Chapitre 3

Initialisation de MS-DOS

3.1 Vue d'ensemble du chargement de MS-DOS

Les étapes de la **procédure de démarrage** (en anglais *System Startup Procedure*) d'un ordinateur muni du système d'exploitation MS-DOS sont les suivantes :

— Comme nous l'avons rappelé dans le premier chapitre, consacré au démarrage d'un système d'exploitation, quel qu'il soit, lors de la mise sous tension de la machine le microprocesseur *Intel* exécute l'instruction située à l'adresse `FFFF0h`, qui doit donc 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.

Le BIOS (de l'IBM-PC) effectue quelques tests sur le matériel (microprocesseur, ROM, RAM, coupleurs), initialise les périphériques, établit deux emplacements de données du BIOS, et, si tout s'est bien passé, essaie de lire le secteur de démarrage de la disquette `A:`, le charge à l'adresse `7C00h` et lui donne la main.

— Dans le cas de MS-DOS, le programme situé sur l'enregistrement de démarrage lit le répertoire racine de la disquette pour vérifier que les fichiers (cachés) `IO.SYS` et `MSDOS.SYS` sont bien présents. Si c'est le cas, pour MS-DOS 1.25, l'enregistrement de démarrage lit ces deux fichiers `IO.COM` et `MSDOS.COM` et en place le contenu en mémoire centrale au segment `60h`, décalage `0h` (donc à l'adresse absolue `600h`) et passe le contrôle à cette adresse, correspondant au premier octet de `IO.COM`.

À partir de MS-DOS 2.0, les fichiers système, renommés `IO.SYS` et `MSDOS.SYS`, étant de tailles trop importantes, il ne charge en mémoire centrale que les trois premiers secteurs du programme `IO.SYS` et passe toujours le contrôle au premier octet de la copie en mémoire centrale des trois premiers secteurs de `IO.SYS`.

— Pour MS-DOS 1.25, le début du code de `IO.COM` initialise les pilotes de périphériques (console, écran, disque, port série, notamment). À ce moment, les périphériques par défaut sont disponibles. `IO.COM` donne déplace le code de `MSDOS.COM` pour qu'il se situe immédiatement après celui de IO.COM et lui donne la main, pour que soit exécutée sa routine d'initialisation.

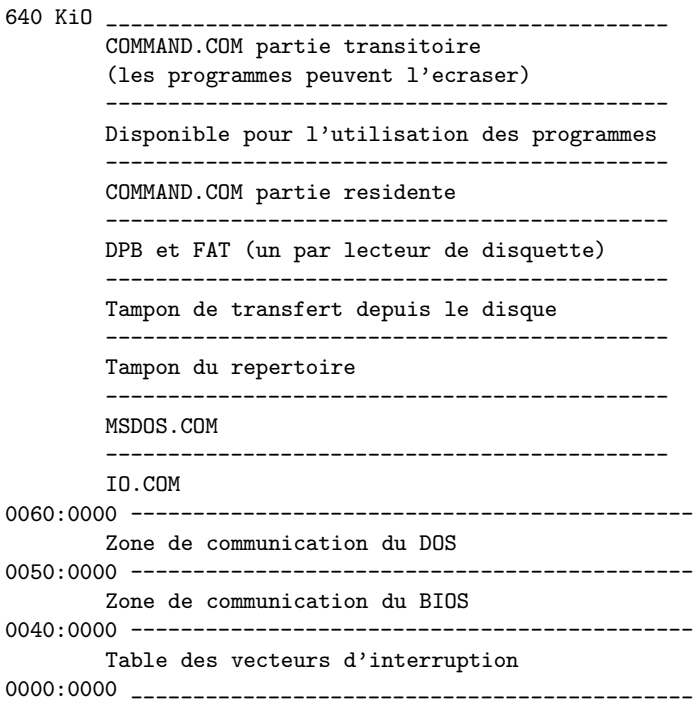
À partir de MS-DOS 2.0, le début du code de `IO.SYS`, celui contenu sur les trois premiers secteurs, charge `IO.SYS` complètement en mémoire, initialise les pilotes de périphériques, charge alors le programme `MSDOS.SYS` et donne la main à sa routine d'initialisation.

— À ce moment, un accès au système de fichiers est disponible. On retourne à `IO.SYS` pour charger et passer le contrôle à `COMMAND.COM`.

Cet interpréteur de commandes comprend trois parties : la **partie résidente** reste en permanence en mémoire vive (à une adresse fixée par Microsoft) ; la **partie initialisation** n'est exécutée que lors du chargement de MS-DOS ; la **partie transitoire** (en anglais *transient part*) est chargée en partie haute de la mémoire et peut être écrasée si le besoin s'en fait sentir.

La partie résidente s'occupe des messages d'erreur et implémente les routines de service des interruptions concernant les erreurs critiques et CTRL-Break. La partie initialisatio lit le fichier de traitement par lot `AUTOEXEC.BAT` et l'exécute. La partie transitoire entre dans un cycle sans fin : faire apparaître une invite de commande et interprète la commande tapée au clavier.

À ce moment, la mémoire centrale est structurée comme le montre la figure ci-dessous pour MS-DOS 1.25 :



On peut retrouver l'adresse de segment de MSDOS.COM comme valeur de la variable se trouvant à l'emplacement absolu `0000:0086`.

On peut retrouver l'adresse de segment de la portion résidente de COMMAND.COM comme valeur de la variable se trouvant à l'emplacement absolu `0000:009E`.

La zone de communication du DOS est structurée de la façon suivante :

- `0050:0000` : drapeau de statut d'impression de l'écran :
  - 0 : impression écran non actif ou succès de l'opération d'impression écran ;
  - 1 : impression écran en cours ;
  - 255 : erreur rencontrée durant l'opération d'impression écran.
- `0050:0004` : octet de statut du mode un seul lecteur de disquette :
  - 0 : la disquette a été utilisée pour le lecteur A : en dernier ;
  - 1 : la disquette a été utilisée pour le lecteur B : en dernier.
- `0050:0080` : gestionnaire de l'interruption INT 14h (*Asynchronous Communications Adapter*), chargé par MODE pour être effectué à nouveau après une pause lorsqu'une improimante série est présente.
- `0050:00C0` : gestionnaire de l'interruption INT 17h (imprimante parallèle), chargé par MODE pour intercepter les requêtes à l'imprimante et les rediriger vers COM 1 ou COM2.

3.2 Première partie d'initialisation de IO.COM

Comme nous l'avons dit, le programme du secteur de démarrage de MS-DOS 1.25 charge les fichiers IO.COM et MSDOS.COM, plus exactement les secteurs 2 à ? ? de la disquette, à l'adresse `600h` puis donne la main à cette adresse, donc au début de IO.COM.

Le contenu de ce fichier est chargé de définir un certain nombre de pré-routines de service des fonctions de l'interruption `21h`, d'initialiser MS-DOS, en faisant exécuter une partie de MSDOS.COM, et enfin de charger et de donner la main à COMMAND.COM, dont on ne reviendra jamais.

