanément le contenu du secteur de démarrage.

Deuxième étape.- On entrepose le programme en langage machine à cet endroit, y compris le nombre magique à la fin :

```
> debug
- E 2000:0000 B4 00 B0 03 CD 10 ...

- E 2000:01F0 55 AA
- q
```

Troisième étape.- On place ce contenu de la mémoire sur le premier secteur d'une disquette formatée (au moins à bas niveau). Pour cela on écrit un programme en langage d'assemblage et on le fait exécuter :

```
> debug
- a
167F:0100 mov ax,2000
          mov es,ax     ; segment
          mov bx,0      ; decalage
          mov ah,3      ; ecriture secteur
          mov al,1      ; 1 secteur
          mov ch,0      ; cylindre 0
          mov cl,1      ; secteur 1
          mov dh,0      ; tete 0
          mov dl,0      ; disque a
          int 13        ; ecriture du secteur
          int 3         ; arret
- g
```

Remarquons que si tout se passe bien, on ne peut pas vérifier ce qui a été écrit avec la commande `L` de `debug`, puisqu'il ne s'agit plus d'une disquette formatée MS-DOS.

1.1.2.5 Test

On fait redémarrer l'ordinateur avec cette disquette dans le lecteur `A:`. On voit apparaître les 120 'b' comme prévu. Évidemment on ne peut pas faire grand chose d'autre. On est obligé de redémarrer l'ordinateur, par exemple grâce au bouton `reset`.

1.1.3 Cas d'une clé USB

Le cas précédent fonctionne très bien si on possède un ordinateur avec un lecteur de disquette, mais ce qui est de plus en plus rare. Par contre, les systèmes modernes possèdent des **ports USB** (*Universal Serial Bus*) dans lesquels on peut introduire des **clés USB** qui les remplacent avantageusement : transfert plus rapide et capacité de stockage beaucoup plus grande. Une clé USB ne fonctionne pas comme une disquette ou un disque dur mais, une fois formatée en FAT16 (en fait à bas niveau), ce qui est le cas lorsqu'on les achète en général, le BIOS émule le comportement d'un disque dur. On va donc pouvoir écrire ce que l'on veut sur le secteur de démarrage, tout au moins ce qui en tient lieu, d'une clé USB.

1.1.3.1 Démarrage de l'ordinateur sur une clé USB

Lorsqu'on démarre un système informatique, le BIOS (ou UEFI) lit de nos jours le secteur de démarrage du (premier) disque dur ou disque SSD sur lequel se trouve le chargeur du système d'exploitation ou un utilitaire de démarrage multiple, tel que GRUB, dans le cas d'un système à démarrage multiple (si on veut avoir le choix entre plusieurs systèmes d'exploitation, Windows et Linux par exemple).

Il faut mettre le système d'exploitation régulièrement à jour mais ce système de démarrage est suffisant. Par contre si le système est altéré pour une raison ou une autre (et on espère en général que cela n'arivera jamais car c'est toujours assez délicat, et en tous cas stressant, d'y remédier) ou si on veut changer de système d'exploitation (l'ordinateur est livré avec Windows mais on veut Linux), il faut pouvoir démarrer d'une autre façon.

Sur les premiers systèmes on utilisait des disquettes, puis le lecteur de CD-ROM, équipements qui ont bien souvent disparus des systèmes actuels. De nos jours, on démarre donc alors sur un équipement USB.

Changement de l'ordre de démarrage.- Pour ne pas passer son temps à chercher l'équipement par lequel commencer, un choix est effectué avec le disque dur (ou SSD) en premier. Il faut changer ce choix.

Pour cela, au démarrage il faut aller dans l'interface du BIOS. Ce n'est pas difficile mais dépend du BIOS lui-même. En général il faut appuyer sur la touche ESCAPE (*Échap.*), la touche F2 ou la touche DEL (*Suppr*) une fois qu'on a appuyé sur le bouton d'alimentation.

Cas de UEFI.- Pour tout compliquer, les systèmes démarrant sous Windows n'utilisent plus le BIOS mais UEFI. Dans ce cas, on verra bien les clés USB mais uniquement sous la forme « ? ? », ce qui ne convient pas.

Il faut désactiver « scanboot », le nom exact et la façon de faire dépendant du BIOS/UEFI mais il est toujours possible de le faire. On pourra alors choisir le démarrage sur le secteur de démarrage d'une clé USB.

Que voit-on ?.- Pour l'instant on n'aura rien d'extraordinaire sinon le message :

mais cela montre que nous sommes sur la bonne voie.

1.1.3.2 Les éditeurs hexadécimaux

Nous aurons besoin d'un **éditeur hexadécimal**.

Nous sommes habitués à utiliser un éditeur de texte, en particulier pour écrire des programmes source : ils manipulent les caractères dits *affichables* et les passages à la ligne. Dans la suite, nous allons devoir analyser et écrire des fichiers *binaires*, pour lesquels les éditeurs de texte ne donnent pas grand chose à part une suite de signes incompréhensibles.

Une éditeur hexadécimal graphique comporte un *bandeau* avec les menus déroulants habituels (*File*, *Edit*, *Search*, *View*,...) et une fenêtre active partagée en trois parties horizontales : la première partie donne le décalage du premier octet de la ligne, la deuxième partie seize octets, chacun étant représenté sous la forme de deux chiffres hexadécimaux, et la troisième partie le caractère ASCII correspondant lorsque celui-ci est affichable, un point sinon.

Sous Windows, les deux éditeurs hexadécimaux les plus utilisés sont *WinHex* et *HxD*.

1.1.3.3 Cygwin

On pouvait jusqu'à récemment manipuler les secteurs de démarrage, à ses risques et périls, des disques durs et autres sous Windows. Puis Microsoft a complètement bridé cette fonctionnalité, ainsi que d'autres d'ailleurs. On peut encore le faire avec Linux et la commande `dd`. Pour retrouver cette commande, et d'autres commandes Unix, sous Windows, le plus simple est d'installer *Cygwin*

Cygwin est une collection de logiciels libres, à l'origine développés par *Cygnus Solutions*, permettant à différentes versions de Windows de Microsoft d'émuler les utilitaires Unix. Son site Web est :

`https://www.cygwin.com/`

1.1.3.4 Récupération du secteur de démarrage d'une clé USB

On peut utiliser la commande `dd` pour récupérer le secteur de démarrage d'une clé USB.

Repérer la bonne clé.- Une fois *Cygwin* installé, démarrez son invite de commande spécifique, appelée *Cygwin64 Terminal*, en tant qu'administrateur système.

Sans avoir encore placé la clé que vous voulez manipuler dans un port USB, effectuez la commande `cat` suivante :

```
$ cat /proc/partitions
major minor  #blocks  name   win-mounts

    8     0         0 sda
    8    16         0 sdb

Patrick@Patrick-PC ~
$
```

On voit apparaître deux équipements permanents, dont il ne faut surtout pas changer le secteur de démarrage. Recommencez après avoir introduit la clé dans un port USB. On voit maintenant :

```
$ cat /proc/partitions
major minor  #blocks  name   win-mounts

    8     0         0 sda
    8    16         0 sdb
    8    32         0 sdc

Patrick@Patrick-PC ~
```

La clé USB est donc `sdc`, au moins pour cette session.

Récupérationdu MBR.- En efectuant la commande `dd` suivante :

```
$ dd if=/dev/sdc of=./mbr.bin bs=512 count=1
1+0 enregistrements lus
1+0 enregistrements écrits
512 bytes copied, 0,0061838 s, 82,8 kB/s

Patrick@Patrick-PC ~
$
```

une copie du premier secteur de la clé, celui correspondant à son secteur de démarrage, est enregistré dans le fichier `mbr.bin`, dans le répertoire `cygwin64/home/Patrick` (dans mon cas).

Conservez-le soigneusement ! Ne le surchargez pas car on le remettra en place sur la clé ultérieurement.

1.1.3.5 Visualisation du contenu du secteur de démarrage

On peut utiliser *WinHex* par exemple pour visualiser le contenu du fichier binaire `mbr.bin`. Pour cela on clique sur le menu déroulant *File* du bandeau, on choisit *Open* et on cherche comme d'habitude le fichier que l'on veut ouvrir. On obtient ce qui est représenté sur la figure ci-dessous :

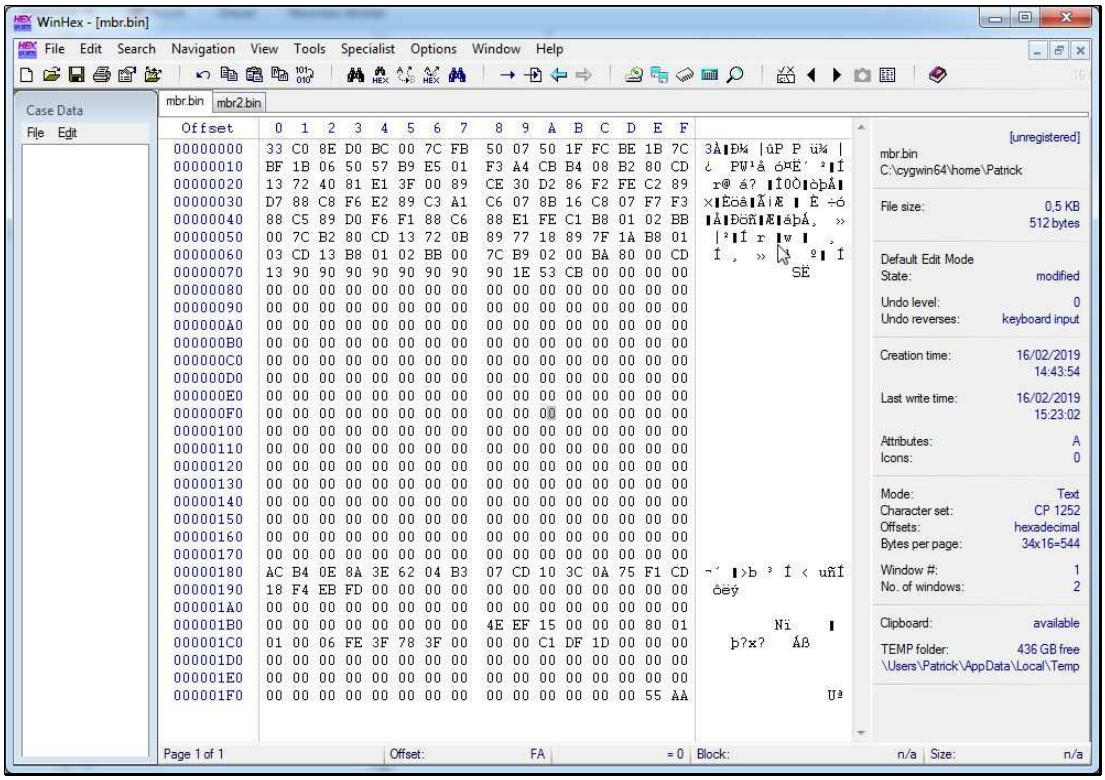

FIGURE 1.1 – Visualisation du secteur de démarrage

1.1.3.6 Changement du contenu du secteur de démarrage

Ne surtout pas effectuer les changements sur le fichier `mbr.bin` qui doit être sauvegardé!

Dans le menu déroulant, choisissez *Save as...* et sauvegardez une copie du fichier, par exemple sous le nom `mbr2.bin`. Nous pouvons alors effectuer des modifications sur ce nouveau fichier.

Placez le curseur devant le premier octet et copiez le programme : B4 00 B0... Il est inutile de placer les deux octets 55 et AA à la fin puisqu'ils existent déjà.

Ne pas oubliez de sauvegardez les changements!

1.1.3.7 Mise en place du nouveau secteur de démarrage

Pour enregistrer ce nouveau secteur de démarrage sur la clé USB, on utilise à nouveau la commande `dd` de *Cygwin* :

```
$ dd if=./mbr2.bin of=/dev/sdc bs=512 count=1
1+0 enregistrements lus
1+0 enregistrements écrits
512 bytes copied, 0,00803538 s, 63,7 kB/s

Patrick@Patrick-PC ~
$
```

1.1.3.8 Test

On fait redémarrer l'ordinateur avec cette clé USB placée dans un port USB et on choisit cet équipement de démarrage après avoir fait apparaître l'interface du BIOS. On voit alors affichés les 120 'b' comme prévu. Évidemment on ne peut pas faire grand chose d'autre. On est obligé de redémarrer l'ordinateur.

1.2 Quelques exemples de secteurs de démarrage

Voyons quelques exemples de secteurs de démarrage plus sophistiqués.

1.2.1 Deuxième exemple : disquette avec redémarrage

La façon de traduire le programme de démarrage en langage machine vu à propos du premier exemple n'est pas très pratique. Il en est de même de la façon de le placer sur le premier secteur. Nous allons donc commencer par améliorer ces deux points.

Nous allons également améliorer le contenu du secteur de démarrage. Nous allons placer le **bloc des paramètres BIOS** (BPB, en anglais *BIOS Parameter Block*) du système MS-DOS à la place adéquate, ce qui rendra lisible la disquette par MS-DOS. D'autre part, la disquette indiquera qu'elle n'est pas une disquette système, attendra qu'on la retire et après frappe d'une touche essaiera à nouveau de charger un système d'exploitation.

1.2.1.1 Le programme

Écrivons un programme en langage d'assemblage :

```
; boot144.asm
.MODEL tiny

boot_loc  equ 7C00h

MAIN    segment
        assume cs:main, ds:main, ss:main
        jmp short boot_program
        nop
BIOS_Parameter_Block:
; pour une disquette 1,44 Mi0 uniquement
        db 'BOOT skg'
        dw 200h         ; nombre d'octets/secteur
        db 1            ; nombre secteurs/unite d'allocation
        dw 1            ; nombre de secteurs reserves
        db 2            ; 2 FAT
        dw 0E0h         ; # entrees dans le repertoire racine
        dw 0B40h        ; # secteurs/disquette
        db 0F0h         ; identificateur de format
        dw 9            ; # secteurs/FAT
        dw 12h          ; # secteurs/piste
        dw 2            ; # tetes
        db 22h dup (0)  ; emplacement reserve

boot_program:

        mov ax,cs       ; setup segment registers
        mov ds,ax
        mov ss,ax
        mov sp,boot_loc
        mov si,boot_loc + offset Message

        call display_string
        jmp  end_program

display_string:         ; ds:si pointe sur le message
        lodsb
        or al,al
        jz end_string_loop
        mov ah,0Eh
```

```
        mov bx,7
        int 10h
        jmp display_string
end_string_loop:
        ret

end_program:
        xor ax,ax
        int 16h         ; attend la frappe d'une touche

; revient au chargement du systeme d'exploitation
; s'il n'y a plus de disquette passe au disque dur

int 19h

CR      equ 0Dh
LF      equ 0Ah

Message db CR,LF,"Essai de chargement de systeme d'exploitation"
        db CR,LF,"Ceci n'est pas une disquette systeme !"
        db CR,LF,"Retirer la disquette et appuyer sur une touche"
        db CR,LF,0

        org 1FEh
Signature:
        db 55h,0AAh

MAIN    ends

        end
```

Remarques.- 1^o) Nous utilisons un modèle *tiny* puisque le programme doit être un programme `.com`. Il n'y a donc qu'un seul segment.

- 2^o) Le programme commence par un saut inconditionnel devant occuper trois octets (d'où l'instruction de non opération), pour être compatible avec le format d'une disquette MS-DOS.

- 3^o) Ce saut est suivi du bloc des paramètres BIOS, toujours pour des raisons de compatibilité. Nous ne rappelons pas ici la structure du bloc des paramètres BIOS mais il est commenté. Les valeurs sont celles d'une disquette 3,5 $''$ haute densité ; il faut évidemment les changer pour un autre type de disquettes.

- 4^o) Le secteur doit se terminer par la signature `55h,AAh`. On remarquera la façon de la positionner en indiquant une adresse (le décalage seulement) absolue.

- 5^o) Le programme lui-même n'est pas celui d'un vrai secteur d'amorçage. Il n'amorce rien ici. Il se contente d'écrire un message, d'attendre la frappe d'une touche et de faire appel à l'interruption `19h` de chargement.

1.2.1.2 Assemblage et placement sur le premier secteur

Assemblage.- On assemble le programme avec notre assembleur habituel, par exemple MASM 6.11, en lui indiquant que l'on veut un programme `.com` :

```
C:\>ml /AT boot144.asm
Microsoft (R) Macro Assembler Version 6.11
Copyright (C) Microsoft Corp 1981-1993.  All rights reserved.

 Assembling: boot144.asm

Microsoft (R) Segmented Executable Linker  Version 5.31.009 Jul 13 1992
Copyright (C) Microsoft Corp 1984-1992.  All rights reserved.

Object Modules [.obj]: boot144.obj/t
Run File [boot144.com]: "boot144.com"
List File [nul.map]: NUL
Libraries [.lib]:
Definitions File [nul.def]:
LINK : warning L4055: start address not equal to 0x100 for /TINY
```

On notera que l'assembleur fait remarquer que le programme ne commence pas à l'adresse `100h`. Rappelons que le fait qu'il commence à l'adresse `100h` est important pour un programme chargé par le système MS-DOS, puisque ce système d'exploitation place un préfixe occupant les 256 premiers octets du segment de code. Cela n'est pas nécessaire pour nous puisque le programme n'est pas chargé par MS-DOS. De plus si on le faisait, on perdrait 256 précieux octets alors que le programme de chargement ne peut pas occuper plus de 512 octets.

Placement sur le premier secteur.- Pour placer ce programme sur le premier secteur d'une disquette formatée MS-DOS, on peut se servir de `debug` :

```
C:\>debug
-n boot144.com
-L 100
-w 100 0 0 1
-q

C:\>
```

1.2.1.3 Comportement

Introduisons la disquette dans le premier lecteur de disquettes et faisons démarrer l'ordinateur (éventuellement avec réglage des paramètres du BIOS pour qu'il tente de charger un système d'exploitation depuis une disquette). Le message s'affiche.

Si on laisse la disquette dans le lecteur et qu'on appuie sur une touche (plus exactement une touche déjà présente sur un clavier à 83 touches), le message s'affiche à nouveau. En effet le chargeur réessaie la disquette `a:` et on a donc le même le résultat.

Si on enlève la disquette et qu'on appuie sur une touche, le chargeur passe au disque dur et charge le système habituel.

1.2.2 Troisième exemple : chargement de plusieurs secteurs

Le rôle du programme d'amorçage est de charger le système d'exploitation proprement dit, qui occupe évidemment plusieurs secteurs. Nous allons voir, dans l'exemple de cette section, le principe du chargement de plusieurs secteurs par le secteur d'amorçage.

Pour charger un gros programme, on peut en effet le placer sur plusieurs secteurs de la disquette d'amorçage. Le secteur d'amorçage a alors pour but de charger ces secteurs à un emplacement déterminé de la mémoire vive puis d'effectuer un saut à l'adresse du premier octet du programme chargé en mémoire vive.

Le programme ne peut pas être très gros puisque le chargement de chacun des secteurs (ou quelques groupes de secteurs) occupe plusieurs octets et que le secteur d'amorçage est limité à 512 octets. Pour les fichiers plus gros, on charge de la façon précédente un *chargeur* qui, lui, chargera le gros programme.

Nous allons concevoir une disquette d'amorçage permettant de charger un programme occupant le premier secteur de données des disquettes formatées MS-DOS. Pour cela nous allons écrire deux programmes : le programme `bootrun` sert à charger le deuxième programme, il sera placé sur le secteur d'amorçage ; le programme `bootsys` est le programme à exécuter, à placer sur le secteur convenable.

1.2.2.1 Les problèmes du chargement des fichiers système

La méthode précédente permet donc de charger le système d'exploitation mais elle n'est pas très flexible. Il faut en effet calculer les secteurs à charger, ceux-ci dépendant d'un type de disquette à l'autre.

Deux problèmes se posent pour le chargement des fichiers systèmes : leur présence et leur taille.

Présence des fichiers système.- En général on commence par vérifier que les fichiers système sont bien présents avant de les charger. La disquette du système d'exploitation MS-DOS possède elle-même la structure des disquettes MS-DOS. Les fichiers sont donc présents dans le répertoire.

Taille des fichiers système.- Un système d'exploitation évolue. La taille des fichiers systèmes évolue donc également. Le programme d'amorçage ne peut donc pas charger la totalité des fichiers système car il n'en connaît pas la taille. Il pourrait la connaître en allant consulter la FAT, mais la taille (512 octets) du programme d'amorçage ne le permet pas. On commence donc par charger une partie du premier fichier (quelques secteurs) et par lui donner la main ; cette première partie est un chargeur plus évolué.

1.2.2.2 Le programme `bootrun`

Le programme.- Écrivons un programme en langage d'assemblage, sans oublier de placer le BPB pour qu'on puisse utiliser les utilitaires MS-DOS :

```
; bootrun.asm

.MODEL small

bootsys segment at 7E0h
        assume cs:bootsys, ds:bootsys
        org 00h
run_bootsys label far
bootsys ends

boot_loc    equ 7C00h
bootsys_loc equ 7E00h

MAIN    segment
        assume cs:main, ds:main, ss:main
        jmp short boot_program
        nop
BIOS_Parameter_Block:
; pour une disquette 1,44 MiO uniquement
        db 'BOOT skg'
        dw 200h             ; nombre d'octets/secteur
        db 1                ; nombre secteurs/unite d'allocation
        dw 1                ; nombre de secteurs reserves
        db 2                ; 2 FAT
        dw 0E0h             ; # entrees dans le repertoire racine
        dw 0B40h            ; # secteurs/disquette
        db 0F0h             ; identificateur de format
        dw 9                ; # secteurs/FAT
        dw 12h              ; # secteurs/piste
        dw 2                ; # tetes
        db 22h dup (0)      ; emplacement reserve

boot_program:

        mov ax,cs           ; setup segment registers
        mov ds,ax
        mov es,ax
        mov ss,ax
        mov sp,boot_loc

        mov ax,0
        int 13h             ; reset disk system (ax <-- 0)
;       jc erreurs

load_sys:
        mov ax,0
```

```
        mov es,ax
        mov ah,02h          ; lire
        mov al,01h          ; un secteur
        mov bx,bootsys_loc  ; et le placer a 0000:7E00h+
        mov ch,00h          ; piste 0
        mov cl,10h          ; secteur 16 - debut de
                            ; l'emplacement des donnees
        mov dh,01h          ; cote 1
        mov dl,00h          ; disque 0
        int 13h
;       jc erreurs

        jmp far ptr run_bootsys ; execute bootsys.bin

erreurs:

        mov si,boot_loc + offset Message
        call display_string
        jmp end_program

display_string:             ; ds:si pointe sur un mot
        lodsb
        or al,al
        jz end_string_loop
        mov ah,0Eh
        mov bx,7
        int 10h
        jmp display_string
end_string_loop:
        ret

end_program:

        xor ax,ax
        int 16h             ; attend la frappe d'une touche

; revient au chargement du systeme d'exploitation
; s'il n'y a plus de disquette passe au disque dur
        int 19h

CR equ 0Dh
LF equ 0Ah

Message db CR,LF,"Fichier BOOTSYS.BIN non trouve"
        db CR,LF,"Ceci n'est pas une disquette systeme !"
        db CR,LF,"Retirer la disquette et appuyer sur une touche"
        db CR,LF,0

        org 1FEh
Signature:
```

```
        db 55h,0AAh

MAIN    ends
        end boot_program
```

Commentaires.- 1^o) Le programme chargé est placé, arbitrairement, à l'adresse `7E00h`, c'est-à-dire juste après le secteur d'amorçage (placé en `7C00h` et occupant 512 octets).

- 2^o) Pour pouvoir transférer le contrôle à cette adresse `7E00h`, on a besoin de définir un second segment, appelé ici `bootsys`, ayant une adresse absolue (grâce à la directive `at`).

- 3^o) Le fait qu'il y ait deux segments nous empêche d'utiliser le modèle `tiny`. On utilise donc le modèle `small`.

Assemblage.- On assemble le programme avec notre assembleur habituel, par exemple MASM 6.11, mais on ne peut pas lui indiquer que l'on veut un programme `.com`, car cela exige un modèle `tiny`. On obtient donc un exécutable `bootrun.exe` de 1 024 octets : les 512 octets qui nous intéressent, précédé des 512 octets de l'en-tête MS-DOS.

Transformation en programme `.com`.- Il faut donc enlever ces 512 premiers octets. Les premières versions de MS-DOS étaient fournies avec un utilitaire, appelé `exe2bin` permettant d'enlever les 512 premiers octets, mais ce n'est plus le cas des versions ultérieures.

On peut soit récupérer cet utilitaire d'une ancienne version de DOS, soit construire un utilitaire analogue, écrit par exemple en langage C :

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

void main(void)
   {
   const int prefix = 512;
   FILE *source, *but;
   char nsource[30], nbut[30];
   int i;
   char C;

   printf("Nom du fichier source : ");
   gets(nsource);
   printf("Nom du fichier but : ");
   gets(nbut);
   source = fopen(nsource,"rb");
   but = fopen(nbut,"wb");
   i = 0;
   while(!feof(source))
      {
      C = fgetc(source);
      i++;
      if (i > prefix) fputc(C,but);
      }
   fclose(source);
   fclose(but);
```

```
}
```

À la vérité ce programme ajoute un octet à la fin du fichier (indiquant la fin du fichier justement), mais ce n'est pas important puisqu'on ne le recopiera pas.

Après avoir compilé ce programme et en le faisant exécuter, on peut obtenir le programme voulu, de 513 octets, appelé par exemple `bootrun.com`.

Placement sur le premier secteur.- On fait comme dans l'exemple précédent, grâce à `debug`.

1.2.2.3 Le programme `bootsys`

Le programme.- Reprenons, par exemple, le programme permettant d'afficher 120 'b', écrit en utilisant `debug` :

```
17A6:0100 MOV     AH,00
17A6:0102 MOV     AL,03
17A6:0104 INT     10
17A6:0106 MOV     AH,09
17A6:0108 MOV     AL,62
17A6:010A MOV     BL,16
17A6:010C MOV     BH,00
17A6:010E MOV     CX,0078
17A6:0111 INT     10
17A6:0113 INT     03
```

que l'on sauvegarde sous le nom `bootsys.bin`, toujours avec `debug` :

```
-n bootsys.bin
-r cx
0000 : 14
-w
14 octets ecrits
```

Placement du programme.- L'intérêt du formatage MS-DOS est que l'on peut placer le programme sur la disquette grâce aux commandes MS-DOS :

```
copy bootsys.bin a:
```

à condition que la disquette utilisée vienne juste d'être formatée (sinon le fichier peut être placé sur un autre secteur).

On peut aussi utiliser `debug` :

```
c:>debug bootsys.bin
-w 100 0 21 1
```

Test.- On fait redémarrer l'ordinateur avec cette disquette dans le lecteur `a`. On voit apparaître les 120 'b' comme prévu.

1.3 Cas de MS-DOS

Dans le cas de MS-DOS, on ne dispose pas des sources, y compris après la diffusion tardive, en 2014, par Microsoft de la version 1.25. Nous sommes donc obligés de faire de l'ingénierie inverse ou rétro-ingénierie (en anglais *reverse engeneering*).

1.3.1 La rétro-ingénierie

1.3.2 Structure de la zone des données d'un secteur d'amorçage MS-DOS

Le secteur d'amorçage contient les informations permettant de charger (ou amorcer) les fichiers système IO.SYS, MSDOS.SYS et COMMAND.COM, depuis le disque vers la mémoire centrale. Tous les disques formatés contiennent un secteur d'amorçage, même si les fichiers système ne sont pas présents ; il s'agit toujours du premier secteur, comme nous l'avons vu.

Description.- Donnons la description du secteur d'amorçage octet par octet :

- `00h-02h` : ces trois octets contiennent une instruction de saut JMP, soit intersegmentaire `E9 XX XX`, soit court `EB XX 90`. Dans le cas d'un saut intersegmentaire, d'opcode `E9`, le déplacement tient sur deux octets. Dans le cas d'un saut court, d'opcode `EB`, le déplacement tient sur un octet ; dans ce cas, l'octet restant est `90h`, l'opcode de la non opération NOP.
- `03h-0Ah` : ces huit octets donnent le nom du fabricant et la version du système MS-DOS sous laquelle le disque a été formaté, le tout codé en ASCII.

  Notons que ce n'est pas très clair pour les disquettes achetées déjà formatées.
- `0Bh-0Ch` : ces deux octets spécifient le nombre d'octets par secteur, le plus souvent `00-20`, c'est-à-dire `200h` (rappelons-nous qu'il faut d'abord lire le deuxième octet en représentation petit boutienne), soit 512 octets.
- `0Dh` : cet octet spécifie le nombre de secteurs par unité d'allocation.
- `0Eh-0Fh` : ces deux octets spécifient le nombre de secteurs réservés.
- `10h` : cet octet spécifie le nombre de copies de la FAT (1 ou 2). En général il y en a deux, la FAT originelle et une copie, bien que la copie ne soit pas utilisée par MS-DOS (mais elle peut permettre de récupérer les données d'une disquette endommagée).
- `11h-12h` : ces deux octets spécifient le nombre maximum d'entrées dans le répertoire racine.
- `13h-14h` : ces deux octets spécifient le nombre total de secteurs sur le disque, si celui-ci est inférieur à 32 MiO ; les octets `20h` à `23h` sont utilisés dans le cas d'un dique de plus de 32 MiO, les octets `13h-14h` étant alors nuls. Cela permet, évidemment, de déterminer la capacité du disque.
- `15h` : cet octet est un descripteur du type de disque (on le retrouve comme premier octet de la FAT) :

Octet	Type de disque	Capacité
F0	disquette 3,5" double face, 18 secteurs/piste	1,44 MiO
	disquette 3,5" double face, 36 secteurs/piste	2,88 MiO
F8	disque dur	
F9	disquette 5,25" double face, 15 secteurs/piste	1,2 MiO
	disquette 3,5" double face, 9 secteurs/piste	720 KiO
FC	disquette 5,25" simple face, 9 secteurs/piste	180 KiO
FD	disquette 5,25" double face, 9 secteurs/piste	360 KiO
FE	disquette 5,25", 8 secteurs/piste	320 KiO
FF	disquette 5,25 double face, 8 secteurs/piste	

  Remarquons que `F0h` et `F9h` identifient chacun deux formats de disque.
- `16h-17h` : ces deux octets spécifient le nombre de secteurs occupés par chacune des FAT.
- `18h-19h` : ces deux octets spécifient le nombre de secteurs par piste.

— `1Ah-1Bh` : ces deux octets spécifient le nombre de têtes, soit 1 pour une disquette simple face, 2 pour une disquette double face et beaucoup plus pour un disque dur.

— `1Ch-1Fh` : ces quatre octets spécifient le nombre de secteurs cachés.

Jusqu'au DOS 3.3, les informations proprement dites se terminaient à l'octet `1Dh` et la routine de chargement commençait à l'octet `1Eh`, c'est-à-dire que les trois premiers octets constituaient un saut à l'octet `1Eh`. DOS 4.0 a étendu la zone des informations de `20h` à `1FFh` :

— `20h-23h` : ces quatre octets spécifient le nombre total de secteurs pour un disque d'une capacité plus grande que 32 MiO.

— `24h` : cet octet spécifie le numéro de lecteur (A = 0, B = 1, C = 2...).

— `25h` : cet octet est réservé au système.

— `26h` : cet octet est la signature du secteur d'amorçage étendu.

— `27A-2Ah` : ces quatre octets spécifient le numéro d'identification du disque.

— `2Bh-35h` : ces dix octets donnent le nom du disque (volume) en ASCII.

— `36h-3Dh` : ces huit octets sont réservés au système.

— `3Eh-??` : c'est le début du programme de chargement depuis le DOS 4.0.

Exemple.- Chargeons le secteur d'amorçage de la première disquette de mise à jour du D0S 6.0 avec `debug` et interprétons-le :

```
- L 100 0 0 1
- D100
17A5:0100 EB 3C 90 4D 53 44 4F 53-36 2E 30 00 02 01 01 00 .<.MSDOS6.0.....
17A5:0110 02 E0 00 40 0B F0 09 00-12 00 02 00 00 00 00 00 ...@............
17A5:0120 00 00 00 00 00 00 29 69-6B 6A 6D 4E 4F 20 4E 41 ......)ikjmNO NA
17A5:0130 4D 45 20 20 20 20 46 41-54 31 32 20 20 20 FA 33 ME FAT12 .3
17A5:0140 C0 8E D0 BC 00 7C 16 07-BB 78 00 36 C5 37 1E 56 .....|...x.6.7.V
17A5:0150 16 53 BF 3E 7C B9 0B 00-FC F3 A4 06 1F C6 45 FE .S.>|.........E.
17A5:0160 0F 8B 0E 18 7C 88 4D F9-89 47 02 C7 07 3E 7C FB ....|.M..G...>|.
17A5:0170 CD 13 72 79 33 C0 39 06-13 7C 74 08 8B 0E 13 7C ..ry3.9..|t....|
```

— Le premier octet est `EB`, le code opération du saut court. Le second octet, `3C`, est la valeur du déplacement pour trouver le début du programme d'amorçage, qui se trouve donc à l'adresse `13Eh`, qui commence par `FA`, comme on peut le constater trois lignes plus bas. Le troisième octet est le code de non opération NOP.
— Les cinq octets suivant donnent le nom du fabricant, ici « MSDOS » et les trois suivants le numéro de version, ici 6.0.
— Les deux octets suivants `00h` et `02h` spécifent le nombre d'octets par secteur, soit `0200h` ou 512.
— L'octet suivant, `01h`, spécifie le nombre de secteurs par unité d'allocation ; ici une unité d'allocation se confond avec un secteur.
— Les deux octets suivants, `01h` et `00h`, donnent le nombre de secteurs réservés, ici un seul.
— Le premier octet de la deuxième ligne, `02`, spécifie le nombre de FAT, ici 2.
— Les deux octets suivants, `E0h` et `00h` spécifent le nombre maximum d'entrées dans le répertoire racine, ici 224.
— Les deux octets suivants, `40h` et `0Bh`, spécifent le nombre total de secteurs sur le disque, ici `0B40h`, soit 2 880.
— L'octet suivant, `F0h`, spécifie le type de disquette, ici une disquette 3,5" double face de 2,88 MiO.
— Les deux octets suivants, `09h` et `00h`, spécifient le nombre de secteurs par FAT, ici 9, soit les secteurs 1 à 10 et 11 à 19.
— Les deux octets suivants, `12h` et `00h`, spécifient le nombre de secteurs par piste, soit ici 18.
— Les deux octets suivants, `02h` et `00h`, spécifient le nombre de têtes de lecture-écriture, soit ici 2.
— Les quatre octets suivants, `00h`, `00h`, `00h` et `00h`, spécifient le nombre de secteurs cachés ; ici il n'y en a pas.
— Les quatre premiers octets de la troisième ligne, `00h`, `00h`, `00h` et `00h`, spécifient le nombre total d'octets pour un disque de capacité plus grande que 32 MiO ; ici tous les octets sont donc nuls.
— L'octet suivant, `00h`, spécifie le numéro du lecteur de disque ; il s'agit donc ici du lecteur A.
— L'octet suivant est réservé par le système et est donc nul.
— L'octet suivant, `29h`, est la signature du secteur d'amorçage étendu.
— Les quatre octets suivants, `69h`, `6Bh`, `6A` et `6D`, spécifient le numéro d'identification du disque.
— Les dix octets suivants spécifient le nom du volume ; il n'y en a pas ici, mais on a « NO NAME » et en plus on utilise la fin et les octets réservés suivants pour rappeler qu'on a une FAT douze bits.

1.3.3 Code du secteur d'amorçage

Formatons une disquette haute densité, qu'importe le système d'exploitation utilisé, par exemple Windows 98. Cette disquette contient alors un secteur d'amorçage mais avec lequel il est difficile de travailler avec `debug` ou les autres utilitaires DOS.

1.3.3.1 Lecture du secteur d'amorçage

On peut lire ce secteur avec l'interruption `13h` du BIOS. Écrivons donc un programme, nommé `LitDOS.asm`, en langage d'assemblage, permettant de lire ce premier secteur, de le placer dans un fichier, nommé `dos.com`, sur la même disquette et d'en afficher le contenu.