Lorsque le programme du secteur de démarrage donne la main à IO.COM, on exécute ce qui se trouve au début de son code, ce qui correspond dans le fichier IO.ASM à :

```
109 :     ORG   0
110 :     PUT   100H
111 :
112 :     JMP   INIT
113 :     JMP   STATUS
114 :     JMP   INP
115 :     JMP   OUTP
116 :     JMP   PRINT
117 :     JMP   AUXIN
118 :     JMP   AUXOUT
119 :     JMP   READ
120 :     JMP   WRITE
121 :     JMP   DSKCHG
122 :     JMP   SETDATE
123 :     JMP   SETTIME
124 :     JMP   GETTIME
125 :     JMP   FLUSH
126 :     JMP   MAPDEV
127 : MAPDEV:
128 :     RET   L
129 :
130 : INIT:
131 :     XOR   BP,BP      ; Set up stack just below I/O system.
132 :     MOV   SS,BP
133 :     MOV   SP,BIOSSEG*16
134 :
135 :     IF    INTINP-1
136 :     MOV   AL,0FFH    ; Mask all interrupts.
137 :     OUTB  BASE+3
138 :     ENDIF
139 :
140 :     IF    INTINP
141 :     DI               ; Set up keyboard interrupt vector.
142 :     MOV   [BP+64H],KBINT
143 :     MOV   [BP+66H],CS
144 :     EI
145 :     ENDIF
146 :
147 :     MOV   [BP+4*38H],PRNFCB
148 :     MOV   [BP+4*38H+2],CS
149 :     PUSH  CS
150 :     POP   DS
151 : ;
152 : ; Initialize time-of-day clock.
153 : ;
154 :     MOV   SI,STCTAB
155 :     MOV   CX,4        ;Initialize 4 registers
156 :     UP
```

```
157 : INITSTC:
158 :     LODB
159 :     OUT   STCCOM      ;Select register to initialize
160 :     LODB
161 :     OUT   STCDATA
162 :     LODB
163 :     OUT   STCDATA
164 :     LOOP  INITSTC
165 :
166 :     IF    SERIAL
167 :     MOV   CX,4
168 : SERINIT:
169 :     LODB
170 :     OUT   SIOBASE+1
171 :     OUT   SIOBASE+3
172 :     LOOP  SERINIT
173 :     LODB               ;Baud rate for channel 0
174 :     OUT   SIOBASE+8
175 :     LODB               ;Baud rate for channel 1
176 :     OUT   SIOBASE+9
177 :     ENDIF
178 : ;
179 : ; Move MS-DOS down to the first segment just above the I/O system.
180 : ;
181 :     MOV   SI,BIOSLEN   ; Source points to where MS-DOS currently is.
182 :     MOV   AX,DOSSEG    ; Destination is beginning of DOSSEG.
183 :     MOV   ES,AX
184 :     SUB   DI,DI
185 :     MOV   CX,DOSLEN/2  ; CX is number of words to move.
186 :     REP
187 :     MOVSW
188 :
189 :     MOV   SI,INITTAB
190 :     MOV   DX,1         ; Do auto memory scan.
191 :     CALL  0,DOSSEG
```

Commentaires.- 1^o) La ligne 109 marque le début du code de IO.COM. La première instruction est un saut vers la partie de code repérée par l'étiquette INIT, débutant ligne 130.

Initialisation de la pile

2^o) On place la pile avant le code de IO.COM : le pointeur de pile est initialisé à 0 et l'adresse du segment de pile à 1024, c'est-à-dire juste après la table des vecteurs d'interruption (lignes 131–133).

L'adresse du segment de IO.COM est donnée par la constante BIOSSEG, dont la valeur est 1 024 donc juste après la table des vecteurs d'interruption :

```
74 : BIOSSEG:EQU    40H        ; I/O system segment.
```

Puisque IO.COM commence à l'adresse `600h`, soit 1 536, cela donne une pile de 512 octets.

Façon de lire un caractère au clavier

3^o) Suivant que la lecture d'un caractère au clavier s'effectue par interrogation ou par interruption respectivement, on inhibe toutes les interruptions masquables ou on initialise le vecteur d'interruption correspondant, c'est-à-dire celui de l'interruption `16h`, et on se met à l'écoute des interruptions (lignes 135–145).

Avant l'assemblage du fichier IO.ASM, on choisit la façon de faire en initialisant de façon adéquate la constante INTINP :

```
6 : ; Select whether console input is interrupt-driven or polled.
7 : INTINP:     EQU   1
```

Le numéro de port de base de la carte est spécifié par la constante BASE :

```
79 : BASE: EQU    0F0H        ; CPU Support card base port number.
```

L'étiquette KBINT repère la routine de service de l'interruption `16h` définie dans IO.COM, ligne 405, remplaçant celle du BIOS, que nous étudierons en même temps que les autres routines et pré-routines de service de IO.COM.

Initialisation du FCB de l'imprimante

4^o) On place l'adresse du bloc de contrôle du fichier de l'imprimante comme vecteur d'interruption (fictif) de l'interruption `38h` et on confond le segment des données avec le segment de code (lignes 147–150).

Le bloc de contrôle du fichier correspondant à l'imprimante est repéré par l'étiquette PRNFCB, dont le premier octet est initialisée à `FFh` pour indiquer qu'il s'agit d'un FCB étendu :

```
559 : PRNFCB:  DB    -1
560 :          DS    36
```

Initialisation de l'heure

5^o) On fait pointer `SI` au début du tableau STCTAB, on place 4 dans `CX` parce qu'on va envoyer quatre commandes au minuteur et, pour chacune de ces quatre commandes, on place le premier octet de l'entrée correspondante du tableau dans le registre STCCOM pour spécifier le type de la commande, le second et le troisième octet dans le registre STCDATA des données du coupleur, pour spécifier les paramètres de la commande. La première commande sélectionne le coupleur en mode maître puis on fait démarrer le décompte de l'heure, le premier compteur, le deuxième compteur et le décompte des jours (lignes 151–164).

Les deux registres, des données et du type de commande, du minuteur sont repérés par les constantes STCDATA et STCCOM :

```
86  : STCDATA:  EQU   BASE+4     ; Ports for 9513 Timer chip.
87  : STCCOM:   EQU   BASE+5
```

Le tableau de contrôle d'initialisation de l'heure est STCTAB (pour *Set Time Control TABle*) :

```
249 : STCTAB:   DB    17H         ;Select master mode register
250 :           DW    84F3H       ;Enable time-of-day
251 :           DB    1           ;Counter 1 mode register
252 :           DW    0138H
253 :           DB    2
254 :           DW    0038H
255 :           DB    3
256 :           DW    0008H       ;Set counter 3 to count days
```

Sélection et initialisation du port série comme port auxiliaire

6^o) Avant d'assembler le fichier source IO.ASM, il faut choisir la nature du port auxiliaire (parallèle ou série). Si on a choisi le port série, on initialise les paramètres de celui-ci (lignes 166–177).

On choisit le port série grâce à la constante SERIAL :

```
85 : SERIAL:    EQU    SERIALPRN+SERIALAUX
```

Les constantes PARALLELAUX, SERIALAUX, PARALLELPRN et SERIALPRN, permettent, avant l'assemblage, de spécifier la nature du port auxiliaire et la façon dont l'imprimante est reliée :

```
9  : ; Select whether the auxiliary port is the Support Card parallel port
10 : ; or on channel 1 of a Multiport Serial card addressed at 10H.
11 : PARALLELAUX:   EQU    1
12 : SERIALAUX:     EQU    0
13 : ;
14 : ; Select whether the printer is connected to the Support card parallel
15 : ; output port (standard) or channel 0 of a Multiport Serial card
16 : ; addressed at 10H.
17 : PARALLELPRN:   EQU    1
18 : SERIALPRN:     EQU    0
```

Le numéro de port de base de la carte série est donnée par la constante SIOBASE :

```
80 : SIOBASE:       EQU    10H        ; Base port number of Multiport Serial card.
```