```
TITLE LitDOS.asm
; lit le premier secteur du repertoire du disque A,
; le place en DOS.com
; et l'affiche
      .model small
      .stack 256
      .data
tampon  db 512 dup(?)
message db 'erreur de lecture', '$'
fichier db 'a:\dos.com',00h
numero  dw ?                ; descripteur de fichier
      .code
debut:
      mov ax, @data
      mov ds, ax
      mov es, ax

      mov ah, 2             ; lire secteur
      mov al, 1             ; un secteur
      mov bx, offset tampon ; a entreposer dans tampon
      mov ch, 0             ; cylindre 0
      mov cl, 1             ; secteur 1
      mov dh, 0             ; tete 0
      mov dl, 0             ; disque A
      int 13h               ; lecture du secteur
      jc erreurs            ; en cas d'erreur

; entrepose sur dos.com de a:

      mov ah,3Ch            ; creer fichier
      mov cx,00             ; attributs normaux
      lea dx,fichier        ; nom du fichier
      int 21h
      mov numero,ax

      mov ah,40h            ; ecrire dans fichier
      mov bx,numero         ; descripteur
      mov cx,512            ; longueur de l'enregistrement
      lea dx,tampon         ; adresse de l'enregistrement
      int 21h

      jmp affiche

erreurs:
      mov dx, offset message
      mov ah, 9h
      int 21h               ; affiche erreur de lecture
      jmp fin
```

```
; affiche le contenu

affiche:
      mov ah, 2h      ; affichage d'un caractere
      mov cl, 16      ; 16 lignes
nouvelle:
      mov dl, 13
      int 21h
      mov dl, 10
      int 21h         ; passage a la ligne
      mov ch, 32      ; 32 caracteres
ligne:
      mov dl, [bx]
      int 21h
      inc bx          ; caractere suivant
      dec ch
      cmp ch, 0
      jne ligne
      dec cl
      cmp cl, 0
      jne nouvelle

fin:
      mov ax, 4c00h
      int 21h
      end debut
```

Le fait de conserver le secteur d'amorçage dans un fichier va nous permettre de travailler avec. Nous pouvons maintenant faire afficher le contenu du secteur avec `debug` :

```
C:\>debug a:dos.com
-d 100
17CF:0100  EB 3E 90 29 6A 2E 32 38-49 48 43 00 02 01 01 00   .>.)j.28IHC.....
17CF:0110  02 E0 00 40 0B F0 09 00-12 00 02 00 00 00 00 00   ...@............
17CF:0120  00 00 00 00 00 00 29 34-1B 01 12 4E 4F 20 4E 41   ......)4...NO NA
17CF:0130  4D 45 20 20 20 20 46 41-54 31 32 20 20 20 F1 7D   ME    FAT12   .}
17CF:0140  FA 33 C9 8E D1 BC FC 7B-16 07 BD 78 00 C5 76 00   .3.....{...x..v.
17CF:0150  1E 56 16 55 BF 22 05 89-7E 00 89 4E 02 B1 0B FC   .V.U."..~..N....
17CF:0160  F3 A4 06 1F BD 00 7C C6-45 FE 0F 8B 46 18 88 45   ......|.E...F..E
17CF:0170  F9 FB 38 66 24 7C 04 CD-13 72 3C 8A 46 10 98 F7   ..8f$|...r<.F...
-d
17CF:0180  66 16 03 46 1C 13 56 1E-03 46 0E 13 D1 50 52 89   f..F..V..F...PR.
17CF:0190  46 FC 89 56 FE B8 20 00-8B 76 11 F7 E6 8B 5E 0B   F..V.. ..v....^.
17CF:01A0  03 C3 48 F7 F3 01 46 FC-11 4E FE 5A 58 BB 00 07   ..H...F..N.ZX...
17CF:01B0  8B FB B1 01 E8 94 00 72-47 38 2D 74 19 B1 0B 56   .......rG8-t...V
17CF:01C0  8B 76 3E F3 A6 5E 74 4A-4E 74 0B 03 F9 83 C7 15   .v>..^tJNt......
17CF:01D0  3B FB 72 E5 EB D7 2B C9-B8 D8 7D 87 46 3E 3C D8   ;.r...+...}.F><.
17CF:01E0  75 99 BE 80 7D AC 98 03-F0 AC 84 C0 74 17 3C FF   u...}.......t.<.
17CF:01F0  74 09 B4 0E BB 07 00 CD-10 EB EE BE 83 7D EB E5   t............}..
-d
17CF:0200  BE 81 7D EB E0 33 C0 CD-16 5E 1F 8F 04 8F 44 02   ..}..3...^....D.
17CF:0210  CD 19 BE 82 7D 8B 7D 0F-83 FF 02 72 C8 8B C7 48   ....}.}....r...H
17CF:0220  48 8A 4E 0D F7 E1 03 46-FC 13 56 FE BB 00 07 53   H.N....F..V....S
17CF:0230  B1 04 E8 16 00 5B 72 C8-81 3F 4D 5A 75 A7 81 BF   .....[r..?MZu...
17CF:0240  00 02 42 4A 75 9F EA 00-02 70 00 50 52 51 91 92   ..BJu....p.PRQ..
17CF:0250  33 D2 F7 76 18 91 F7 76-18 42 87 CA F7 76 1A 8A   3..v...v.B...v..
17CF:0260  F2 8A 56 24 8A E8 D0 CC-D0 CC 0A CC B8 01 02 CD   ..V$............
17CF:0270  13 59 5A 58 72 09 40 75-01 42 03 5E 0B E2 CC C3   .YZXr.@u.B.^....
-d
17CF:0280  03 18 01 27 0D 0A 49 6E-76 61 6C 69 64 20 73 79   ...'..Invalid sy
17CF:0290  73 74 65 6D 20 64 69 73-6B FF 0D 0A 44 69 73 6B   stem disk...Disk
17CF:02A0  20 49 2F 4F 20 65 72 72-6F 72 FF 0D 0A 52 65 70    I/O error...Rep
17CF:02B0  6C 61 63 65 20 74 68 65-20 64 69 73 6B 2C 20 61   lace the disk, a
17CF:02C0  6E 64 20 74 68 65 6E 20-70 72 65 73 73 20 61 6E   nd then press an
17CF:02D0  79 20 6B 65 79 0D 0A 00-49 4F 20 20 20 20 20 20   y key...IO
17CF:02E0  53 59 53 4D 53 44 4F 53-20 20 20 53 59 53 80 01   SYSMSDOS   SYS..
17CF:02F0  00 57 49 4E 42 4F 4F 54-20 53 59 53 00 00 55 AA   .WINBOOT SYS..U.
```

Remarquons que le programme commence par un saut inconditionnel et se termine par le mot magique `55 AAh`.

1.3.3.2 Données du secteur d'amorçage

Commençons par décrypter les données du secteur d'amorçage ne constituant pas le programme lui-même. Il faut évidemment connaître la structure du secteur d'amorçage de MS-DOS pour cela.

— Commençons par la première ligne. Le premier octet est `EB`, le code opération du saut inconditionnel court. Le second octet, `3E`, est la valeur du déplacement permettant de trouver le début du programme d'amorçage, donc à l'adresse `140h`. Ce programme d'amorçage commence par `FA`, comme on peut le constater trois lignes plus bas. Le troisième octet est le code de la non opération NOP, puisqu'il s'agissait d'un saut court.
— Les cinq octets suivant spécifient le nom du fabricant, ici curieusement «)j.28 » et les trois suivants le numéro de version, ici, tout aussi curieusement, « IHC ».
— Les deux octets suivants, `00h` et `02h`, spécifient le nombre d'octets par secteur, soit `0200h` ou 512.
— L'octet suivant, `01h`, spécifie le nombre de secteurs par unité d'allocation ; ici une unité d'allocation se confond avec un secteur.
— Les deux octets suivants, `01h` et `00h`, spécifient le nombre de secteurs réservés, ici un seul.
— Le premier octet de la deuxième ligne, `02`, spécifie le nombre de FAT, ici 2.
— Les deux octets suivants, `E0h` et `00h`, spécifient le nombre maximum d'entrées dans le répertoire racine, ici 224.
— Les deux octets suivants, `40h` et `0Bh`, spécifient le nombre total de secteurs sur le disque, ici `0B40h`, soit 2 880.
— L'octet suivant, `F0h`, spécifie le type de disquette, ici une disquette 3,5" double face de 2,88 MiO (bien qu'il s'agisse d'une disquette de 1,44 MiO).
— Les deux octets suivants, `09h` et `00h`, spécifient le nombre de secteurs par FAT, ici 9. Elle est donc constituée des secteurs 1 à 10 et sa copie des secteurs 11 à 19.
— Les deux octets suivants, `12h` et `00h`, spécifient le nombre de secteurs par piste, soit ici 18.
— Les deux octets suivants, `02h` et `00h`, spécifient le nombre de têtes de lecture-écriture, soit ici 2.
— Les quatre derniers octets de la deuxième ligne, ici `00h`, `00h`, `00h` et `00h`, spécifient le nombre de secteurs cachés ; ici il n'y en a donc aucun.
— Les quatre premiers octets de la troisième ligne, `00h`, `00h`, `00h` et `00h`, spécifient le nombre total d'octets pour un disque de capacité supérieure à 32 MiO ; ici les octets sont donc tous nuls.
— L'octet suivant, `00h`, spécifie le numéro du lecteur de disquettes ; il s'agit donc ici du lecteur A.
— L'octet suivant est réservé par le système et est donc nul.
— L'octet suivant, `29h`, est la signature du secteur d'amorçage étendu.
— Les quatre octets suivants, `34h`, `1Bh`, `01` et `12`, spécifient le numéro d'identification du disque.
— Les dix octets suivants spécifient le nom du volume ; il n'y en a pas ici, mais on a « NO NAME » et en plus on utilise la fin et les octets réservés suivants pour rappeler qu'on a une FAT douze bits.

1.3.3.3 Détermination du programme d'amorçage

Si on se sert de `debug` de façon abrupte pour désassembler le programme d'assemblage, on n'obtient rien, puisque `debug` commence par essayer de désassembler les données commentées ci-dessus. Il n'interprète pas correctement le code qui commence à l'adresse `140h`, en fait seules les deux premières lignes sont bien analysées :

```
C:\>debug a:dos.com
-u
17CF:0100 EB3E          JMP     0140
17CF:0102 90            NOP
17CF:0103 296A2E        SUB     [BP+SI+2E],BP
17CF:0106 3238          XOR     BH,[BX+SI]
17CF:0108 49            DEC     CX
17CF:0109 48            DEC     AX
17CF:010A 43            INC     BX
17CF:010B 0002          ADD     [BP+SI],AL
17CF:010D 0101          ADD     [BX+DI],AX
17CF:010F 0002          ADD     [BP+SI],AL
17CF:0111 E000          LOOPNZ  0113
17CF:0113 40            INC     AX
17CF:0114 0BF0          OR      SI,AX
17CF:0116 0900          OR      [BX+SI],AX
17CF:0118 1200          ADC     AL,[BX+SI]
17CF:011A 0200          ADD     AL,[BX+SI]
17CF:011C 0000          ADD     [BX+SI],AL
17CF:011E 0000          ADD     [BX+SI],AL
-u
17CF:0120 0000          ADD     [BX+SI],AL
17CF:0122 0000          ADD     [BX+SI],AL
17CF:0124 0000          ADD     [BX+SI],AL
17CF:0126 2934          SUB     [SI],SI
17CF:0128 1B01          SBB     AX,[BX+DI]
17CF:012A 124E4F        ADC     CL,[BP+4F]
17CF:012D 204E41        AND     [BP+41],CL
17CF:0130 4D            DEC     BP
17CF:0131 45            INC     BP
17CF:0132 2020          AND     [BX+SI],AH
17CF:0134 2020          AND     [BX+SI],AH
17CF:0136 46            INC     SI
17CF:0137 41            INC     CX
17CF:0138 54            PUSH    SP
17CF:0139 3132          XOR     [BP+SI],SI
17CF:013B 2020          AND     [BX+SI],AH
17CF:013D 20F1          AND     CL,DH
17CF:013F 7DFA          JGE     013B
```

En demandant le déassemblage à partir de `140`, nous trouvons du code plus compréhensible :

```
C:\>debug a:dos.com
-u 140
17CF:0140 FA            CLI
17CF:0141 33C9          XOR     CX,CX
17CF:0143 8ED1          MOV     SS,CX
17CF:0145 BCFC7B        MOV     SP,7BFC
17CF:0148 16            PUSH    SS
17CF:0149 07            POP     ES
17CF:014A BD7800        MOV     BP,0078
17CF:014D C57600        LDS     SI,[BP+00]
17CF:0150 1E            PUSH    DS
17CF:0151 56            PUSH    SI
17CF:0152 16            PUSH    SS
17CF:0153 55            PUSH    BP
17CF:0154 BF2205        MOV     DI,0522
17CF:0157 897E00        MOV     [BP+00],DI
17CF:015A 894E02        MOV     [BP+02],CX
17CF:015D B10B          MOV     CL,0B
17CF:015F FC            CLD
-u
17CF:0160 F3            REPZ
17CF:0161 A4            MOVSB
17CF:0162 06            PUSH    ES
17CF:0163 1F            POP     DS
17CF:0164 BD007C        MOV     BP,7C00
17CF:0167 C645FE0F      MOV     BYTE PTR [DI-02],0F
17CF:016B 8B4618        MOV     AX,[BP+18]
17CF:016E 8845F9        MOV     [DI-07],AL
17CF:0171 FB            STI
17CF:0172 386624        CMP     [BP+24],AH
17CF:0175 7C04          JL      017B
17CF:0177 CD13          INT     13
17CF:0179 723C          JB      01B7
17CF:017B 8A4610        MOV     AL,[BP+10]
17CF:017E 98            CBW
17CF:017F F76616        MUL     WORD PTR [BP+16]
-u
17CF:0182 03461C        ADD     AX,[BP+1C]
17CF:0185 13561E        ADC     DX,[BP+1E]
17CF:0188 03460E        ADD     AX,[BP+0E]
17CF:018B 13D1          ADC     DX,CX
17CF:018D 50            PUSH    AX
17CF:018E 52            PUSH    DX
17CF:018F 8946FC        MOV     [BP-04],AX
17CF:0192 8956FE        MOV     [BP-02],DX
17CF:0195 B82000        MOV     AX,0020
17CF:0198 8B7611        MOV     SI,[BP+11]
17CF:019B F7E6          MUL     SI
17CF:019D 8B5E0B        MOV     BX,[BP+0B]
17CF:01A0 03C3          ADD     AX,BX
-u
17CF:01A2 48            DEC     AX
17CF:01A3 F7F3          DIV     BX
17CF:01A5 0146FC        ADD     [BP-04],AX
17CF:01A8 114EFE        ADC     [BP-02],CX
17CF:01AB 5A            POP     DX
17CF:01AC 58            POP     AX
17CF:01AD BB0007        MOV     BX,0700
17CF:01B0 8BFB          MOV     DI,BX
17CF:01B2 B101          MOV     CL,01
17CF:01B4 E89400        CALL    024B
```