Le tableau STCTAB se continue de la façon suivante :

```
257 :
258 :  IF SERIAL
259 :  DB 0B7H, 77H, 4EH, 37H, PRNBAUD, AUXBAUD
260 :  ENDIF
```

les constantes PRNBAUD et AUXBAUD étant définies de la façon suivante :

```
20 : ; If the Multiport Serial was chosen for either the auxiliary or the
21 : ; printer, select the baud rate here. Refer to Multiport Serial manual
22 : ; page 11 to pick the correct value for a given baud rate.
23 : PRNBAUD:EQU 7        ; 1200 baud
24 : AUXBAUD:EQU 0FH      ; 19200 baud
```

Transfert de la copie de MSDOS.COM juste après celle de IO.COM

7^o) Comme on ne connaît pas la taille exacte de IO.COM avant son chargement, tout en exigeant que la taille maximum de ce fichier soit 2 kiO, soit 4 secteurs comme nous l'avons dit, le programme du secteur de démarrage charge en mémoire centrale le contenu de ce fichier dans 2 kiO, puis le contenu de MSDOS.COM juste derrière : en fait, la notion de fichier n'est pas connue, il charge ? ? secteurs, qui contiennent ces deux fichiers. Une fois IO.COM chargé, on peut en déterminer la taille exacte et donc déplacer la copie de MSDOS.COM en mémoire centrale pour que celle-ci suive immédiatement la copie en mémoire de IO.COM, en s'alignant cependant sur un paragraphe.

Pour cela, on fait pointer `SI` sur l'adresse actuelle de la copie de MSDOS.COM, c'est-à-dire celle à laquelle le contenu de MSDOS.COM a été chargé, on prend comme adresse du segment supplémentaire celle qui suit la copie de IO.COM, en s'alignant (nécessairement) sur un paragraphe, puis on déplace la copie de MSDOS.COM pour qu'elle soit amenée juste après celle du code de IO.COM (lignes 178–187).

La taille maximum de IO.COM est spécifiée par la constante BIOSLEN :

```
75 : BIOSLEN:EQU    2048        ; Maximum length of I/O system.
```

L'adresse de segment de MSDOS.COM est spécifiée par la constante DOSSEG :

```
1929 : DOSSEG:    EQU    ($+15)/16+BIOSSEG ; Compute segment to use for 86-DOS.
```

Appel de MSDOS.COM

8^o) Une fois le code de MSDOS.COM déplacé, on lui donne la main. Pour cela, on fait pointer `SI` sur la table d'initialisation, on place 1 dans `DX`, pour spécifier à MSDOS.COM d'analyser la mémoire et on effectue un saut au début de la copie de MSDOS.COM (lignes 189–191).

La table INITTAB dépend du lecteur de disquette choisi :

```
1460 : ; Explanation of tables below.
1461 : ;
1462 : ; DRVTAB is a table of bytes which are sent to the disk controller as drive-
1463 : ; select bytes to choose which physical drive is selected for each disk I/O
1464 : ; driver.  It also selects whether the disk is 5.25-inch or 8-inch, single-
1465 : ; density or double-density.  Always select side 0 in the drive-select byte if
1466 : ; a side-select bit is available.  There should be one entry in the DRVTAB
1467 : ; table for each disk I/O driver.  Exactly which bits in the drive-select byte
1468 : ; do what depends on which disk controller is used.
1469 : ;
1470 : ; TRKTAB is a table of bytes used to store which track the read/write
1471 : ; head of each drive is on.  Each physical drive should have its own
1472 : ; entry in TRKTAB.
1473 : ;
1474 : ; TRKPT is a table of bytes which indicates which TRKTAB entry each
1475 : ; disk I/O driver should use.  Since each physical drive may be used for
1476 : ; more than one disk I/O driver, more than one entry in TRKPT may point
1477 : ; to the same entry in TRKTAB.  Drives such as PerSci 277s which use
1478 : ; the same head positioner for more than one drive should share entrys
1479 : ; in TRKTAB.
1480 : ;
1481 : ; INITTAB is the initialization table for 86-DOS as described in the
1482 : ; 86-DOS Programer's Manual under "Customizing the I/O System."
1483 : ;
1484 :  IF    SCP*COMBIN*FASTSEEK
1485 : ;
1486 : ; A PerSci 277 or 299 and one 5.25-inch drive.
1487 : ;
1488 : DRVTAB:   DB    00H,08H,01H,09H,10H,18H,00H,08H,01H,09H
1489 : TRKPT:    DB    0,0,0,0,1,1,0,0,0,0
1490 : TRKTAB:   DB    -1,-1
1491 : INITTAB:
1492 :  IF    CONVERT-1
1493 :           DB    6     ; Number of disk I/O drivers.
1494 :  ENDIF
1495 :
1496 :  IF    CONVERT
1497 :           DB    10
1498 :  ENDIF
1499 :
1500 :          DB    0       ; Disk I/O driver 0 uses disk drive 0.
1501 :          DW    LSDRIVE ; Disk I/O driver 0 is 8-inch single-density.
1502 :          DB    0       ; Disk I/O driver 1 uses disk drive 0.
1503 :          DW    LDDRIVE ; Disk I/O driver 1 is 8-inch double-density.
1504 :          DB    1       ; Etc.
1505 :          DW    LSDRIVE
1506 :          DB    1
1507 :          DW    LDDRIVE
1508 :          DB    2
1509 :          DW    SSDRIVE
1510 :          DB    2
1511 :          DW    SDDRIVE
1512 :
1513 :  IF    CONVERT
```

```
1514 :          DB    3
1515 :          DW    OLDLSDRIVE
1516 :          DB    3
1517 :          DW    OLDLDDRIVE
1518 :          DB    4
1519 :          DW    OLDLSDRIVE
1520 :          DB    4
1521 :          DW    OLDLDDRIVE
1522 :  ENDIF
1523 :  ENDIF
1524 :
1525 :  IF    SCP*LARGE*FASTSEEK
```

Le choix du contrôleur de lecteur de disquette s'effectue, avant assemblage, grâce aux constantes SCP, TARBELLSD, TRABELLDD, COMENCO4FDC et CROMENCO16FDC :

```
26 : ; Select disk controller here.
27 : SCP:              EQU   1
28 : TARBELLSD:        EQU   0
29 : TARBELLDD:        EQU   0
30 : CROMEMCO4FDC:     EQU   0
31 : CROMEMCO16FDC:    EQU   0
32 : ;
33 : ; Select if you want a special conversion version which can read/write
34 : ; both the new Microsoft format and the old SCP format.
35 : ; For a two drive system, drives A and B are the new Microsoft format,
36 : ; and drives C and D are the old SCP format (where C is the same physical
37 : ; drive as A, and D is the same drive as B).  CONVERT has no effect
38 : ; on 5.25-inch drives.
39 : CONVERT:          EQU   1
40 : ;
41 : ; Select disk configuration:
42 : LARGE:            EQU   1     ; Large drives.
43 : COMBIN:           EQU   0     ; Two 8-inch and one 5.25-inch.
44 : SMALL:            EQU   0     ; Three 5.25-inch drives.
45 : CUSTOM:           EQU   0     ; User defined.
46 : ;
47 : ; If 8-inch drives are PerSci, select FASTSEEK here:
48 : ; (Fastseek with Tarbell controllers doesn't work yet).
49 : FASTSEEK:         EQU    1
50 : ;
51 : ; For double-density controllers, select double-sided operation of
52 : ; 8-inch disks in double-density mode.
53 : LARGEDS:          EQU    0
54 : ;
55 : ; For double-density controllers, select double-sided operation of
56 : ; 5.25-inch disks in double-density mode.
57 : SMALLDS:          EQU    0
58 : ;
59 : ; Use table below to select head step speed. Step times for 5" drives
60 : ; are double that shown in the table. Times for Fast Seek mode (using
61 : ; PerSci drives) is very small - 200-400 microseconds.
62 : ;
63 : ; Step value     1771  1793
64 : ;
65 : ;     0           6ms   3ms
66 : ;     1           6ms   6ms
67 : ;     2          10ms  10ms
68 : ;     3          20ms  15ms
69 : ;
70 : ; STPSPD:         EQU    0
71 : ;
```

```
72 : ; ****** End of selections ******************************************
```

3.3 La partie initialisation de MSDOS.COM

La partie initialisation de MSDOS.COM initialise ses tables internes, détermine les emplacements mémoire de la copie en mémoire centrale de la FAT de chaque lecteur de disquette, du répertoire, des tampons de données, initialise les vecteurs d'interruption des interruptions MS-DOS, c'est-à-dire `20h` à `27h`, et commence à construire un PSP pour le programme COMMAND.COM, en le plaçant sur le segment disponible le plus bas possible, puis revient à IO.COM.