```
17CF:01B7 7247          JB      0200
17CF:01B9 382D          CMP     [DI],CH
17CF:01BB 7419          JZ      01D6
17CF:01BD B10B          MOV     CL,0B
17CF:01BF 56            PUSH    SI
17CF:01C0 8B763E        MOV     SI,[BP+3E]
-u
17CF:01C3 F3            REPZ
17CF:01C4 A6            CMPSB
17CF:01C5 5E            POP     SI
17CF:01C6 744A          JZ      0212
17CF:01C8 4E            DEC     SI
17CF:01C9 740B          JZ      01D6
17CF:01CB 03F9          ADD     DI,CX
17CF:01CD 83C715        ADD     DI,+15
17CF:01D0 3BFB          CMP     DI,BX
17CF:01D2 72E5          JB      01B9
17CF:01D4 EBD7          JMP     01AD
17CF:01D6 2BC9          SUB     CX,CX
17CF:01D8 B8D87D        MOV     AX,7DD8
17CF:01DB 87463E        XCHG    AX,[BP+3E]
17CF:01DE 3CD8          CMP     AL,D8
17CF:01E0 7599          JNZ     017B
17CF:01E2 BE807D        MOV     SI,7D80
-u
17CF:01E5 AC            LODSB
17CF:01E6 98            CBW
17CF:01E7 03F0          ADD     SI,AX
17CF:01E9 AC            LODSB
17CF:01EA 84C0          TEST    AL,AL
17CF:01EC 7417          JZ      0205
17CF:01EE 3CFF          CMP     AL,FF
17CF:01F0 7409          JZ      01FB
17CF:01F2 B40E          MOV     AH,0E
17CF:01F4 BB0700        MOV     BX,0007
17CF:01F7 CD10          INT     10
17CF:01F9 EBEE          JMP     01E9
17CF:01FB BE837D        MOV     SI,7D83
17CF:01FE EBE5          JMP     01E5
17CF:0200 BE817D        MOV     SI,7D81
17CF:0203 EBE0          JMP     01E5
-u
17CF:0205 33C0          XOR     AX,AX
17CF:0207 CD16          INT     16
17CF:0209 5E            POP     SI
17CF:020A 1F            POP     DS
17CF:020B 8F04          POP     [SI]
17CF:020D 8F4402        POP     [SI+02]
17CF:0210 CD19          INT     19
17CF:0212 BE827D        MOV     SI,7D82
17CF:0215 8B7D0F        MOV     DI,[DI+0F]
17CF:0218 83FF02        CMP     DI,+02
17CF:021B 72C8          JB      01E5
17CF:021D 8BC7          MOV     AX,DI
17CF:021F 48            DEC     AX
17CF:0220 48            DEC     AX
17CF:0221 8A4E0D        MOV     CL,[BP+0D]
17CF:0224 F7E1          MUL     CX
-u
17CF:0226 0346FC        ADD     AX,[BP-04]
17CF:0229 1356FE        ADC     DX,[BP-02]
17CF:022C BB0007        MOV     BX,0700
```

```
17CF:022F 53            PUSH    BX
17CF:0230 B104          MOV     CL,04
17CF:0232 E81600        CALL    024B
17CF:0235 5B            POP     BX
17CF:0236 72C8          JB      0200
17CF:0238 813F4D5A      CMP     WORD PTR [BX],5A4D
17CF:023C 75A7          JNZ     01E5
17CF:023E 81BF0002424A  CMP     WORD PTR [BX+0200],4A42
17CF:0244 759F          JNZ     01E5
-u
17CF:0246 EA00027000    JMP     0070:0200
17CF:024B 50            PUSH    AX
17CF:024C 52            PUSH    DX
17CF:024D 51            PUSH    CX
17CF:024E 91            XCHG    CX,AX
17CF:024F 92            XCHG    DX,AX
17CF:0250 33D2          XOR     DX,DX
17CF:0252 F77618        DIV     WORD PTR [BP+18]
17CF:0255 91            XCHG    CX,AX
17CF:0256 F77618        DIV     WORD PTR [BP+18]
17CF:0259 42            INC     DX
17CF:025A 87CA          XCHG    CX,DX
17CF:025C F7761A        DIV     WORD PTR [BP+1A]
17CF:025F 8AF2          MOV     DH,DL
17CF:0261 8A5624        MOV     DL,[BP+24]
17CF:0264 8AE8          MOV     CH,AL
-u
17CF:0266 D0CC          ROR     AH,1
17CF:0268 D0CC          ROR     AH,1
17CF:026A 0ACC          OR      CL,AH
17CF:026C B80102        MOV     AX,0201
17CF:026F CD13          INT     13
17CF:0271 59            POP     CX
17CF:0272 5A            POP     DX
17CF:0273 58            POP     AX
17CF:0274 7209          JB      027F
17CF:0276 40            INC     AX
17CF:0277 7501          JNZ     027A
17CF:0279 42            INC     DX
17CF:027A 035E0B        ADD     BX,[BP+0B]
17CF:027D E2CC          LOOP    024B
17CF:027F C3            RET
17CF:0280 0318          ADD     BX,[BX+SI]
17CF:0282 0127          ADD     [BX],SP
17CF:0284 0D0A49        OR      AX,490A
-u
17CF:0287 6E            DB      6E
17CF:0288 7661          JBE     02EB
17CF:028A 6C            DB      6C
17CF:028B 69            DB      69
17CF:028C 64            DB      64
17CF:028D 207379        AND     [BP+DI+79],DH
17CF:0290 7374          JNB     0306
17CF:0292 65            DB      65
17CF:0293 6D            DB      6D
17CF:0294 206469        AND     [SI+69],AH
17CF:0297 736B          JNB     0304
17CF:0299 FF0D          DEC     WORD PTR [DI]
17CF:029B 0A4469        OR      AL,[SI+69]
17CF:029E 736B          JNB     030B
17CF:02A0 20492F        AND     [BX+DI+2F],CL
17CF:02A3 4F            DEC     DI
```

```
17CF:02A4 206572        AND     [DI+72],AH
-u
17CF:02A7 726F          JB      0318
17CF:02A9 72FF          JB      02AA
17CF:02AB 0D0A52        OR      AX,520A
17CF:02AE 65            DB      65
17CF:02AF 706C          JO      031D
17CF:02B1 61            DB      61
17CF:02B2 63            DB      63
17CF:02B3 65            DB      65
17CF:02B4 207468        AND     [SI+68],DH
17CF:02B7 65            DB      65
17CF:02B8 206469        AND     [SI+69],AH
17CF:02BB 736B          JNB     0328
17CF:02BD 2C20          SUB     AL,20
17CF:02BF 61            DB      61
17CF:02C0 6E            DB      6E
17CF:02C1 64            DB      64
17CF:02C2 207468        AND     [SI+68],DH
17CF:02C5 65            DB      65
17CF:02C6 6E            DB      6E
-u
17CF:02C7 207072        AND     [BX+SI+72],DH
17CF:02CA 65            DB      65
17CF:02CB 7373          JNB     0340
17CF:02CD 20616E        AND     [BX+DI+6E],AH
17CF:02D0 7920          JNS     02F2
17CF:02D2 6B            DB      6B
17CF:02D3 65            DB      65
17CF:02D4 790D          JNS     02E3
17CF:02D6 0A00          OR      AL,[BX+SI]
17CF:02D8 49            DEC     CX
17CF:02D9 4F            DEC     DI
17CF:02DA 2020          AND     [BX+SI],AH
17CF:02DC 2020          AND     [BX+SI],AH
17CF:02DE 2020          AND     [BX+SI],AH
17CF:02E0 53            PUSH    BX
17CF:02E1 59            POP     CX
17CF:02E2 53            PUSH    BX
17CF:02E3 4D            DEC     BP
17CF:02E4 53            PUSH    BX
17CF:02E5 44            INC     SP
17CF:02E6 4F            DEC     DI
-u
17CF:02E7 53            PUSH    BX
17CF:02E8 2020          AND     [BX+SI],AH
17CF:02EA 205359        AND     [BP+DI+59],DL
17CF:02ED 53            PUSH    BX
17CF:02EE 800100        ADD     BYTE PTR [BX+DI],00
17CF:02F1 57            PUSH    DI
17CF:02F2 49            DEC     CX
17CF:02F3 4E            DEC     SI
17CF:02F4 42            INC     DX
17CF:02F5 4F            DEC     DI
17CF:02F6 4F            DEC     DI
17CF:02F7 54            PUSH    SP
17CF:02F8 205359        AND     [BP+DI+59],DL
17CF:02FB 53            PUSH    BX
17CF:02FC 0000          ADD     [BX+SI],AL
17CF:02FE 55            PUSH    BP
17CF:02FF AA            STOSB
```

On peut évidemment s'arrêter à l'adresse `02FFh`, puisque le premier secteur contient 512 octets

(soit `1FFh`, mais n'oublions pas que nous commençons à `100h`). Nous allons ainsi déjà bien au-delà de ce qui est nécessaire puisque les deux derniers octets, par exemple, correspondent à un nombre magique alors qu'ils sont ici désassemblés.

Notre tâche est maintenant de comprendre ce que fait ce code.

1.3.4 Que fait le code ?

Nous allons essayer de comprendre ce que fait le code.

1.3.4.1 Chargement des paramètres du disque système

Visiblement les lignes `140h` à `0179h` permettent de charger les paramètres du disque système comme le montrent les commentaires que nous y ajoutons :

```
17CF:0140 FA            CLI                 ; pas d'interruption masquable
17CF:0141 33C9          XOR     CX,CX       ; cx = 0
17CF:0143 8ED1          MOV     SS,CX       ; ss = 0
17CF:0145 BCFC7B        MOV     SP,7BFC     ; sp = 7BFCh = 7C00h - 3
; donc pile de 3 octets en-dessous du debut du code
17CF:0148 16            PUSH    SS
17CF:0149 07            POP     ES          ; es = ss = 0
17CF:014A BD7800        MOV     BP,0078     ; bp = 4 * 1Eh
; l'interruption 1Eh donne les parametres du disque
17CF:014D C57600        LDS     SI,[BP+00]
; si au debut des parametres du disque
17CF:0150 1E            PUSH    DS          ; sauvegarde des
17CF:0151 56            PUSH    SI          ; registres qui
17CF:0152 16            PUSH    SS          ; seront utilises
17CF:0153 55            PUSH    BP
17CF:0154 BF2205        MOV     DI,0522
; tampon debutant a l'adresse absolue 522h
; pour sauvegarder les parametres du disque
17CF:0157 897E00        MOV     [BP+00],DI
17CF:015A 894E02        MOV     [BP+02],CX
; que font ces deux lignes ?
17CF:015D B10B          MOV     CL,0B       ; 11 octets a copier
17CF:015F FC            CLD                 ; direction positive
17CF:0160 F3            REPZ                ; copie des 11 octets
17CF:0161 A4            MOVSB               ; de si a di
17CF:0162 06            PUSH    ES
17CF:0163 1F            POP     DS          ; ds = es = 0
17CF:0164 BD007C        MOV     BP,7C00     ; bp au debut du
; programme d'amorcage
17CF:0167 C645FE0F      MOV     BYTE PTR [DI-02],0F
; 0Fh en 522 + 11 - 2,
; c'est-a-dire 15 pour le temps de stabilisation de la tete
17CF:016B 8B4618        MOV     AX,[BP+18] ; ax = 12
; bp est place au debut du programme ; les deux octets
; 18h et 19h donnent le nombre de secteurs par piste
17CF:016E 8845F9        MOV     [DI-07],AL
; di = 522 + 11 - 7 d'ou 12 secteurs par piste
17CF:0171 FB            STI
; remise a l'ecoute des interruptions masquables
17CF:0172 386624        CMP     [BP+24],AH
; [BP+24] contient le numero de lecteur, ici 0
; AH contient 0 ici
17CF:0175 7C04          JL      017B
; si moins aller en 017B, donc on n'y va pas
17CF:0177 CD13          INT     13
; repositionnement du disque A
17CF:0179 723C          JB      01B7
; si erreur aller en 01B7
```

1.3.4.2 Calcul de l'emplacement du répertoire racine

Les lignes `17B` à `1AC` permettent de déterminer l'emplacement du répertoire racine :

```
17CF:017B 8A4610        MOV     AL,[BP+10]
; al = 2, nombre de FAT
17CF:017E 98            CBW
; convertit l'octet AL en mot AX
17CF:017F F76616        MUL     WORD PTR [BP+16]
; multiplie par le nombre de secteurs par FAT
17CF:0182 03461C        ADD     AX,[BP+1C]
17CF:0185 13561E        ADC     DX,[BP+1E]
; on a ajoute le nombre de secteurs caches
17CF:0188 03460E        ADD     AX,[BP+0E]
17CF:018B 13D1          ADC     DX,CX
; cx = 0 mais cela permet d'ajouter la retenue eventuelle
; on a ajoute le nombre de secteurs reserves
17CF:018D 50            PUSH    AX
17CF:018E 52            PUSH    DX
; ax et dx sauvegardes sur la pile
17CF:018F 8946FC        MOV     [BP-04],AX
17CF:0192 8956FE        MOV     [BP-02],DX
; donc sauvegardes deux fois
17CF:0195 B82000        MOV     AX,0020
; ax = 32, taille d'une entree du repertoire
17CF:0198 8B7611        MOV     SI,[BP+11]
; si = nombre maximum d'entrees dans le repertoire
17CF:019B F7E6          MUL     SI
; ax:dx contient le nombre d'octets du repertoire racine
17CF:019D 8B5E0B        MOV     BX,[BP+0B]
; bx = nombre d'octets par secteur
17CF:01A0 03C3          ADD     AX,BX
; on ajoute le secteur d'amorcage
17CF:01A2 48            DEC     AX
; ax = ax-1
17CF:01A3 F7F3          DIV     BX
; on divise par le nombre d'octets par secteur
17CF:01A5 0146FC        ADD     [BP-04],AX
17CF:01A8 114EFE        ADC     [BP-02],CX
; on sauvegarde la somme
17CF:01AB 5A            POP     DX
17CF:01AC 58            POP     AX
; on restaure les anciennes valeurs
```

1.3.4.3 Routine de lecture d'un secteur

On lit ensuite le répertoire racine. Pour cela, comme nous allons le voir, on fait appel à un sous-programme commençant à l'adresse `024Bh`. Ce sous-programme s'arrête à l'adresse `027Fh`, première occurrence d'une instruction de retour. Son but est de lire un secteur, son numéro logique étant transmis *via* les registres `ax` et `dx`, le nombre de secteurs à lire via `cl` et l'adresse du tampon pour entreposer via `es:bx` :

```
17CF:024B 50            PUSH    AX
17CF:024C 52            PUSH    DX
17CF:024D 51            PUSH    CX
; sauvegarde sur la pile des registres qui vont etre utilises
17CF:024E 91            XCHG    CX,AX
; on place ax dans cx
17CF:024F 92            XCHG    DX,AX
; on place dx dans ax
; ax contient donc le numero de secteur
; mais division sur ax:dx donc il faut mettre dx a 0
17CF:0250 33D2          XOR     DX,DX                ; dx = 0
17CF:0252 F77618        DIV     WORD PTR [BP+18]
; on divise ax par le nombre de secteurs par piste
17CF:0255 91            XCHG    CX,AX
17CF:0256 F77618        DIV     WORD PTR [BP+18]
; on divise dx par le nombre de secteurs par piste
17CF:0259 42            INC     DX
; on incremente dx de 1 pour tenir compte
17CF:025A 87CA          XCHG    CX,DX
; on met ax dans dx
17CF:025C F7761A        DIV     WORD PTR [BP+1A]
; on divise par le nombre de tetes
17CF:025F 8AF2          MOV     DH,DL
; le numero de la tete dans dh
17CF:0261 8A5624        MOV     DL,[BP+24]
; le numero du lecteur de disquette dans dl
17CF:0264 8AE8          MOV     CH,AL
; le numero de cylindre dans ch
17CF:0266 D0CC          ROR     AH,1
17CF:0268 D0CC          ROR     AH,1
17CF:026A 0ACC          OR      CL,AH
; le numero de secteur dans cl
17CF:026C B80102        MOV     AX,0201
; lecture d'un secteur grace a une interruption logicielle du BIOS
17CF:026F CD13          INT     13
17CF:0271 59            POP     CX
17CF:0272 5A            POP     DX
17CF:0273 58            POP     AX
; on restaure les contenus initiaux des registres
17CF:0274 7209          JB      027F
; en cas de reussite on termine ; sinon
17CF:0276 40            INC     AX
17CF:0277 7501          JNZ     027A
17CF:0279 42            INC     DX
17CF:027A 035E0B        ADD     BX,[BP+0B]
17CF:027D E2CC          LOOP    024B
17CF:027F C3            RET
```

1.3.4.4 Chargement du répertoire racine

Une fois l'emplacement du répertoire racine déterminé, on le charge à l'adresse `700h` :

```
17CF:01AB 5A            POP     DX
17CF:01AC 58            POP     AX
; ax:dx contient le numero logique du secteur
17CF:01AD BB0007        MOV     BX,0700
; le tampon de transfert commence a l'adresse 700h
17CF:01B0 8BFB          MOV     DI,BX
; di pointe sur ce tampon
17CF:01B2 B101          MOV     CL,01
; lecture d'un secteur
; c'est-a-dire placement du contenu du repertoire
; racine a l'adresse 700, grace au sous-programme etudie ci-dessus
17CF:01B4 E89400        CALL    024B
```

1.3.4.5 Fichiers systèmes présents ?

Une fois le répertoire racine chargé, on vérifie que les deux fichiers systèmes, de noms `io.sys` et `msdos.sys`, sont, non seulement présents, mais sont de plus les deux premiers fichiers du répertoire racine.