3.3.1 Assemblage de MSDOS.ASM

Pour la version 1.25, le fichier MSDOS.COM est obtenu en assemblant le fichier `STDDOS.ASM`, de 21 lignes, initialisant un certain nombre de constantes (FALSE, TRUE, MSVER pour la version *Microsoft*, soit MS-DOS, destinée aux constructeurs de PC compatibles, IBM pour la version PC-DOS) puis appelant le noyau proprement dit, à savoir MSDOS.ASM :

```
1  : TITLE    MS-DOS version 1.25 by Tim Paterson     March 3, 1982
2  : PAGE     60,132
3  : ; Use the following booleans to set the switches
4  : FALSE    EQU    0
5  : TRUE     EQU    NOT FALSE
6  :
7  : ; Use the switches below to produce the standard Microsoft version of the IBM
8  : ; version of the operating system
9  : MSVER    EQU    TRUE
10 : IBM      EQU    FALSE
11 :
12 : ; Set this switch to cause DOS to move itself to the end of memory
13 : HIGHMEM  EQU    FALSE
14 :
15 : ; Turn on switch below to allow testing disk code with DEBUG. It sets
16 : ; up a different stack for disk I/O (functions > 11) than that used for
17 : ; character I/O which effectively makes the DOS re-entrant.
18 :
19 : DSKTEST  EQU    FALSE
20 :
21 :          INCLUDE MSDOS.ASM
```

Le début du fichier source MSDOS.ASM nous prouve qu'il a été écrit par Tom Paterson à partir de 86-DOS :

```
1  : ; 86-DOS  High-performance operating system for the 8086  version 1.25
2  : ;       by Tim Paterson
3  :
4  :
5  : ; ****************** Revision History ************************
6  : ;          >> EVERY change must noted below!! <<
7  : ;
8  : ; 0.34 12/29/80 General release, updating all past customers
9  : ; 0.42 02/25/81 32-byte directory entries added
10 : ; 0.56 03/23/81 Variable record and sector sizes
11 : ; 0.60 03/27/81 Ctrl-C exit changes, including register save on user stack
12 : ; 0.74 04/15/81 Recognize I/O devices with file names
13 : ; 0.75 04/17/81 Improve and correct buffer handling
14 : ; 0.76 04/23/81 Correct directory size when not 2^N entries
15 : ; 0.80 04/27/81 Add console input without echo, Functions 7 & 8
16 : ; 1.00 04/28/81 Renumber for general release
17 : ; 1.01 05/12/81 Fix bug in 'STORE'
18 : ; 1.10 07/21/81 Fatal error trapping, NUL device, hidden files, date & time,
19 : ;               RENAME fix, general cleanup
```

```
20 : ; 1.11 09/03/81 Don't set CURRENT BLOCK to 0 on open; fix SET FILE SIZE
21 : ; 1.12 10/09/81 Zero high half of CURRENT BLOCK after all (CP/M programs don't)
22 : ; 1.13 10/29/81 Fix classic "no write-through" error in buffer handling
23 : ; 1.20 12/31/81 Add time to FCB; separate FAT from DPT; Kill SMALLDIR;
24 : ;               Add FLUSH and MAPDEV calls; allow disk mapping in DSKCHG;
25 : ;               Lots of smaller improvements
26 : ; 1.21 01/06/82 HIGHMEM switch to run DOS in high memory
27 : ; 1.22 01/12/82 Add VERIFY system call to enable/disable verify after write
28 : ; 1.23 02/11/82 Add defaulting to parser; use variable escape character
29 : ;               Don't zero extent field in IBM version (back to 1.01!)
30 : ; 1.24 03/01/82 Restore fcn. 27 to 1.0 level; add fcn. 28
31 : ; 1.25 03/03/82 Put marker (00) at end of directory to speed searches
32 : ;
33 : ; ************************************************************
```

La zone des données du DOS se trouve dans le segment DATA, dont nous ne reproduisons que les premières lignes :

```
3674 : DATA    SEGMENT WORD
3675 : ; Init code overlaps with data area below
3676 :
3677 :         ORG     0
3678 : INBUF   DB      128 DUP (?)
3679 : CONBUF  DB      131 DUP (?)             ;The rest of INBUF and console buffer
3680 : LASTENT DW      ?
3681 : EXITHOLD DB     4 DUP (?)
3682 : FATBASE DW      ?
3683 : NAME1   DB      11 DUP (?)              ;File name buffer
3684 : ATTRIB  DB      ?
3685 : NAME2   DB      11 DUP (?)
3686 : NAME3   DB      12 DUP (?)
3687 : EXTFCB  DB      ?
3688 : ;WARNING - the following two items are accessed as a word
3689 : CREATING DB     ?
3690 : DELALL  DB      ?
3691 : TEMP    LABEL   WORD
3692 : SPSAVE  DW      ?
3693 : SSSAVE  DW      ?
```

comprenant en particulier le tampon du clavier et les variables TEMP, SPSAVE et SSSAVE :

```
3633 : ;***** DATA AREA *****
3634 : CONSTANTS       SEGMENT BYTE
3635 :         ORG     0
3636 : CONSTRT EQU     $               ;Start of constants segment
3637 :
3638 : IONAME:
3639 :         IF      NOT IBM
3640 :         DB      "PRN ","LST ","NUL ","AUX ","CON "
3641 :         ENDIF
3642 :         IF      IBM
3643 :         DB      "COM1","PRN ","LPT1","NUL ","AUX ","CON "
3644 :         ENDIF
3645 : DIVMES  DB      13,10,"Divide overflow",13,10,"$"
3646 : CARPOS  DB      0
3647 : STARTPOS DB     0
3648 : PFLAG   DB      0
3649 : DIRTYDIR DB     0                ;Dirty buffer flag
3650 : NUMDRV  DB      0                ;Number of drives
3651 : NUMIO   DB      ?                ;Number of disk tables
3652 : VERFLG  DB      0                ;Initialize with verify off
3653 : CONTPOS DW      0
```

```
3654 : DMAADD  DW      80H                ;User's disk transfer address (disp/seg)
3655 :         DW      ?
3656 : ENDMEM  DW      ?
3657 : MAXSEC  DW      0
3658 : BUFFER  DW      ?
3659 : BUFSECNO DW     0
3660 : BUFDRVNO DB     -1
3661 : DIRTYBUF DB     0
3662 : BUFDRVBP DW     ?
3663 : DIRBUFID DW     -1
3664 : DAY     DB      0
3665 : MONTH   DB      0
3666 : YEAR    DW      0
3667 : DAYCNT  DW      -1
3668 : WEEKDAY DB      0
3669 : CURDRV  DB      0                  ;Default to drive A
3670 : DRVTAB  DW      0                  ;Address of start of DPBs
3671 : DOSLEN  EQU     CODSIZ+($-CONSTRT)  ;Size of CODE + CONSTANTS segments
3672 : CONSTANTS       ENDS
```

que nous commenterons au fur et à mesure.