```
; erreur de chargement  si CF = 1
17CF:01B7 7247          JB      0200
; aller alors au traitement d'erreur a la ligne 200h
17CF:01B9 382D          CMP     [DI],CH
; si le premier caractere est 0, le tampon n'est pas initialise
; donc aller a l'initialisation, a la ligne 1D6
17CF:01BB 7419          JZ      01D6
; sinon 11 caracteres a comparer (8 + 3)
17CF:01BD B10B          MOV     CL,0B
; sauvegarder le contenu de SI sur la pile
17CF:01BF 56            PUSH    SI
17CF:01C0 8B763E        MOV     SI,[BP+3E]
; SI pointe sur une partie reservee
; au debut F1h 7Dh
; apres 1D6 on a la bonne valeur
; pointe sur 'IO      SYS' ?
17CF:01C3 F3              REPZ
17CF:01C4 A6              CMPSB
; on compare les deux noms
17CF:01C5 5E            POP     SI
; s'ils sont egaux passer a la suite en 212h
17CF:01C6 744A          JZ      0212
17CF:01C8 4E            DEC     SI
17CF:01C9 740B          JZ      01D6
17CF:01CB 03F9          ADD     DI,CX
17CF:01CD 83C715        ADD     DI,+15
; on a ajoute 22 a di pour qu'il pointe sur 'msdos   sys'
17CF:01D0 3BFB          CMP     DI,BX
17CF:01D2 72E5          JB      01B9
; si erreur recharger le repertoire racine
17CF:01D4 EBD7          JMP     01AD
```

Le traitement de l'erreur `200h` est :

```
17CF:0200 BE817D        MOV     SI,7D81
17CF:0203 EBE0          JMP     01E5
```

la terminaison du traitement de l'erreur se manifestant par un saut inconditionnel.

L'initialisation 1D6 place l'adresse de 'IO SYS' en [BP+3E] :

```
17CF:01D6 2BC9          SUB     CX,CX       ; cx = 0
17CF:01D8 B8D87D        MOV     AX,7DD8
; il s'agit d'une adresse absolue lorsque le programme
; est charge en 7C00
; donc en 1D8h + 100h = 2D8h ici
; soit le debut de 'IO     SYS'
17CF:01DB 87463E        XCHG    AX,[BP+3E]
; place l'adresse dans une zone reservee,
; celle qui suit le nom du volume
17CF:01DE 3CD8          CMP     AL,D8
; si deuxieme passage alors ax = [BP+3E]
; donc AL = D8h
17CF:01E0 7599          JNZ     017B
; sinon retourner recalculer l'emplacement du repertoire
; racine et revenir en 1B7h apres initialisation
17CF:01E2 BE807D        MOV     SI,7D80
; il s'agit encore d'une adresse absolue
; donc ici en 180h + 100h
; il y a 03h 18h 01h 27h 0Dh 0Ah dans notre cas
17CF:01E5 AC            LODSB
; charge un octet dans AL
17CF:01E6 98            CBW
; le convertit en un mot dans AX
17CF:01E7 03F0          ADD     SI,AX
; si = 180h + 100h + 03h
17CF:01E9 AC            LODSB
; al = 27h dans notre cas
17CF:01EA 84C0          TEST    AL,AL
17CF:01EC 7417          JZ      0205
; si nul redemarrer (voir ci-dessous)
17CF:01EE 3CFF          CMP     AL,FF
; si Al = FBh aller en 1FB
17CF:01F0 7409          JZ      01FB
; sinon
17CF:01F2 B40E          MOV     AH,0E
; ecrire en teletype
17CF:01F4 BB0700        MOV     BX,0007
; sur la page 0 en couleur 07
17CF:01F7 CD10          INT     10
; ce qui est dans AL, soit ' dans notre cas
17CF:01F9 EBEE          JMP     01E9
; on boucle en retournant en 1E9
17CF:01FB BE837D        MOV     SI,7D83
; il s'agit d'une adresse absolue donc pour nous
; 100h + 183h = 283h
17CF:01FE EBE5          JMP     01E5
```

La sous-routine de redémarrage 205h est :

```
17CF:0205 33C0          XOR     AX,AX       ; ax = 0
17CF:0207 CD16          INT     16
; attend un caractere au clavier
17CF:0209 5E            POP     SI
17CF:020A 1F            POP     DS
17CF:020B 8F04          POP     [SI]
17CF:020D 8F4402        POP     [SI+02]
; restaure les valeurs
17CF:0210 CD19          INT     19
; et redemarre
```

1.3.4.6 Chargement des quatre premiers secteurs de `io.sys`

Après s'être assurés que les fichiers systèmes sont présents, on charge les quatre premiers secteurs de `io.sys` :

```
17CF:0212 BE827D        MOV     SI,7D82
; adresse absolue donc pour nous
; 100h + 182h = 282h
17CF:0215 8B7D0F        MOV     DI,[DI+0F]
; rappelons que DI pointe sur l'adresse 700h
; lors du chargement du repertoire racine
; et qu'on a compare onze octets donc pointe maintenant sur 70Bh
; en ajoutant 0Fh, SI pointe sur 71Ah,
; c'est-a-dire l'unite d'allocation de depart du fichier
; io.sys
17CF:0218 83FF02        CMP     DI,+02
; s'il s'agit du premier fichier on doit avoir 2
17CF:021B 72C8          JB      01E5
; si ce n'est pas le cas retourner en 1E5
17CF:021D 8BC7          MOV     AX,DI
; ax contient le numero d'unite d'allocation relatif
17CF:021F 48            DEC     AX
17CF:0220 48            DEC     AX
; on diminue de 2 car on part de 2
17CF:0221 8A4E0D        MOV     CL,[BP+0D]
; cl = nombre de secteurs par unite d'allocation
17CF:0224 F7E1          MUL     CX
; on obtient le numero de secteur relatif
17CF:0226 0346FC        ADD     AX,[BP-04]
17CF:0229 1356FE        ADC     DX,[BP-02]
; on y avait place le nombre de secteurs
; jusqu'au repertoire inclus
17CF:022C BB0007        MOV     BX,0700
; tampon pour lire les 4 secteurs
17CF:022F 53            PUSH    BX
17CF:0230 B104          MOV     CL,04
; lire quatre secteurs
17CF:0232 E81600        CALL    024B
17CF:0235 5B            POP     BX
17CF:0236 72C8          JB      0200
; en cas d'erreur de lecture aller au traitement d'erreur 200
17CF:0238 813F4D5A      CMP     WORD PTR [BX],5A4D
17CF:023C 75A7          JNZ     01E5
17CF:023E 81BF0002424A  CMP     WORD PTR [BX+0200],4A42
17CF:0244 759F          JNZ     01E5
; nombres magiques presents ?
17CF:0246 EA00027000    JMP     0070:0200
; adresse absolue
; demarrer le programme lu sur ces secteurs
```

Le sous-programme commençant en `24Bh` permet de lire les quatre secteurs demandés :

```
17CF:024B 50            PUSH    AX
17CF:024C 52            PUSH    DX
17CF:024D 51            PUSH    CX
17CF:024D 51            PUSH    CX
; on sauvegarde sur la pile les registres que l'on va utiliser
17CF:024E 91            XCHG    CX,AX
17CF:024F 92            XCHG    DX,AX
17CF:0250 33D2          XOR     DX,DX               ; dx = 0
17CF:0252 F77618        DIV     WORD PTR [BP+18]
; on divise par le nombre de secteurs par piste
17CF:0255 91            XCHG    CX,AX
17CF:0256 F77618        DIV     WORD PTR [BP+18]
17CF:0259 42            INC     DX
17CF:025A 87CA          XCHG    CX,DX
17CF:025C F7761A        DIV     WORD PTR [BP+1A]
; on divise par le nombre de tetes
17CF:025F 8AF2          MOV     DH,DL
17CF:0261 8A5624        MOV     DL,[BP+24]
; le numero de lecteur de disquette dans dl
17CF:0264 8AE8          MOV     CH,AL
; le numero de cylindre dans ch
17CF:0266 D0CC          ROR     AH,1
17CF:0268 D0CC          ROR     AH,1
17CF:026A 0ACC          OR      CL,AH
; le numero de secteur dans cl
17CF:026C B80102        MOV     AX,0201
17CF:026F CD13          INT     13
; lecture d'1 secteur
17CF:0271 59            POP     CX
17CF:0272 5A            POP     DX
17CF:0273 58            POP     AX
17CF:0274 7209          JB      027F
17CF:0276 40            INC     AX
17CF:0277 7501          JNZ     027A
17CF:0279 42            INC     DX
17CF:027A 035E0B        ADD     BX,[BP+0B]
17CF:027D E2CC          LOOP    024B
; recommencer
17CF:027F C3            RET
```

On arrive ainsi à la fin du code comme le montre deux indices : les six octets suivants ont été utilisés (il s'agit de repères) et les octets suivants sont visiblement des données ; on a été renvoyé à poursuivre le programme chargé.

1.4 Cas du secteur d'amorçage de

1.4.1 Contenu

En demandant le déassemblage à partir de 100, nous trouvons :

```
C:\>debug mbr.bin
-u 100
17CF:0100 33C0          XOR     AX,AX
17CF:0102 8ED0          MOV     SS,AX
17CF:0104 BC007C        MOV     SP,7C00
17CF:0107 FB            STI
17CF:0108 50            PUSH    AX
17CF:0109 07            POP     ES
17CF:010A 50            PUSH    AX
17CF:010B 1F            POP     DS
17CF:010C FC            CLD
17CF:010D BE1B7C        MOV     SI,7C1B
17CF:0110 BF1B06        MOV     DI,061B
17CF:0113 50            PUSH    AX
17CF:0114 57            PUSH    DI
17CF:0115 B9E501        MOV     CX,01E5
17CF:0118 F3            REPZ
17CF:0119 A4            MOVSB
17CF:011A CB            RETF
17CF:011B B408          MOV     AH,08
17CF:011D B280          MOV     DL,80
17CF:011F CD13          INT     13
-u
17CF:0121 7240          JB      0163
17CF:0123 81E13F00      AND     CX,003F
17CF:0127 89CE          MOV     SI,CX
17CF:0129 30D2          XOR     DL,DL
17CF:012B 86F2          XCHG    DH,DL
17CF:012D FEC2          INC     DL
17CF:012F 89D7          MOV     DI,DX
17CF:0131 88C8          MOV     AL,CL
17CF:0133 F6E2          MUL     DL
17CF:0135 89C3          MOV     BX,AX
17CF:0137 A1C607        MOV     AX,[07C6]
17CF:013A 8B16C807      MOV     DX,[07C8]
17CF:013E F7F3          DIV     BX
17CF:0140 88C5          MOV     CH,AL
-u
17CF:0142 89D0          MOV     AX,DX
17CF:0144 F6F1          DIV     CL
17CF:0146 88C6          MOV     DH,AL
17CF:0148 88E1          MOV     CL,AH
17CF:014A FEC1          INC     CL
17CF:014C B80102        MOV     AX,0201
17CF:014F BB007C        MOV     BX,7C00
17CF:0152 B280          MOV     DL,80
17CF:0154 CD13          INT     13
17CF:0156 720B          JB      0163
17CF:0158 897718        MOV     [BX+18],SI
17CF:015B 897F1A        MOV     [BX+1A],DI
17CF:015E B80103        MOV     AX,0301
17CF:0161 CD13          INT     13
-u
17CF:0163 B80102        MOV     AX,0201
17CF:0166 BB007C        MOV     BX,7C00
17CF:0169 B90200        MOV     CX,0002
```

```
17CF:016C BA8000        MOV     DX,0080
17CF:016F CD13          INT     13
17CF:0171 90            NOP
17CF:0172 90            NOP
17CF:0173 90            NOP
17CF:0174 90            NOP
17CF:0175 90            NOP
17CF:0176 90            NOP
17CF:0177 90            NOP
17CF:0178 90            NOP
17CF:0179 1E            PUSH    DS
17CF:017A 53            PUSH    BX
17CF:017B CB            RETF
[que des 00 ensuite jusqu'a 0280]
-u
17CF:0280 AC            LODSB
17CF:0281 B40E          MOV     AH,0E
17CF:0283 8A3E6204      MOV     BH,[0462]
-u
17CF:0287 B307          MOV     BL,07
17CF:0289 CD10          INT     10
17CF:028B 3C0A          CMP     AL,0A
17CF:028D 75F1          JNZ     0280
17CF:028F CD18          INT     18
17CF:0291 F4            HLT
17CF:0292 EBFD          JMP     0291
[Que des zeros ensuite jusqu'a 2B8]
-u
17CF:02B8 4E            DEC     SI
17CF:02B9 EF            OUT     DX,AX
17CF:02BA 150000        ADC     AX,0000
17CF:02BD 00800101      ADD     [BX+SI+0101],AL
17CF:02C1 0006FE3F      ADD     [3FFE],AL
17CF:02C5 783F          JS      0306
17CF:02C7 0000          ADD     [BX+SI],AL
-u
17CF:02C9 00C1          ADD     CL,AL
17CF:02CB DF1D                  FISTP   WORD PTR [DI]
[Que des zeros ensuite jusqu'a 2FD]
-u
17CF:02FD 0055AA        ADD     [DI-56],DL
```

On peut évidemment s'arrêter à l'adresse `02FFh`, puisque le premier secteur contient 512 octets (soit `1FFh`, mais n'oublions pas que nous commençons à `100h`). Nous allons ainsi déjà bien au-delà de ce qui est nécessaire puisque les deux derniers octets, par exemple, correspondent à un nombre magique alors qu'ils sont ici désassemblés.

1.4.2 Commentaires

```
C:\>debug mbr.bin
-u 100
17CF:0100 33C0          XOR     AX,AX           ; ax = 0
17CF:0102 8ED0          MOV     SS,AX           ; ss = 0
17CF:0104 BC007C        MOV     SP,7C00         ; sp = 7C00
17CF:0107 FB            STI                     ; on se met a l'ecoute des interruptions masquables
17CF:0108 50            PUSH    AX
17CF:0109 07            POP     ES              : es = 0
17CF:010A 50            PUSH    AX
17CF:010B 1F            POP     DS              : ds = 0
17CF:010C FC            CLD                     ; on inhibe les interruptions masquables
17CF:010D BE1B7C        MOV     SI,7C1B         ; si = 7C1B, soit 27 octets apres le debut
17CF:0110 BF1B06        MOV     DI,061B         ; di = 061B, soit 1563
; ou 539 octets apres la table des vecteurs d'interruption
17CF:0113 50            PUSH    AX              ; on sauvegarde ax
17CF:0114 57            PUSH    DI              ; et di sur la pile
17CF:0115 B9E501        MOV     CX,01E5         ; cx = 01E5
17CF:0118 F3            REPZ                    ; on deplace 1E5, soit 485,
17CF:0119 A4            MOVSB                   ; octets
17CF:011A CB            RETF                    ; revient a un saut en ax:di, les deux derniers
                                                ; sauvegardes donc apres ce qui vient d'etre copie,
                                                ; en 2049
17CF:011B B408          MOV     AH,08           ; fonction 08 de lecture des parametres
                                                ; de formatage du disque
17CF:011D B280          MOV     DL,80           ; dl = 80 pour premier disque dur
17CF:011F CD13          INT     13              ; interruption 13h de gestionnaire du disque
-u
17CF:0121 7240          JB      0163            ; si inferieur
17CF:0123 81E13F00      AND     CX,003F         ; masquer cx avec 3F
17CF:0127 89CE          MOV     SI,CX           ; si = cx
17CF:0129 30D2          XOR     DL,DL           ; dl = 0
17CF:012B 86F2          XCHG    DH,DL           ; dh = 0
17CF:012D FEC2          INC     DL              ; dl = 1
17CF:012F 89D7          MOV     DI,DX           ; di = 1
17CF:0131 88C8          MOV     AL,CL           ; al = cl
17CF:0133 F6E2          MUL     DL              ; ax = al x dl
17CF:0135 89C3          MOV     BX,AX           ; bx = ax
17CF:0137 A1C607        MOV     AX,[07C6]       ; ax = 07C6
17CF:013A 8B16C807      MOV     DX,[07C8]       ; dx = 07C8
17CF:013E F7F3          DIV     BX              ; ax = (dx:ax)/bx
17CF:0140 88C5          MOV     CH,AL           : ch = al numero de cylindre sur lequel lire
-u
17CF:0142 89D0          MOV     AX,DX           ; ax = dx
17CF:0144 F6F1          DIV     CL              ; al = ax/cl
17CF:0146 88C6          MOV     DH,AL           ; dh = al numero de tete de lecture
17CF:0148 88E1          MOV     CL,AH           ; cl = ah
17CF:014A FEC1          INC     CL              ; cl++ numero de secteur a lire
17CF:014C B80102        MOV     AX,0201         ; ah = 02 pour lecteur de secteurs
                                                ; al = 01 pour 1 secteur
17CF:014F BB007C        MOV     BX,7C00         : bx = 7C00
17CF:0152 B280          MOV     DL,80           ; premier disque dur
17CF:0154 CD13          INT     13              ; lecture d'un secteur
17CF:0156 720B          JB      0163            ; si inferieur
17CF:0158 897718        MOV     [BX+18],SI      ; sauvegarde de SI
17CF:015B 897F1A        MOV     [BX+1A],DI      ; sauvegarde de DI
17CF:015E B80103        MOV     AX,0301         ; ah = 03 pour ecriture de secteurs
                                                ; al = 01 pour 1 secteur
17CF:0161 CD13          INT     13              ; ecriture d'un secteur
-u
17CF:0163 B80102        MOV     AX,0201         ; ah = 02 pour lecture de secteurs
                                                ; al = 01 pour 1 secteur
```

```
17CF:0166 BB007C        MOV     BX,7C00         ; bx = 7C00
17CF:0169 B90200        MOV     CX,0002         ; ch = 00 pour numero du cylindre
                                                ; cl = 02 numero du secteur a lire
17CF:016C BA8000        MOV     DX,0080         ; dh = 00 numero de tete de lecture
                                                ; dl = 80 pour premier disque dur
17CF:016F CD13          INT     13              ; lecture d'un secteur
17CF:0171 90            NOP
17CF:0172 90            NOP
17CF:0173 90            NOP
17CF:0174 90            NOP
17CF:0175 90            NOP
17CF:0176 90            NOP
17CF:0177 90            NOP
17CF:0178 90            NOP
17CF:0179 1E            PUSH    DS              ; aller
17CF:017A 53            PUSH    BX              ; a l'adresse
17CF:017B CB            RETF                    ; DS:BX
[que des 00 ensuite jusqu'a 0280]
-u
17CF:0280 AC            LODSB                   ; octet suivant dans al, caractere a afficher
17CF:0281 B40E          MOV     AH,0E           ; ah = 0Eh ecriture d'un caractere en mode teletype
17CF:0283 8A3E6204      MOV     BH,[0462]       ; numero de page
-u
17CF:0287 B307          MOV     BL,07           ; bl = 07 couleur d'ecriture
17CF:0289 CD10          INT     10              ; affichage
17CF:028B 3C0A          CMP     AL,0A           ; si pas fin de ligne
17CF:028D 75F1          JNZ     0280            ; passer au caractere suivant
17CF:028F CD18          INT     18              ; sinon passer au BASIC s'il est installe
17CF:0291 F4            HLT                     ; arreter le systeme
17CF:0292 EBFD          JMP     0291            ; on ne sait jamais
[Que des zeros ensuite jusqu'a 2B8]
-u
17CF:02B8 4E            DEC     SI
17CF:02B9 EF            OUT     DX,AX
17CF:02BA 150000        ADC     AX,0000
17CF:02BD 00800101      ADD     [BX+SI+0101],AL
17CF:02C1 0006FE3F      ADD     [3FFE],AL
17CF:02C5 783F          JS      0306
17CF:02C7 0000          ADD     [BX+SI],AL
-u
17CF:02C9 00C1          ADD     CL,AL
17CF:02CB DF1D                  FISTP   WORD PTR [DI]
[Que des zeros ensuite jusqu'a 2FD]
-u
17CF:02FD 0055AA        ADD     [DI-56],DL
```

1.5 Encore un autre secteur d'amorçage

Prenons notre clé USB.

1.5.1 Récupération du secteur d'amorçage

Faisons démarrer notre clé USB contenant MD-DOS 5.0. Utilisons `debug` pour récupérer le secteur d'amorçage :

```
C:\> debug
- L 100 2 0 1
-N secteura.bin
-rcx
CX 0000
:200
-W
Writing 00200 bytes
-q

C:\>
```

Le fait de conserver le secteur d'amorçage dans un fichier va nous permettre de travailler avec. Nous pouvons maintenant faire afficher le contenu du secteur avec `debug` :

```
C:\>debug secteura.bin
-d 100
17CF:0100  EB 3C 90 4D 53 44 4F 53-35 2E 30 00 02 20 02 00   .<.MSDOS5.0.. ..
17CF:0110  02 00 02 00 00 F8 EF 00-3F 00 FF 00 3F 00 00 00   ........?...?...
17CF:0120  C1 DF 1D 00 80 00 29 29-7D D1 6E 4E 4F 20 4E 41   ......))}.nNO NA
17CF:0130  4D 45 20 20 20 20 46 41-54 31 36 20 20 20 FA 33   ME    FAT16   .3
17CF:0140  C0 8E D0 BC 00 7C 16 07-BB 78 00 36 C5 37 1E 56   .....|...x.6.7.V
17CF:0150  16 53 BF 3E 7C B9 0B 00-FC F3 A4 06 1F C6 45 FE   .S.>|.........E.
17CF:0160  0F 8B 0E 18 7C 88 4D F9-89 47 02 C7 07 3E 7C FB   ....|.M..G...>|.
17CF:0170  CD 13 72 79 33 C0 39 06-13 7C 74 08 8B 0E 13 7C   ..ry3.9..|t....|
-d a continuer
17CF:0180  66 16 03 46 1C 13 56 1E-03 46 0E 13 D1 50 52 89   f..F..V..F...PR.
17CF:0190  46 FC 89 56 FE B8 20 00-8B 76 11 F7 E6 8B 5E 0B   F..V.. ..v....^.
17CF:01A0  03 C3 48 F7 F3 01 46 FC-11 4E FE 5A 58 BB 00 07   ..H...F..N.ZX...
17CF:01B0  8B FB B1 01 E8 94 00 72-47 38 2D 74 19 B1 0B 56   .......rG8-t...V
17CF:01C0  8B 76 3E F3 A6 5E 74 4A-4E 74 0B 03 F9 83 C7 15   .v>..^tJNt......
17CF:01D0  3B FB 72 E5 EB D7 2B C9-B8 D8 7D 87 46 3E 3C D8   ;.r...+...}.F><.
17CF:01E0  75 99 BE 80 7D AC 98 03-F0 AC 84 C0 74 17 3C FF   u...}.......t.<.
17CF:01F0  74 09 B4 0E BB 07 00 CD-10 EB EE BE 83 7D EB E5   t............}..
-d
17CF:0200  BE 81 7D EB E0 33 C0 CD-16 5E 1F 8F 04 8F 44 02   ..}..3...^....D.
17CF:0210  CD 19 BE 82 7D 8B 7D 0F-83 FF 02 72 C8 8B C7 48   ....}.}....r...H
17CF:0220  48 8A 4E 0D F7 E1 03 46-FC 13 56 FE BB 00 07 53   H.N....F..V....S
17CF:0230  B1 04 E8 16 00 5B 72 C8-81 3F 4D 5A 75 A7 81 BF   .....[r..?MZu...
17CF:0240  00 02 42 4A 75 9F EA 00-02 70 00 50 52 51 91 92   ..BJu....p.PRQ..
17CF:0250  33 D2 F7 76 18 91 F7 76-18 42 87 CA F7 76 1A 8A   3..v...v.B...v..
17CF:0260  F2 8A 56 24 8A E8 D0 CC-D0 CC 0A CC B8 01 02 CD   ..V$............
17CF:0270  13 59 5A 58 72 09 40 75-01 42 03 5E 0B E2 CC C3   .YZXr.@u.B.^....
-d
17CF:0280  03 18 01 27 0D 0A 49 6E-76 61 6C 69 64 20 73 79   ...'..Invalid sy
17CF:0290  73 74 65 6D 20 64 69 73-6B FF 0D 0A 44 69 73 6B   stem disk...Disk
17CF:02A0  20 49 2F 4F 20 65 72 72-6F 72 FF 0D 0A 52 65 70    I/O error...Rep
17CF:02B0  6C 61 63 65 20 74 68 65-20 64 69 73 6B 2C 20 61   lace the disk, a
17CF:02C0  6E 64 20 74 68 65 6E 20-70 72 65 73 73 20 61 6E   nd then press an
17CF:02D0  79 20 6B 65 79 0D 0A 00-49 4F 20 20 20 20 20 20   y key...IO
17CF:02E0  53 59 53 4D 53 44 4F 53-20 20 20 53 59 53 80 01   SYSMSDOS   SYS..
17CF:02F0  00 57 49 4E 42 4F 4F 54-20 53 59 53 00 00 55 AA   .WINBOOT SYS..U.
```

Remarquons que le programme commence par un saut inconditionnel et se termine par la signature `55 AAh`.

1.5.2 Données du secteur d'amorçage

Commençons par décrypter les données du secteur d'amorçage ne constituant pas le programme lui-même. Il faut évidemment connaître la structure du secteur d'amorçage de MS-DOS pour cela.

— Commençons par la première ligne. Le premier octet est `EB`, le code opération du saut inconditionnel court. Le second octet, `3C`, est la valeur du déplacement permettant de trouver le début du programme d'amorçage, donc à l'adresse `13Eh`. Ce programme d'amorçage commence par `FA`, comme on peut le constater trois lignes plus bas. Le troisième octet est le code de la non opération NOP, puisqu'il s'agit d'un saut court.
— Les cinq octets suivant spécifient le nom du fabricant, ici « MSDOS » et les trois suivants le numéro de version, ici « 5.0 ».
— Les deux octets suivants, `00h` et `02h`, spécifient le nombre d'octets par secteur, soit `0200h` ou 512.
— L'octet suivant, `20h`, spécifie le nombre de secteurs par unité d'allocation ; ici une unité d'allocation contient 32 secteurs.
— Les deux octets suivants, `02h` et `00h`, spécifient le nombre de secteurs réservés, ici deux.
— Le premier octet de la deuxième ligne, `02`, spécifie le nombre de FAT, ici 2.
— Les deux octets suivants, `00h` et `02h`, spécifient le nombre maximum d'entrées dans le répertoire racine, ici 512.
— Les deux octets suivants, `00h` et `00h`, spécifient le nombre total de secteurs sur le disque.
— L'octet suivant, `F8h`, spécifie le type de disquette, ici un disque dur.
— Les deux octets suivants, `EFh` et `00h`, spécifient le nombre de secteurs par FAT, ici 239.
— Les deux octets suivants, `3Fh` et `00h`, spécifient le nombre de secteurs par piste, soit ici 63.
— Les deux octets suivants, `FFh` et `00h`, spécifient le nombre de têtes de lecture-écriture, soit ici 255.
— Les quatre derniers octets de la deuxième ligne, ici `3Fh`, `00h`, `00h` et `00h`, spécifient le nombre de secteurs cachés.
— Les quatre premiers octets de la troisième ligne, `C1h`, `DFh`, `1Dh` et `00h`, spécifient le nombre total d'octets pour ce disque qui est de capacité supérieure à 32 MiO.
— L'octet suivant, `80h`, spécifie le numéro du lecteur de disquettes ; il s'agit donc ici du premier disque dur C.
— L'octet suivant est réservé par le système et est donc nul.
— L'octet suivant, `29h`, est la signature du secteur d'amorçage étendu.
— Les quatre octets suivants, `29h`, `7Dh`, `D1h` et `6Eh`, spécifient le numéro d'identification du disque.
— Les dix octets suivants spécifient le nom du volume ; il n'y en a pas ici, mais on a « nNO NAME » et en plus on utilise la fin et les octets réservés suivants pour rappeler qu'on a une FAT seize bits.

1.5.3 Détermination du programme d'amorçage

Si on se sert de `debug` de façon abrupte pour désassembler le programme d'assemblage, on n'obtient rien, puisque `debug` commence par essayer de désassembler les données commentées ci-dessus. Il n'interprète pas correctement le code qui commence à l'adresse `13Eh`. En fait seules les deux premières lignes sont bien analysées :

```
C:\>debug secteura.bin
-u
17CF:0100 EB3C          JMP     013E
17CF:0102 90            NOP
17CF:0103 4D            DEC     BP
```

En demandant le désassemblage à partir de `13E`, nous trouvons du code plus compréhensible :

```
C:\>debug secteura.bin
-u 13E
17CF:013E FA            CLI
17CF:013F 33C0          XOR     AX,AX
17CF:0141 8ED0          MOV     SS,AX
17CF:0143 BC007C        MOV     SP,7C00
17CF:0146 16            PUSH    SS
17CF:0147 07            POP     ES
17CF:0148 BB7800        MOV     BX,0078
17CF:014B 36            SS:
17CF:014C C537          LDS     SI,[BX]
17CF:014E 1E            PUSH    DS
17CF:014F 56            PUSH    SI
17CF:0150 16            PUSH    SS
17CF:0151 53            PUSH    BX
17CF:0152 BF3E7C        MOV     DI,7C3E
17CF:0155 B90B00        MOV     CX,000B
17CF:0158 FC            CLD
17CF:0159 F3            REPZ
17CF:015A A4            MOVSB
17CF:015B 06            PUSH    ES
17CF:015C 1F            POP     DS
17CF:015D C645FE0F      MOV     BYTE PTR [DI-02],0F
-u
17CF:0161 8B0E187C      MOV     CX,[7C18]
17CF:0165 884DF9        MOV     [DI-07],CL
17CF:0168 894702        MOV     [BX+02],AX
17CF:016B C7073E7C      MOV     WORD PTR [BX],7C3E
17CF:016F FB            STI
17CF:0170 CD13          INT     13
17CF:0172 7279          JB      01ED
17CF:0174 33C0          XOR     AX,AX
17CF:0176 3906137C      CMP     [7C13],AX
17CF:017A 7408          JZ      0184
17CF:017C 8B0E137C      MOV     CX,[7C13]
17CF:0180 890E207C      MOV     [7C20],CX
-u
17CF:0184 A0107C        MOV     AL,[7C10]
17CF:0187 F726167C      MUL     WORD PTR [7C16]
17CF:018B 03061C7C      ADD     AX,[7C1C]
17CF:018F 13161E7C      ADC     DX,[7C1E]
17CF:0193 03060E7C      ADD     AX,[7C0E]
17CF:0197 83D200        ADC     DX,+00
17CF:019A A3507C        MOV     [7C50],AX
17CF:019D 8916527C      MOV     [7C52],DX
17CF:01A1 A3497C        MOV     [7C49],AX
-u
```

```
17CF:01A4 89164B7C      MOV     [7C4B],DX
17CF:01A8 B82000        MOV     AX,0020
17CF:01AB F726117C      MUL     WORD PTR [7C16]
17CF:01AF 8B1E0B7C      MOV     BX,[7C0B]
17CF:01B3 03C3          ADD     AX,BX
17CF:01B5 48            DEC     AX
17CF:01B6 F7F3          DIV     BX
17CF:01B8 0106497C      ADD     [7C49],AX
17CF:01BC 83164B7C00    ADC     WORD PTR [7C4B],+00
17CF:01C1 BB0005        MOV     BX,0500
-u
17CF:01C4 8B16527C      MOV     DX,[7C52]
17CF:01C8 A1507C        MOV     AX,[7C50]
17CF:01CB E89200        CALL    0260
17CF:01CE 721D          JB      01ED
17CF:01D0 B001          MOV     AL,01
17CF:01D2 E8AC00        CALL    0281
17CF:01D5 7216          JB      01ED
17CF:01D7 8BFB          MOV     DI,BX
17CF:01D9 B90B00        MOV     CX,000B
17CF:01DC BEE67D        MOV     SI,7DE6
17CF:01DF F3            REPZ
17CF:01E0 A6            CMPSB
17CF:01E1 750A          JNZ     01ED
17CF:01E3 8D7F20        LEA     DI,[BX+20]
-u
17CF:01E6 B90B00        MOV     CX,000B
17CF:01E9 F3            REPZ
17CF:01EA A6            CMPSB
17CF:01EB 74E18         JZ      0205
17CF:01ED BE9E7D        MOV     SI,7D9E
17CF:01F0 E85F00        CALL    0252
17CF:01F3 33C0          XOR     AX,AX
17CF:01F5 CD16          INT     16
17CF:01F7 5E            POP     SI
17CF:01F8 1F            POP     DS
17CF:01F9 8F04          POP     [SI]
17CF:01FB 8F4402        POP     [SI+02]
17CF:01FE CD19          INT     19
17CF:0200 58            POP     AX
17CF:0201 58            POP     AX
17CF:0202 58            POP     AX
17CF:0203 EBE8          JMP     01ED
17CF:0205 8B471A        MOV     AX,[BX+1A]
-u
17CF:0208 48            DEC     AX
17CF:0209 48            DEC     AX
17CF:020A 8A1E0D7C      MOV     BL,[7C0D]
17CF:020E 32FF          XOR     BH,BH
17CF:0210 F7E3          MUL     BX
17CF:0212 0306497C      ADD     AX,[7C49]
17CF:0216 13164B7C      ADC     DX,[7C4B]
17CF:021A BB0007        MOV     BX,0700
17CF:021D B90300        MOV     CX,0003
17CF:0220 50            PUSH    AX
17CF:0221 52            PUSH    DX
17CF:0222 51            PUSH    CX
17CF:0223 E83A00        CALL    0260
17CF:0226 72D8          JB      0200
-u
17CF:0228 B001          MOV     AL,01
17CF:022A E85400        CALL    0281
```