3.3.2 Initialisation du noyau du DOS

Le rôle de cette initialisation est précisé à la page B-2 du manuel : MS-DOS initialise ses tables internes, détermine les emplacements en mémoire centrale des copies de ses FAT (1 par lecteur de disquette), du répertoire et des tampons de données, initialise les vecteurs d'interruption pour les interruptions `20h` à `27h`, commence à construire un PSP (*Program Segment Prefix*) pour le programme COMMAND.COM au segment disponible d'adresse la plus basse possible puis revient à IO.COM.

Le code commence tout naturellement à l'adresse 0 (et non à l'adresse `100h`, contrairement aux exécutables de MS-DOS, car il n'a pas de PSP ; on ne sait même pas ce que c'est à ce moment-là) :

```
211 : ; Start of code
212 :
213 : CODE    SEGMENT
214 : ASSUME  CS:DOSGROUP,DS:DOSGROUP,ES:DOSGROUP,SS:DOSGROUP
215 :
216 :         ORG     0
217 : CODSTRT EQU     $
218 :         JMP     DOSINIT
```

Il est constitué d'un saut au code d'initialisation (lignes 211–218), repéré par l'étiquette DOSINIT, qui sera plus tard recouvert par la zone des données du DOS et par le processeur de commande lorsque l'initialisation aura été effectuée :

```
3730 : ;Init code below overlaps with data area above
3731 :
3732 :          ORG     0
3733 :
3734 : MOVFAT:
3735 : ;This section of code is safe from being overwritten by block move
3736 :          REP     MOVS    BYTE PTR [DI],[SI]
3737 :          CLD
3738 :          MOV     ES:[DMAADD+2],DX
3739 :          MOV     SI,[DRVTAB]     ;Address of first DPB
3740 :          MOV     AL,-1
3741 :          MOV     CL,[NUMIO]      ;Number of DPBs
3742 : FLGFAT:
3743 :          MOV     DI,ES:[SI.FAT]  ;get pointer to FAT
3744 :          DEC     DI              ;Point to dirty byte
3745 :          STOSB                   ;Flag as unused
3746 :          ADD     SI,DPBSIZ       ;Point to next DPB
3747 :          LOOP    FLGFAT
3748 :          MOV     AX,[ENDMEM]
3749 :          CALL    SETMEM          ;Set up segment
3750 :
3751 : XXX     PROC FAR
3752 :          RET
3753 : XXX     ENDP
3754 :
3755 : DOSINIT:
3756 :          CLI
3757 :          CLD
3758 :          PUSH    CS
3759 :          POP     ES
3760 :          MOV     ES:[ENDMEM],DX
3761 :          LODSB                   ;Get no. of drives & no. of I/O drivers
3762 :          MOV     ES:[NUMIO],AL
3763 :          MOV     DI,OFFSET DOSGROUP:MEMSTRT
3764 : PERDRV:
```

```
3765 :          MOV     BP,DI
3766 :          MOV     AL,ES:[DRVCNT]
3767 :          STOSB           ;DEVNUM
3768 :          LODSB           ;Physical unit no.
3769 :          STOSB           ;DRVNUM
3770 :          CMP     AL,15
3771 :          JA      BADINIT
3772 :          CBW             ;Index into FAT size table
3773 :          SHL     AX,1
3774 :          ADD     AX,OFFSET DOSGROUP:FATSIZTAB
3775 :          XCHG    BX,AX
3776 :          LODSW           ;Pointer to DPT
3777 :          PUSH    SI
3778 :          MOV     SI,AX
3779 :          LODSW
3780 :          STOSW           ;SECSIZ
3781 :          MOV     DX,AX
3782 :          CMP     AX,ES:[MAXSEC]
3783 :          JBE     NOTMAX
3784 :          MOV     ES:[MAXSEC],AX
3785 : NOTMAX:
3786 :          LODSB
3787 :          DEC     AL
3788 :          STOSB           ;CLUSMSK
3789 :          JZ      HAVSHFT
3790 :          CBW
3791 : FIGSHFT:
3792 :          INC     AH
3793 :          SAR     AL,1
3794 :          JNZ     FIGSHFT
3795 :          MOV     AL,AH
3796 : HAVSHFT:
3797 :          STOSB           ;CLUSSHFT
3798 :          MOVSW           ;FIRFAT (= number of reserved sectors)
3799 :          MOVSB           ;FATCNT
3800 :          MOVSW           ;MAXENT
3801 :          MOV     AX,DX           ;SECSIZ again
3802 :          MOV     CL,5
3803 :          SHR     AX,CL
3804 :          MOV     CX,AX           ;Directory entries per sector
3805 :          DEC     AX
3806 :          ADD     AX,ES:[BP.MAXENT]
3807 :          XOR     DX,DX
3808 :          DIV     CX
3809 :          STOSW           ;DIRSEC (temporarily)
3810 :          MOVSW                   ;DSKSIZ (temporarily)
3811 : FNDFATSIZ:
3812 :          MOV     AL,1
3813 :          MOV     DX,1
3814 : GETFATSIZ:
3815 :          PUSH    DX
3816 :          CALL    FIGFATSIZ
3817 :          POP     DX
3818 :          CMP     AL,DL           ;Compare newly computed FAT size with trial
3819 :          JZ      HAVFATSIZ       ;Has sequence converged?
3820 :          CMP     AL,DH           ;Compare with previous trial
3821 :          MOV     DH,DL
3822 :          MOV     DL,AL           ;Shuffle trials
3823 :          JNZ     GETFATSIZ       ;Continue iterations if not oscillating
3824 :          DEC     WORD PTR ES:[BP.DSKSIZ] ;Damp those oscillations
3825 :          JMP     SHORT FNDFATSIZ ;Try again
3826 :
```

```
3827 : BADINIT:
3828 :          MOV     SI,OFFSET DOSGROUP:BADMES
3829 :          CALL    OUTMES
3830 :          STI
3831 :          HLT
3832 :
3833 : HAVFATSIZ:
3834 :          STOSB                   ;FATSIZ
3835 :          MUL     ES:BYTE PTR[BP.FATCNT]  ;Space occupied by all FATs
3836 :          ADD     AX,ES:[BP.FIRFAT]
3837 :          STOSW                   ;FIRDIR
3838 :          ADD     AX,ES:[BP.DIRSEC]
3839 :          MOV     ES:[BP.FIRREC],AX       ;Destroys DIRSEC
3840 :          CALL    FIGMAX
3841 :          MOV     ES:[BP.MAXCLUS],CX
3842 :          MOV     AX,BX           ;Pointer into FAT size table
3843 :          STOSW                   ;Allocate space for FAT pointer
3844 :          MOV     AL,ES:[BP.FATSIZ]
3845 :          XOR     AH,AH
3846 :          MUL     ES:[BP.SECSIZ]
3847 :          CMP     AX,ES:[BX]      ;Bigger than already allocated
3848 :          JBE     SMFAT
3849 :          MOV     ES:[BX],AX
3850 : SMFAT:
3851 :          POP     SI              ;Restore pointer to init. table
3852 :          MOV     AL,ES:[DRVCNT]
3853 :          INC     AL
3854 :          MOV     ES:[DRVCNT],AL
3855 :          CMP     AL,ES:[NUMIO]
3856 :          JAE     CONTINIT
3857 :          JMP     PERDRV
3858 :
3859 : BADINITJ:
3860 :          JMP     BADINIT
3861 :
3862 : CONTINIT:
3863 :          PUSH    CS
3864 :          POP     DS
3865 : ;Calculate true address of buffers, FATs, free space
3866 :          MOV     BP,[MAXSEC]
3867 :          MOV     AX,OFFSET DOSGROUP:DIRBUF
3868 :          ADD     AX,BP
3869 :          MOV     [BUFFER],AX     ;Start of buffer
3870 :          ADD     AX,BP
3871 :          MOV     [DRVTAB],AX     ;Start of DPBs