```
17CF:022D 59            POP     CX
17CF:022E 5A            POP     DX
17CF:022F 58            POP     AX
17CF:0230 72BB          JB      01ED
17CF:0232 050100        ADD     AX,0001
17CF:0235 83D200        ADC     DX,+00
17CF:0238 031E0B7C      ADD     BX,[7C0B]
17CF:023C E2E2          LOOP    0220
17CF:023E 8A2E157C      MOV     CH,[7C15]
17CF:0242 8A16247C      MOV     DL,[7C24]
17CF:0246 8B1E497C      MOV     BX,[7C49]
-u
17CF:024A A14B7C        MOV     AX,[7C4B]
17CF:024D EA00007000    JMP     0070:0000
17CF:0252 AC            LODSB
17CF:0253 0AC0          OR      AL,AL
17CF:0255 7429          JZ      0200
17CF:0257 B40E          MOV     AH,0E
17CF:0259 BB0700        MOV     BX,0007
17CF:025C CD10          INT     10
17CF:025E EBF2          JMP     0252
17CF:0260 3B16187C      CMP     DX,[7C18]
17CF:0264 7319          JNB     027F
17CF:0266 F736187C      DIV     WORD PTR [7C18]
-u
17CF:026A FEC2          INC     DL
17CF:026C 88164F7C      MOV     [7C4F],DL
17CF:0270 33D2          XOR     DX,DX
17CF:0272 F7361A7C      DIV     WORD PTR [7C1A]
17CF:0276 8816257C      MOV     [7C25],DL
17CF:027A A34D7C        MOV     [7C4D],AX
17CF:027D F8            CLC
17CF:027E C3            RET
17CF:027F F9            STC
17CF:0280 C3            RET
17CF:0281 B402          MOV     AH,02
17CF:0283 8B164D7C      MOV     DX,[7C4D]
17CF:0287 B106          MOV     CL,06
17CF:0289 D2E6          SHL     DH,CL
-u
17CF:028B 0A364F7C      OR      DH,[7C4F]
17CF:028F 8BCA          MOV     CX,DX
17CF:0291 86E9          XCHG    CH,CL
17CF:0293 8A16247C      MOV     DL,[7C24]
17CF:0297 8A36257C      MOV     DH,[7C25]
17CF:029B CD13          INT     13
17CF:029D C3            RET
```

Nous verrons qu'on peut s'arrêter à l'adresse `029Dh`, car le reste est constitué de constantes chaînes de caractères et, bien sûr, de la signature.

Notre tâche est maintenant de comprendre ce que fait ce code.

1.5.4 Que fait le code ?

Nous allons essayer de comprendre ce que fait le code. Rappelons que le chargeur du BIOS a chargé le secteur d'amorçage à l'adresse 7C00h.

1.5.4.1 Mise en place des paramètres du disque système

Les lignes `13Eh` à `0172h` mettent en place les paramètres du disque système pour sa réinitialisation, comme le montrent les commentaires que nous y ajoutons :

```
17CF:013E FA            CLI                 ; inhibition des interruptions masquables
17CF:013F 33C0          XOR     AX,AX       ; ax = 0
17CF:0141 8ED0          MOV     SS,AX       ; ss = 0
17CF:0143 BC007C        MOV     SP,7C00     ; sp = 7C00h
; donc sommet de la pile au debut du code charge
; ce qui n'est pas grave car le saut court le constituant ne sera plus utilise
17CF:0146 16            PUSH    SS
17CF:0147 07            POP     ES          ; es = ss = 0
; segment supplementaire en debut de memoire
17CF:0148 BB7800        MOV     BX,0078     ; bx = 4 * 1Eh
; bx point sur le vecteur de l'interruption 1Eh
; en fait pointeur vers la table des parametres d'une unite de disquette
17CF:014B C537          LDS     SI,[BX]
; si pointe sur la table des parametres du disque
17CF:014E 1E            PUSH    DS          ; sauvegarde des
17CF:014F 56            PUSH    SI          ; registres qui
17CF:0150 16            PUSH    SS          ; seront utilises
17CF:0151 53            PUSH    BX          ; utilises
17CF:0152 BF3E7C        MOV     DI,7C3E
; di pointe a l'adresse absolue 7C3Eh = 7C00h + 3Eh
; c'est-a-dire juste apres les donnees du secteur d'amorcage
17CF:0155 B90B00        MOV     CX,000B     ; 11 octets a copier
17CF:0158 FC            CLD                 ; direction negative (positive)
17CF:0159 F3            REPZ                ; copie des 11 octets
17CF:015A A4            MOVSB               ; de si a di
; deplacement du nom du volume
17CF:015B 06            PUSH    ES
17CF:015C 1F            POP     DS          ; ds = es = 0
; le segment des donnees aura des adresses absolues
17CF:015D C645FE0F      MOV     BYTE PTR [DI-02],0F
; 0Fh en 7C3E - 11 - 2,
; on initialise a 15 le temps de stabilisation de la tete
17CF:0161 8B0E187C      MOV     CX,[7C18]
; on place dans cx les octets 18h et 19h depuis le debut du secteur d'amorcage
; donc le nombre de secteurs par piste sur le disque
17CF:0165 884DF9        MOV     [DI-07],CL
; que l'on reporte dans 7C3Eh - 11 - 7
17CF:0168 894702        MOV     [BX+02],AX
; on place
17CF:016B C7073E7C      MOV     WORD PTR [BX],7C3E
; bx = 7C3E = 7C00 + 3E soit 62 octets apres le debut du programme
; donc pointe sur FA 33, la premiere instruction
17CF:016F FB            STI
; remise a l'ecoute des interruptions masquables
17CF:0170 CD13          INT     13
; et ah = 00h donc initialisation du disque
17CF:0172 7279          JB      01ED
; si erreur aller a la routine de traitement des erreurs, dont on ne revient pas
```

1.5.4.2 Routine de traitement des erreurs

La routine.- Nous venons de voir que si on ne parvient pas a initialiser la disquette, on doit aller à la routine de traitement des erreurs commençant en 1EDh dont le contenu est le suivant :

```
17CF:01ED BE9E7D        MOV     SI,7D9E
; si pointe sur le message commencant a l'adresse 19Eh du secteur d'amorcage
17CF:01F0 E85F00        CALL    0252
; appel de la routine d'affichage dont on ne reviendra pas
```

Le message affiché.- Comme nous le verrons, la routine de traitement des erreurs commence par afficher à l'écran un message, commençant en 7D9Eh, qui se trouve donc en 7D9Eh - 7C00h = 19Eh sur le secteur d'amorçage, qu'on peut lire avec `debug` en 100h + 19Eh = 29Eh :

```
C:\>debug secteura.bin
-d 129E
17CF:0290                                               0D 0A                ..
17CF:02A0  4E 6F 6E 2D 53 79 73 74-65 6D 20 64 69 73 6B 20   Non-System disk
17CF:02B0  6F 72 20 64 69 73 6B 20-65 72 72 6F 72 0D 0A 52   or disk error..R
17CF:02C0  65 70 6C 61 63 65 20 61-6E 64 20 70 72 65 73 53   eplace and press
17CF:02D0  20 61 6E 79 20 6B 65 79-20 77 68 65 6E 20 72 65    any key when re
17CF:02E0  61 64 79 0D 0A 00 49 4F-20 20 20 20 20 53 59   ady...IO      SY
```

(nous n'avons pas reporté les deux dernières lignes car nous allons voir qu'elle ne nous sont pas utiles pour l'instant).

Remarquons l'intérêt de la troisième colonne de l'éditeur hexadécimal dans un tel cas : elle nous évite d'avoir à chercher à la main à quel caractère correspond tel code ASCII.

Nous verrons que la chaîne de caractères à afficher se termine par le caractère '\0', qui se trouve ici en 2E5h. La suite ne nous intéresse donc pas.

En se souvenant que le passage à la ligne est dénotée par les deux octets 0Dh et 0Ah, le message est donc :

```
Non-System disk or disk error
Replace and press any key when ready
```

La routine d'affichage.- Nous avons vu ci-dessus que la routine d'affichage commence en 252h :

```
17CF:0252 AC            LODSB
; place un code ASCII du message dans al
17CF:0253 0AC0          OR      AL,AL
17CF:0255 7429          JZ      0200
; si fin du message aller a la sous-routine commencant en 200h
17CF:0257 B40E          MOV     AH,0E
; sinon fonction Eh de l'ecriture du caractere de code ASCII contenu en al
; en mode teletype de l'interruption 10h
17CF:0259 BB0700        MOV     BX,0007
; numero de page et couleur d'ecriture
17CF:025C CD10          INT     10
; appel de l'interruption
17CF:025E EBF2          JMP     0252
; aller au debut pour l'affichage du caractere suivant
```

La routine d'affichage en boucle.- Une fois l'affichage du message effectué, on va donc à la sous-routine commençant en 200h :

```
17CF:0200 58            POP     AX
17CF:0201 58            POP     AX
17CF:0202 58            POP     AX
17CF:0203 EBE8          JMP     01ED
```

qui attend un peu, en effectuant trois dépilements fictifs, et revient en boucle au début de la routine de traitement des erreurs.

Il faut donc redémarrer l'ordinateur lorsqu'on a changé la disquette et non appuyer sur une touche.

Mais ne devrait-on pas être renvoyé à 1F3h plutôt que 200h :

```
17CF:01F3 33C0          XOR     AX,AX
; ax = 0
17CF:01F5 CD16          INT     16
; attend un caractere au clavier
17CF:01F7 5E            POP     SI
17CF:01F8 1F            POP     DS
17CF:01F9 8F04          POP     [SI]
17CF:01FB 8F4402        POP     [SI+02]
; restaure les valeurs
17CF:01FE CD19          INT     19
; et redemarre
```

1.5.4.3 Calcul de l'emplacement du répertoire racine

S'il n'y a pas eu d'erreur lors de la réinitialisation de la disquette, on va à la ligne suivante, c'est-à-dire 174h. Les lignes `174h` à `1ACh` déterminent le numéro logique du premier secteur du répertoire racine :

```
17CF:0174 33C0          XOR     AX,AX          ; ax = 0
17CF:0176 3906137C      CMP     [7C13],AX
; le mot 13h du fichier specifie le nombre total de secteurs sur le disque (13h)
17CF:017A 7408          JZ      0184
; si ce qui est specifie est nul, ce qui est notre cas, aller deux lignes plus loin
17CF:017C 8B0E137C      MOV     CX,[7C13]
17CF:0180 890E207C      MOV     [7C20],CX
; sinon, ce qui n'est pas notre cas, conserver ce nombre total au mot 20h,
; correspondant au nombre total d'octets, de la copie du secteur d'amorcage
17CF:0184 A0107C        MOV     AL,[7C10]
; al = 2, nombre de FAT
17CF:0187 F726167C      MUL     WORD PTR [7C16]
; on le multiplie par le nombre de secteurs par FAT
17CF:018B 03061C7C      ADD     AX,[7C1C]
17CF:018F 13161E7C      ADC     DX,[7C1E]
; auquel on ajoute le nombre de secteurs caches
17CF:0193 03060E7C      ADD     AX,[7C0E]
17CF:0197 83D200        ADC     DX,+00
; et le nombre de secteurs reserves, ce qui donne le numero du premier secteur du repertoire
17CF:019A A3507C        MOV     [7C50],AX
17CF:019D 8916527C      MOV     [7C52],DX
; que l'on sauvegarde
17CF:01A1 A3497C        MOV     [7C49],AX
17CF:01A4 89164B7C      MOV     [7C4B],DX
; deux fois
17CF:01A8 B82000        MOV     AX,0020
; ax = 32, taille d'une entree du repertoire, en octets
17CF:01AB F726117C      MUL     WORD PTR [7C16]
; on multiplie par le nombre maximum d'entrees dans le repertoire
; ax:dx contient alors le nombre d'octets du repertoire racine
17CF:01AF 8B1E0B7C      MOV     BX,[7C0B]
; bx = nombre d'octets par secteur
17CF:01B3 03C3          ADD     AX,BX
; on ajoute le nombre d'octets du secteur d'amorcage
17CF:01B5 48            DEC     AX
; ax = ax-1 car on numerote les octets a partir de zero
17CF:01B6 F7F3          DIV     BX
; on divise par le nombre d'octets par secteur
; donc ax contient le nombre de secteurs
17CF:01B8 0106497C      ADD     [7C49],AX
17CF:01BC 83164B7C00    ADC     WORD PTR [7C4B],+00
; on sauvegarde le numero du premier secteur du repertoire
17CF:01C1 BB0005        MOV     BX,0500
; bx = 5 * 256 = 1280 ; es:bx est l'adresse du tampon de lecture du secteur
17CF:01C4 8B16527C      MOV     DX,[7C52]
17CF:01C8 A1507C        MOV     AX,[7C50]
; on restaure la valeur du numero du premier secteur du repertoire
17CF:01CB E89200        CALL    0260
; on appelle la sous-routine de transformation du numero logique en parametres physiques
```

1.5.4.4 Transformation du numéro logique en paramètres physiques

Avec un système d'exploitation, il suffit d'utiliser l'appel système de lecture du secteur ayant tel numéro logique. Mais aucun système d'exploitation n'est encore présent, nous ne disposons que des interruptions logicielles du BIOS. Il faut donc déterminer les paramètres physiques du premier secteur du répertoire racine : numéro de piste (cylindre), numéro de secteur sur cette piste et numéro de la tête de lecture-écriture.

Nous avons vu que, pour cela, le code précédent, de détermination de l'emplacement du répertoire racine, fait appel, une fois le numéro du premier secteur du répertoire racine déterminé, à la sous-routine commençant en 260h pour ces calculs :

```
; dx:ax contient le numero du premier secteur a lire
17CF:0260 3B16187C      CMP     DX,[7C18]
; le mot a l'emplacement 18h contient le nombre de secteurs par piste, pour nous 003Fh = 63
17CF:0264 7319          JNB     027F
; si ce nombre est trop grand, lever CF et terminer
17CF:0266 F736187C      DIV     WORD PTR [7C18]
; sinon diviser dx:ax par ce nombre de secteurs par piste
17CF:026A FEC2          INC     DL
; incrementer dl, contenant le numero de secteur sur la bonne piste,
; car on numerote les secteurs a partir de 1 et non de 0
17CF:026C 88164F7C      MOV     [7C4F],DL
; sauvegarder le numero de secteur a l'octet 4Fh
17CF:0270 33D2          XOR     DX,DX           ; dx = 0 car on va effectuer une division
17CF:0272 F7361A7C      DIV     WORD PTR [7C1A]
; on divise par le nombre de tetes de lecture-ecriture
17CF:0276 8816257C      MOV     [7C25],DL
; sauvegarde du numero de la tete comme octet 37
17CF:027A A34D7C        MOV     [7C4D],AX
; sauvegarder des numeros de cylindre et de secteur
17CF:027D F8            CLC
; on baisse le drapeau CF
17CF:027E C3            RET
; et on termine la routine
17CF:027F F9            STC
; on leve le drapeau CF
17CF:0280 C3            RET
; et on termine la routine
```

1.5.4.5 Lecture du premier secteur du répertoire racine

On n'a besoin de lire que le premier secteur du répertoire racine pour s'assurer que les deux premiers fichiers présents sont bien IO.SYS et MSDOS.SYS.

Après avoir calculé, à partir du numéro logique du premier secteur du répertoire racine, les paramètres physiques de celui-ci, on continue en 1CEh :

```
17CF:01CE 721D          JB      01ED
; si erreur aller a la routine de traitement des erreurs, dont on ne revient pas
17CF:01D0 B001          MOV     AL,01
; sinon lecture d'un seul secteur
17CF:01D2 E8AC00        CALL    0281
; appel de la sous-routine de lecture de secteurs
```

c'est-à-dire qu'on va à la routine de traitement des erreurs si le drapeau CF est levé, parce que le nombre de secteurs par piste est trop grand, et sinon on initialise al à 1 pour dire qu'on veut lire un seul secteur puis on appelle la sous-routine de lecture de secteurs.

La sous-routine de lecture de secteurs commence donc en 281h :