3872 :          SHL     BP,1            ;Two sectors - directory and buffer
3873 :          ADD     BP,DI           ;Allocate buffer space
3874 :          ADD     BP,ADJFAC       ;True address of FATs
3875 :          PUSH    BP
3876 :          MOV     SI,OFFSET DOSGROUP:FATSIZTAB
3877 :          MOV     DI,SI
3878 :          MOV     CX,16
3879 : TOTFATSIZ:
3880 :          INC     BP              ;Add one for Dirty byte
3881 :          INC     BP              ;Add one for I/O device number
3882 :          LODSW                   ;Get size of this FAT
3883 :          XCHG    AX,BP
3884 :          STOSW                   ;Save address of this FAT
3885 :          ADD     BP,AX           ;Compute size of next FAT
3886 :          CMP     AX,BP           ;If size was zero done
3887 :          LOOPNZ  TOTFATSIZ
3888 :          MOV     AL,15
```

```
3889 :          SUB     AL,CL           ;Compute number of FATs used
3890 :          MOV     [NUMDRV],AL
3891 :          XOR     AX,AX           ;Set zero flag
3892 :          REPZ    SCASW           ;Make sure all other entries are zero
3893 :          JNZ     BADINITJ
3894 :          ADD     BP,15           ;True start of free space
3895 :          MOV     CL,4
3896 :          SHR     BP,CL           ;First free segment
3897 :          MOV     DX,CS
3898 :          ADD     DX,BP
3899 :          MOV     BX,0FH
3900 :          MOV     CX,[ENDMEM]
3901 :          CMP     CX,1            ;Use memory scan?
3902 :          JNZ     SETEND
3903 :          MOV     CX,DX           ;Start scanning just after DOS
3904 : MEMSCAN:
3905 :          INC     CX
3906 :          JZ      SETEND
3907 :          MOV     DS,CX
3908 :          MOV     AL,[BX]
3909 :          NOT     AL
3910 :          MOV     [BX],AL
3911 :          CMP     AL,[BX]
3912 :          NOT     AL
3913 :          MOV     [BX],AL
3914 :          JZ      MEMSCAN
3915 : SETEND:
3916 :          IF      HIGHMEM
3917 :          SUB     CX,BP
3918 :          MOV     BP,CX           ;Segment of DOS
3919 :          MOV     DX,CS           ;Program segment
3920 :          ENDIF
3921 :          IF      NOT HIGHMEM
3922 :          MOV     BP,CS
3923 :          ENDIF
3924 : ; BP has segment of DOS (whether to load high or run in place)
3925 : ; DX has program segment (whether after DOS or overlaying DOS)
3926 : ; CX has size of memory in paragraphs (reduced by DOS size if HIGHMEM)
3927 :          MOV     CS:[ENDMEM],CX
3928 :          IF      HIGHMEM
3929 :          MOV     ES,BP
3930 :          XOR     SI,SI
3931 :          MOV     DI,SI
3932 :          MOV     CX,(DOSLEN+1)/2
3933 :          PUSH    CS
3934 :          POP     DS
3935 :          REP MOVSW               ;Move DOS to high memory
3936 :          ENDIF
3937 :          XOR     AX,AX
3938 :          MOV     DS,AX
3939 :          MOV     ES,AX
3940 :          MOV     DI,INTBASE
3941 :          MOV     AX,OFFSET DOSGROUP:QUIT
3942 :          STOSW                   ;Set abort address--displacement
3943 :          MOV     AX,BP
3944 :          MOV     BYTE PTR DS:[ENTRYPOINT],LONGJUMP
3945 :          MOV     WORD PTR DS:[ENTRYPOINT+1],OFFSET DOSGROUP:ENTRY
3946 :          MOV     WORD PTR DS:[ENTRYPOINT+3],AX
3947 :          MOV     WORD PTR DS:[0],OFFSET DOSGROUP:DIVOV   ;Set default divide trap address
3948 :          MOV     DS:[2],AX
3949 :          MOV     CX,9
3950 :          REP STOSW               ;Set 5 segments (skip 2 between each)
```

```
3951 :          MOV     WORD PTR DS:[INTBASE+4],OFFSET DOSGROUP:COMMAND
3952 :          MOV     WORD PTR DS:[INTBASE+12],OFFSET DOSGROUP:IRET   ;Ctrl-C exit
3953 :          MOV     WORD PTR DS:[INTBASE+16],OFFSET DOSGROUP:IRET   ;Fatal error exit
3954 :          MOV     AX,OFFSET BIOSREAD
3955 :          STOSW
3956 :          MOV     AX,BIOSSEG
3957 :          STOSW
3958 :          STOSW                   ;Add 2 to DI
3959 :          STOSW
3960 :          MOV     WORD PTR DS:[INTBASE+18H],OFFSET BIOSWRITE
3961 :          MOV     WORD PTR DS:[EXIT],100H
3962 :          MOV     WORD PTR DS:[EXIT+2],DX
3963 :          IF      NOT IBM
3964 :          MOV     SI,OFFSET DOSGROUP:HEADER
3965 :          CALL    OUTMES
3966 :          ENDIF
3967 :          PUSH    CS
3968 :          POP     DS
3969 :          PUSH    CS
3970 :          POP     ES
3971 : ;Move the FATs into position
3972 :          MOV     AL,[NUMIO]
3973 :          CBW
3974 :          XCHG    AX,CX
3975 :          MOV     DI,OFFSET DOSGROUP:MEMSTRT.FAT
3976 : FATPOINT:
3977 :          MOV     SI,WORD PTR [DI]        ;Get address within FAT address table
3978 :          MOVSW                           ;Set address of this FAT
3979 :          ADD     DI,DPBSIZ-2             ;Point to next DPB
3980 :          LOOP    FATPOINT
3981 :          POP     CX                      ;True address of first FAT
3982 :          MOV     SI,OFFSET DOSGROUP:MEMSTRT      ;Place to move DPBs from
3983 :          MOV     DI,[DRVTAB]             ;Place to move DPBs to
3984 :          SUB     CX,DI                   ;Total length of DPBs
3985 :          CMP     DI,SI
3986 :          JBE     MOVJMP                  ;Are we moving to higher or lower memory?
3987 :          DEC     CX                      ;Move backwards to higher memory
3988 :          ADD     DI,CX
3989 :          ADD     SI,CX
3990 :          INC     CX
3991 :          STD
3992 : MOVJMP:
3993 :          MOV     ES,BP
3994 :          JMP     MOVFAT
3995 :
3996 : FIGFATSIZ:
3997 :          MUL     ES:BYTE PTR[BP.FATCNT]
3998 :          ADD     AX,ES:[BP.FIRFAT]
3999 :          ADD     AX,ES:[BP.DIRSEC]
4000 : FIGMAX:
4001 : ;AX has equivalent of FIRREC
4002 :          SUB     AX,ES:[BP.DSKSIZ]
4003 :          NEG     AX
4004 :          MOV     CL,ES:[BP.CLUSSHFT]
4005 :          SHR     AX,CL
4006 :          INC     AX
4007 :          MOV     CX,AX           ;MAXCLUS
4008 :          INC     AX
4009 :          MOV     DX,AX
4010 :          SHR     DX,1
4011 :          ADC     AX,DX           ;Size of FAT in bytes
4012 :          MOV     SI,ES:[BP.SECSIZ]
```

```
4013 :          ADD     AX,SI
4014 :          DEC     AX
4015 :          XOR     DX,DX
4016 :          DIV     SI
4017 :          RET
```

Initialisation des tables internes

Commentaires.- 1^o) On inhibe les interruptions masquable, les registres `DI` et `SI` seront incrémentés dans l'ordre croissant lors des opérations sur les chaînes de caractères et le segment supplémentaire est confondu avec le segment de code (lignes 3756–3759).

On sauvegarde la valeur du registre `DX` dans la variable ENDMEM (ligne 3760). Rappelons que IO.COM a initialisé cette variable à 1 avant d'appeler MSDOS.COM.

La variable ENDMEM est déclarée ligne 3656 :

```
3656 : ENDMEM  DW      ?
```

On place le nombre de lecteurs de disquette, valeur du premier octet de INITTAB, dans la variable NUMIO de la zone des données de MS-DOS (lignes 3761–3762).

Rappelons que IO.COM a fait pointer `SI` sur INITTAB avant d'appeler MSDOS.COM.

La variable NUMIO contient le nombre de lecteurs de disquettes :

```
3651 : NUMIO   DB      ?       ;Number of disk tables
```

Initialisation du bloc des paramètres (DPB) du premier lecteur de disquette

2^o) On fait pointer (lignes 3763 et 3765) `BP` sur l'étiquette MEMSTRT, repérant à la fois la fin de la copie de MSDOS.COM en mémoire centrale et le début du bloc des paramètres (DPB) du lecteur de disquette par défaut, qui suit donc le code de MS-DOS :

```
4027 : MEMSTRT LABEL   WORD
```

— On place comme premier champ, et premier octet, de ce bloc des paramètres le numéro du lecteur de disquette (lignes 3766 et 3767).

La variable DRVCNT contient le numéro du lecteur de disquette, initialisée à zéro pour le lecteur A :

```
4025 : DRVCNT  DB      0
```

— On place dans le second champ, et second octet, de ce bloc des paramètres le numéro physique du lecteur de disquette (lignes 3768–3769).

Si ce numéro est plus grand que 15, on a une erreur puisque MS-DOS n'a prévu d'emplacements que pour 16 lecteurs de disquette. On va donc (lignes 3770 et 3771) à la partie de code repérée par l'étiquette BADINIT (lignes 3827–2831) pour afficher le message d'erreur « INIT TABLE BAD », rétablir les interruptions masquables et arrêter le système.

Le message d'erreur est BADMES :

```
4019 : BADMES:
4020 :          DB      13,10,"INIT TABLE BAD",13,10,"$"
```

Si, au contraire, le numéro physique du lecteur de disquette est inférieur à 15, on le place en tant que mot dans `AX` (ligne 3772), on le multiplie par 2 (ligne 3773), puisque la taille tient sur un mot et non sur un octet, et on lui ajoute la taille obtenue à partir du tableau FATSIZ (ligne 3774).

La taille est spécifiée par le tableau FATSIZTAB :

```
4022 : FATSIZTAB:
4023 :          DW      0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0
```

On échange les valeurs des registres `AX` et `BX` (ligne 3775) et on charge le mot d'adresse `DS:SI` dans l'accumulateur (ligne 3776). On sauvegarde sur la pile la valeur de `SI` (ligne 3777) et on le fait pointer sur la DPT (ligne 3778–3780), la taille d'un secteur.

— On place comme troisième champ, de taille un mot, du DPB la taille d'un secteur de la disquette (lignes 3779–3780).

Si cette taille de secteur est plus grande que celle utilisée jusqu'à maintenant, on la place (lignes 3781–3785) dans la variable MAXSEC de la zone des données :

```
3657 : MAXSEC  DW      0
```

— On décrémente l'octet récupéré suivant, le masque d'unité d'allocation (lignes 3786–3788). S'il est non nul, on le convertit en un mot, on incrémente `AH`, on effectue une rotation des bits vers la droite de `AL` en réinsérant le bit dans l'indicateur de retenue jusqu'au premier bit nul trouvé, on place le contenu de `AH` dans `AL` et on place l'octet ainsi déterminé comme quatrième champ du DPB (lignes 3789–3797).

— On renseigne comme sixième champ du DPB le numéro du premier secteur des FAT (ligne 3798).

— On renseigne le nombre de copies de la FAT, toujours 2 pour MS-DOS (ligne 3799).

— On renseigne le nombre d'entrées du répertoire (ligne 3800).

— On restaure la taille d'un secteur dans `AX`, que l'on divise par 32 (taille d'une entrée de répertoire) pour déterminer le nombre d'entrées de répertoire contenues dans un secteur, que l'on place dans `CX`, ... Le numéro du premier secteur du répertoire plus le nombre de secteurs occupés par le répertoire donne le numéro du premier secteur de la première unité d'allocation, que l'on place provisoirement comme champ MAXCUS du DPB et la taille du disque (lignes 3801–3809).

— On place 1 dans `AL` et dans `DX`, on sauvegarde cette dernière valeur sur la pile et on appelle la routine FIGFATSIZ, commençant ligne 3996, pour obtenir le nombre d'unités d'allocation de la disquette dans `CX` et le nombre de secteurs occupés par la FAT dans `AX` (lignes 3810–3816).

La routine FIGFATSIZ multiplie le nombre de secteurs occupés par une FAT par le nombre de copies de FAT, auquel on ajoute le numéro du premier secteur des FAT et le nombre de secteurs du répertoire pour obtenir le numéro du premier secteur de la première unité d'allocation. On lui soustrait le nombre de secteurs de la disquette, on prend son opposé pour obtenir le nombre de secteurs consacrés aux unités d'allocation. On divise ar le nombre de secteurs par unité d'allocation, pour obtenir le nombre d'unités d'allocation de la disquette, que l'on place dans `CX` (lignes 3996–4007).

La routine FIGFATSIZ continue en ajoutant un à ce nombre, qu'elle place dans `DX`, qu'elle divise par 2 et qu'elle ajoute à `AX` pour obtenir la taille de la FAT en octets. Elle place la taille d'un secteur dans `SI`, qu'elle ajoute à la taille obtenue de la FAT, qu'on décrémente, on place 0 dans `DX` pour que la division par la taille d'un secteur ne porte que sur `AX`, ce qui permet d'obtenir dans `AX` le nombre de secteurs occupés par la FAT (lignes 4008–4017).

— On compare la taille de la FAT ainsi trouvée jusqu'à ce qu'on n'oscille plus et on place alors cette taille dans le champ FATSIZ du DPB (lignes 3817–3825 et 3833–3834).

— On multiplie par le nombre de FAT et on lui ajoute le numéro du premier secteur des FAT pour obtenir le numéro du premier secteur du répertoire, avec lequel on renseigne le champ FIRDIR du DPB (lignes 3835–3837).

— On lui ajoute le nombre de secteurs du répertoire pour obtenir le numéro du premier secteur de la première unité d'allocation, ce qui permet de renseigner le champ FIRREC

du DPB, détruisant au passage le champ DIRSEC qui y avait été entreposés provisoirement (lignes 3838–3839).