```
17CF:0281 B402          MOV     AH,02
; fonction 02h de l'interruption 13h de lecture de secteurs
17CF:0283 8B164D7C      MOV     DX,[7C4D]
; numeros de cylindre et de secteur intialises par la sous-routine precedente
17CF:0287 B106          MOV     CL,06
17CF:0289 D2E6          SHL     DH,CL
17CF:028B 0A364F7C      OR      DH,[7C4F]
17CF:028F 8BCA          MOV     CX,DX
17CF:0291 86E9          XCHG    CH,CL
; numero de cylindre dans ch
; numero de secteur dans cl
17CF:0293 8A16247C      MOV     DL,[7C24]
; numero d'unite de lecture dans dl, ici 80h pour premier disque dur
17CF:0297 8A36257C      MOV     DH,[7C25]
; numero de tete de lecture-ecriture dans dh
17CF:029B CD13          INT     13
; lecture du secteur, dont le contenu est place en es:bx, donc en 0000:0500
17CF:029D C3            RET
```

1.5.4.6 Fichiers systèmes présents ?

Une fois le répertoire racine chargé, on vérifie que les deux fichiers systèmes, de noms `io.sys` et `msdos.sys`, sont, non seulement présents, mais sont de plus les deux premiers fichiers du répertoire racine.

Après avoir fait appel à la routine de lecture du premier secteur du répertoire racine, on continue en 1D5h :

```
17CF:01D5 7216          JB      01ED
; aller a la routine de traitement des erreurs, dont on ne revient pas,
; si on n'est pas parvenu a lire le premier secteur du repertoire racine
17CF:01D7 8BFB          MOV     DI,BX
; di pointe sur la copie en memoire centrale du premier secteur du repertoire racine,
; donc sur la premiere entree de celui-ci
17CF:01D9 B90B00        MOV     CX,000B
; cx = 11, longueur d'un nom de fichier
17CF:01DC BEE67D        MOV     SI,7DE6
; si pointe sur la chaine de caracteres commencant a l'adresse 1E6h du secteur d'amorcage
; a savoir 'IO      SYS', comme nous le verrons ci-dessous
17CF:01DF F3            REPZ
17CF:01E0 A6            CMPSB
; on verifie que le premier nom de fichier du repertoire est bien IO.SYS
17CF:01E1 750A          JNZ     01ED
; si ce n'est pas le cas on va a la routine de traitement des erreurs, dont on revient pas
17CF:01E3 8D7F20        LEA     DI,[BX+20]
; di pointe sur la deuxieme entree du repertoire racine
; si pointe alors sur 'MSDOS  SYS', comme nous le verrons ci-dessous
17CF:01E6 B90B00        MOV     CX,000B
; cx = 11, longueur d'un nom de fichier
17CF:01E9 F3            REPZ
17CF:01EA A6            CMPSB
; on verifie que le premier nom de fichier du repertoire est bien MSDOS.SYS
17CF:01EB 74E18         JZ      0205
; si c'est le cas on va en 205h
17CF:01ED BE9E7D        MOV     SI,7D9E
; sinon on fait pointer si sur le message d'erreur se trouvant a l'adresse 19Eh
17CF:01F0 E85F00        CALL    0252
; et on appelle la routine d'affichage, dont on ne revient pas
```

Les deux noms de fichier.- Nous avons vu qu'on fait pointer SI sur la chaîne de 22 caractères commençant en 7DE6h, qui se trouve donc en 7DE6h - 7C00h = 1E6h sur le secteur d'amorçage, qu'on peut lire avec `debug` en 100h + 1E6h = 2E6h :

```
C:\>debug secteura.bin
-d 2E6
17CF:02E0                    49 4F-20 20 20 20 20 20 53 59        IO       SY
17CF:02F0  53 4D 53 44 4F 53 20 20-20 53 59 53 00 00 55 AA   SMSDOS   SYS..U.
```

(dont nous n'avons reporté que les deux premières lignes, puisqu'on arrive à la fin du secteur d'amorçage).

1.5.4.7 Lecture des trois premiers secteurs de IO.SYS

Nous venons de voir que si les deux fichiers système IO.SYS et MSDOS.SYS sont, non seulement présents, mais correspondent de plus aux deux premières entrées du répertoire racine, on va en 205h. On ne connaît pas la taille de IO.SYS, donc on en lit les trois premiers secteurs et on leur donne la main.

```
17CF:0205 8B471A        MOV     AX,[BX+1A]
; bx pointe sur le debut de la copie en memoire centrale du repertoire depuis la ligne 01C1h
; donc ax contient le numero de la premiere unite d'allocation du fichier IO.SYS
; donx ax = 2
17CF:0208 48            DEC     AX
17CF:0209 48            DEC     AX
; ax = ax - 2 = 0 on diminue de 2 car la premiere unite d'allocation a 2 pour numero
17CF:020A 8A1E0D7C      MOV     BL,[7C0D]
; l'octet 0D du specifie le nombre de secteurs par unite d'allocation, ici 32
17CF:020E 32FF          XOR     BH,BH
; bh = 0
17CF:0210 F7E3          MUL     BX
; on multiplie pour obtenir le numero du premier secteur du fichier IO.SYS
17CF:0212 0306497C      ADD     AX,[7C49]
17CF:0216 13164B7C      ADC     DX,[7C4B]
; dx:ax contient alors le nombre de secteurs jusqu'au repertoire inclus
17CF:021A BB0007        MOV     BX,0700
; tampon pour lire les secteurs
17CF:021D B90300        MOV     CX,0003
; lire trois secteurs
17CF:0220 50            PUSH    AX
17CF:0221 52            PUSH    DX
17CF:0222 51            PUSH    CX
; sauvegarde sur la pile des contenus de ax, dx et cx
17CF:0223 E83A00        CALL    0260
; appel de routine de transformation du numero logique du secteur en parametres physiques
17CF:0226 72D8          JB      0200
; on affiche le message d'erreur habituel si on n'y parvient pas, dont on ne revient pas
17CF:0228 B001          MOV     AL,01
; al = 1 pour lecture d'un secteur
17CF:022A E85400        CALL    0281
; appel de la routine de lecture d'un secteur
17CF:022D 59            POP     CX
17CF:022E 5A            POP     DX
17CF:022F 58            POP     AX
; on restaure les contenus de cx, dx et ax
17CF:0230 72BB          JB      01ED
; si on n'est pas parvenu a lire le secteur, on va a la routine de traitement des erreurs,
; dont on ne revient pas
17CF:0232 050100        ADD     AX,0001
17CF:0235 83D200        ADC     DX,+00
; on passe au secteur suivant
17CF:0238 031E0B7C      ADD     BX,[7C0B]
; le tampon de lecture d'un secteur incremente du nombre d'octets par secteur
17CF:023C E2E2          LOOP    0220
; on recommence deux fois pour lire les trois premiers secteurs de IO.SYS
17CF:023E 8A2E157C      MOV     CH,[7C15]
; ch = F8h, le type d'unite, ici un disque dur
17CF:0242 8A16247C      MOV     DL,[7C24]
; dl = 80h, le numero d'unite, ici le premier disque dur
17CF:0246 8B1E497C      MOV     BX,[7C49]
17CF:024A A14B7C        MOV     AX,[7C4B]
; dx:ax contient alors le nombre de secteurs jusqu'au repertoire inclus
17CF:024D EA00007000    JMP     0070:0000
; donner la main au programme contenu dans ces trois secteurs
```

1.6 Appendice : MS-DOS sur une clé USB

Dans nos premiers exemples, nous supposons que l'on dispose d'un PC muni du système d'exploitation MS-DOS. Si vous disposez déjà d'un système informatique avec MS-DOS (en mode réel, c'est-à-dire jusqu'à la version 6.22), que ce soit un ordinateur auxiliaire ou un double démarrage, il est inutile d'installer MS-DOS. Mais c'est rare de nos jours.

Si vous disposez d'un ordinateur avec un lecteur de disquettes, vous pouvez partitionner votre disque dur et installer MS-DOS sur la première partition. Mais c'est également rare de nos jours. Cela peut également se faire, en jonglant un peu, si l'ordinateur dispose d'un lecteur de CD-ROM.

Le cas le plus fréquent est un ordinateur PC avec un microprocesseur *Intel* puissant, le plus souvent 64 bits, et, pour ce qui nous intéresse, des ports USB.

Nous allons donc installer MS-DOS sur une clé USB. On choisira soit MS-DOS 5.0 disponible gratuitement, soit MS-DOS 6.22, toujours sous licence Microsoft mais disponible gratuitement, comme tous les autres outils de développement Microsoft, pour les étudiants dans un département abonné au système MSDN (ou par d'autres moyens, illicites cependant).

1.6.1 Première étape : récupérer MS-DOS

Il faut récupérer le fichier `en_msdos22.exe` sur MSDN (ou de façon illicite). Placez-le dans un répertoire vide, par exemple de nom '`en_MSDOS622`', et faites exécuter (sous Windows) ce fichier auto-extractible. On obtient deux sous-répertoires : '`DISKS`' et '`UPGRADE`'.

Ces répertoires contiennent à la fois des fichiers au nom bizarre et des fichiers exécutables. Ce sont ces derniers qui nous intéresseront.

1.6.2 Seconde étape : créer une clé USB bootable

Une **clé bootable** est une clé USB qui, lorsqu'on démarre l'ordinateur avec la clé dans un des connecteurs USB (et en spécifiant au BIOS de commencer par chercher un système d'exploitation sur une telle clé, avant d'aller chercher sur un disque dur ou un CD-ROM), charge le système d'exploitation qui se trouve sur celle-ci.

Récupération d'un utilitaire.- Nous verrons plus tard comment on crée une clé bootable. Utilisons pour le moment un utilitaire pour cela. Spécifier '`HP Drive Key Boot Utility v2.1.8 download`' sur votre moteur de recherche favori. Ceci ne renvoie plus au site de Hewlett-Packard car l'utilitaire a évolué (mais la nouvelle version ne convient pas). On peut le récupérer, par exemple (au moment où nous écrivons) sur le site Web :

`http://files.extremeoverclocking.com/file.php?f=197`

en cliquant sur '*Primary Download Site*', ce qui permet de télécharger le fichier '`SP27608.exe`' de 1,97 MiO. Placez-le dans un répertoire vide de votre Windows, par exemple '`HP DriveKey`' et faites-le exécuter. On se retrouve avec quatre fichiers, dont `HPUSBFW.exe` pour *HP USB Disk Storage Format Tool FreeWare*.

Fichiers MS-DOS essentiels.- Préparez un autre répertoire vide de votre Windows, nommé par exemple '`MSDOS622`' et placez-y les trois fichiers '`command.com`', '`io.sys`' et '`msdos.sys`' trouvés dans le sous-répertoire '`UPGRADE`' de '`en_MSDOS622`'. Attention ! ces fichiers sont « cachés » car considérés comme fichiers système : il faut aller dans '`Panneau de configuration`', '`Apparence et personnalisation`' puis '`Afficher les fichiers et dossiers cachés`' pour désélectionner, tout au moins de façon temporaire, l'option '`Masquer les fichiers protégés du système d'exploitation (recommandé)`' afin de rendre visible les fichiers système.

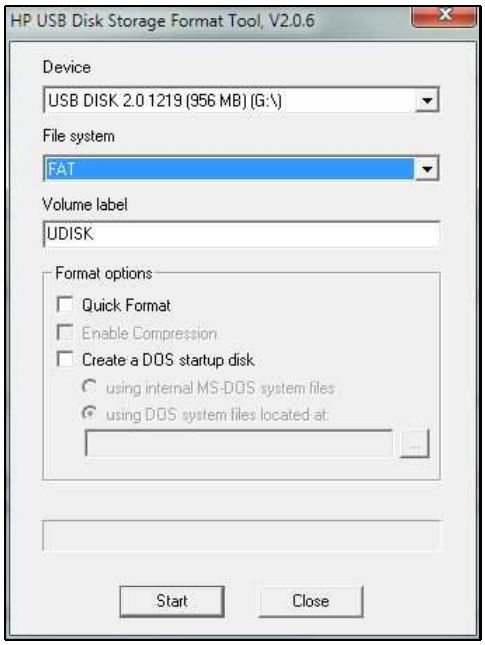

FIGURE 1.2 – Fenêtre de `HP USB Disk Storage Format Tool`

Placement de MS-DOS sur la clé USB.- Insérez une clé USB (de moins de 2 GiO) dans un connecteur USB et faites exécuter l'utilitaire `HPUSBFW.exe` en tant qu'administrateur. On obtient la fenêtre de la figure 1.2. Vérifiez que le volume indiqué dans '`Device`' est bien celui de la clé que vous voulez rendre bootable. Choisissez le système de fichiers FAT16, c'est-à-dire 'FAT'. Choisissez éventuellement un nom de volume autre que celui indiqué par défaut. Cliquez sur '*Create a DOS startup disk*' puis sur '*using DOS system files located at :*'. Appuyez sur les trois petits points pour rechercher le dossier '`MSDOS622`' précédemment préparé puis sur le bouton '*Start*'. La fenêtre de la figure 1.3 s'ouvre.

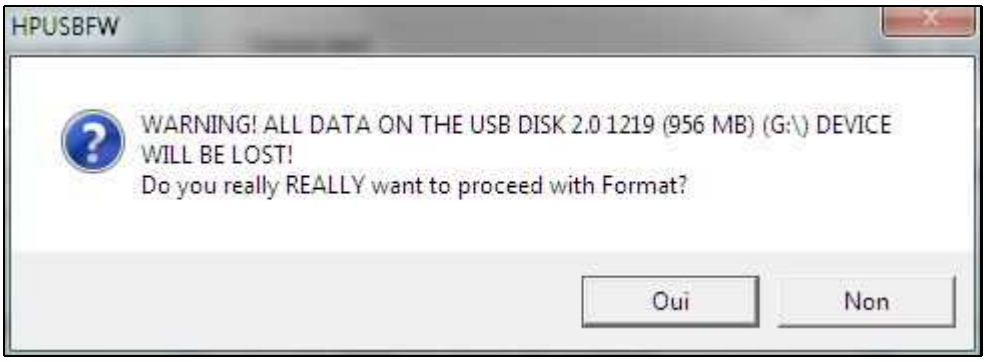

FIGURE 1.3 – Fenêtre d'avertissement de `HP USB Disk Storage Format Tool`

Après une dernière vérification qu'il s'agit bien de la clé USB voulue, appuyez sur le bouton '*Oui*'.

Démarrage de MS-DOS.- Une fois l'opération terminée, redémarrez l'ordinateur en laissant la clé USB dans le connecteur. Il va falloir indiquer au BIOS de booter sur une clé USB en changeant l'ordre du choix de périphérique de boot. Si tout s'est bien passé, on se retrouve avec MS-DOS (avec clavier Qwerty). Plus exactement on voit apparaître :

```
Starting MS-DOS...

Current date is Wed 01-09-2013
Enter new date (mm-dd-yy):
```

Appuyer sur le touche '`retour`'. On a alors :

```
Starting MS-DOS...

Current date is Wed 01-09-2013
Enter new date (mm-dd-yy):
Current time is 11:01:58.32a
Enter new time:
```

Appuyer sur le touche '`retour`'. On se retrouve avec :

```
Starting MS-DOS...

Current date is Wed 01-09-2013
```

```
Enter new date (mm-dd-yy):
Current time is 11:01:58.32a
Enter new time:


Microsoft(R) MS-DOS(R) Version 6.22
            (C)Copyright Microsoft Corp 1981-1994

C:\>
```

On se retrouve avec le prompteur que l'on trouve sur l'émulateur de ligne de commande '`command.com`' de Windows.

On peut commencer à explorer MS-DOS mais on se trouve dans la version anglaise avec clavier querty.

1.6.3 Troisième étape : version française complète

Pour la suite on va se placer dans la version avec clavier azerty et ajouter les utilitaires avec lesquels nous allons travailler, en particulier '`debug`'.

Utilitaires MS-DOS.- Enlevez la clé USB et redémarrez l'ordinateur pour se retrouver avec Windows. Placez la clé dans un connecteur USB. Créez un répertoire '`DOS`' sur la clé et placez-y les fichiers '`COUNTRY.SYS`', '`DEBUG.EXE`', '`EDIT.COM`', '`KEYB.COM`', '`KEYBOARD.SYS`', '`MORE.COM`' et '`QBASIC.EXE`' du sous-répertoire '`UPGRADE`' de '`en_MSDOS622`'.

Fichiers de configuration.- Préparer un fichier de nom '`AUTOEXEC.BAT`' (avec `notepad` par exemple ou tout autre éditeur de texte) à placer à la racine de la clé USB :

```
@ECHO OFF
PROMPT $p$g
PATH C:\DOS
KEYB FR,,C:\DOS\KEYBOARD.SYS
```

Préparer aussi un fichier de nom '`CONFIG.SYS`' à placer également à la racine de la clé USB :

```
COUNTRY=033,850,C:\DOS\COUNTRY.SYS
```

Notre version finale de MS-DOS.- Redémarrez l'ordinateur en laissant la clé USB dans le connecteur. On voit maintenant apparaître :

```
Starting MS-DOS...

C:\>
```

On peut maintenant utiliser MS-DOS avec clavier azerty.