— On appelle la routine FIGMAX, commençant ligne 4000, partie de la routine FIGFATSIZ étudiée ci-dessus, pour recueillir dans dans `CX` le nombre d'unités d'allocation de la disquette et renseigner le champ MAXCLUS du DPB (lignes 3840–3841).

— On récupère le numéro du premier secteur de la FAT, sauvegardé dans `BX` ligne 3775, et on renseigne le champ FAT du DPB (lignes 3842–3843).

— On place le nombre de secteurs d'une FAT dans `AX`, on le multiplie par la taille d'un secteur et on en place le résultat dans `BX` si on obtient une taille plus grande que celle qui y était placée (lignes 3844-3850).

Initialisation du bloc des paramètres (DPB) des autres lecteurs de disquette

3^o) On fait de même pour les DPB des autres lecteurs de disquette : on fait pointer `SI` au début de la table INITTAB, on incrémente le nombre de lecteurs de disquettes contenu dans la variable DRVCNT et on exécute ce que nous avons fait pour le premier lecteur de disquette, puis les suivants (lignes 3851–3857).

Détermination de l'emplacement des copies des FAT en mémoire

4^o) Une fois les DPB de chacun des lecteurs de disquette renseignés :

— On confond le segment des données avec le segment de code (lignes 3856 et 3862–3864).

— On place dans `AX` le décalage de la zone de copie en mémoire centrale d'une unité d'allocation, à laquelle on ajoute le nombre (maximum) d'octets contenus dans un secteur pour obtenir le décalage de la copie en mémoire centrale d'un secteur, que l'on place dans la variable BUFFER (lignes 3865–3869).

La variable BUFFER contient le décalage de la copie en mémoire centrale d'un secteur :

```
3658 : BUFFER  DW      ?
```

— On lui ajoute une nouvelle fois le nombre (maximum) d'octets contenus dans un secteur pour obtenir l'adresse des DPB, que l'on place dans la variable DRVTAB (lignes 3870–3871).

La variable DRVTAB contient l'adresse du début des blocs de paramètres des lecteurs de disquette :

```
3670 : DRVTAB  DW      0               ;Address of start of DPBs
```

— On multiplie le nombre (maximum) d'octets contenus dans un secteur par 2, pour obtenir la taille de la copie en mémoire centrale de deux secteurs (une pour le répertoire et une pour un secteur de données), on lui ajoute le décalage de la fin des DPB ainsi que le décalage du début de la zone des tampons pour obtenir le décalage du début des FAT, que l'on sauvegarde sur la pile (lignes 3872–3875).

La constante d'ajustement est ADJFAC :

```
4028 : ADJFAC  EQU     DIRBUF-MEMSTRT
```

— On fait pointer `SI` sur le tableau FATSIZTAB, on en place le contenu dans la zone mémoire réservée aux 16 FAT, ainsi que pour chacunes d'elles un octet indiquant si la copie de la FAT a été sauvegardée sur la disquette depuis sa dernière modification en mémoire centrale et un octet spécifiant le numéro du lecteur de disquette, on stocke l'adresse de cette copies de FAT et on calcule la taille de la FAT suivante. On quitte la boucle contrôlée lorsqu'on arrive à une taille nulle (lignes 3876–3887).

— On place 15 dans `AL`, le numéro maximum de FAT, on lui soustrait le nombre de FAT qu'il reste à renseigner pour obtenir le nombre de lecteurs de disquettes, que l'on le place dans la variable NUMDRV (lignes 3888–3890).

La variable NUMDRV spécifie le nombre de lecteurs de disquette :

```
3650 : NUMDRV  DB      0       ;Number of drives
```

— On baisse le drapeau ZF et on s'assure que toutes les autres entrées de FATSIZTAB sont nulles. Sinon on arrête le système en affichant un message d'erreur de mauvaise initialisation (lignes 3891–3893).

Analyse de la mémoire

5^o) On ajoute 15 à `BP` pour pointer au début de la zone libre de la mémoire, on place 4 dans `CL` pour multiplier par 16 pour obtenir le premier segment libre et on place ce décalage dans le segment de code dans `DX`. On place `0Fh` dans `BX`, valeur qui va servir au test de la mémoire. Si le contenu de la variable ENDMEM est 1, il faut analyser la mémoire (lignes 3894–3902).

Pour analyser la mémoire, on part de la zone réservée à MS-DOS et, pour chaque octet, on y place `0Fh`, on le complémente (lignes 3903–3914).

Déplacement de MS-DOS en mémoire haute

6^o) Nous avons vu que l'on initialise, dans IO.ASM, la constante HIGHMEM à vrai s'il faut déplacer MS-DOS à la fin de la mémoire. Dans ce cas, on fait pointer `BP` au début du segment de MS-DOS, que l'on ait à déplacer MS-DOS ou non, on donne à `DX` l'adresse du segment de code, que ce soit après MS-DOS ou pour recouvrir MS-DOS, et à `CX` la taille de la mémoire, en paragraphes, réduite à la taille de MS-DOS si on doit déplacer MS-DOS en mémoire haute (lignes 3915–3926).

On effectue alors le déplacement, mot par mot (lignes 3927–3936).

Liste des interruptions logicielles de MS-DOS

7^o) Le système d'exploitation MS-DOS se réserve 8 interruptions : les interruptions `20h` à `27h`. Leurs vecteurs sont rappelés en commentaire :

```
36 : ; Interrupt Entry Points:
37 :
38 : ; INTBASE:      ABORT
39 : ; INTBASE+4:    COMMAND
40 : ; INTBASE+8:    BASE EXIT ADDRESS
41 : ; INTBASE+C:    CONTROL-C ABORT
42 : ; INTBASE+10H:  FATAL ERROR ABORT
43 : ; INTBASE+14H:  BIOS DISK READ
44 : ; INTBASE+18H:  BIOS DISK WRITE
45 : ; INTBASE+40H:  Long jump to CALL entry point
```

On a donc le vecteur de l'interruption `20h`, de terminaison d'un programme, repérée par l'étiquette ABORT, puis celui de l'interruption principale `21h`, avec de nombreuses fonctions, repérée par l'étiquette COMMAND (à ne pas confondre avec l'interpréteur de commandes COMMAND.COM), celui de l'interruption `22h`, qui n'est pas un vecteur d'interruption mais l'adresse à laquelle on tranfère le contrôle lorsqu'on quitte un programme, celui de l'interruption `23h`, appelée par CTRL-C ou CTRL-Break, celui de l'interruption `24h`, appelée lors d'une erreur critique, celui de l'interruption `25h`, de lecture sur la disquette, et enfin celui de l'interruption `26h`, d'écriture sur la disquette.

Il y a en plus un saut long à l'emplacement du vecteur d'interruption de l'interruption `30h`.

L'adresse de base des vecteurs d'interruptions de MS-DOS est donnée par la constante INTBASE, égale à 80h :

```
65 : INTBASE EQU     80H
```

En effet, pour un microprocesseur *Intel*, les 256 vecteurs d'interruptions occupent le premier kiO de la mémoire ; chaque vecteur d'interruption occupe 4 octets ; le vecteur d'interruption pour l'interruption `20h` se trouve donc à l'adresse absolue `80h`.

Initialisation des vecteurs d'interruption et du saut inconditionnel

8^o) Cela s'effectue de la façon suivante :

— On initialise les registres `DS` et `ES` à 0 (lignes 3937–3939), pour parler en termes d'adresses absolues, et on fait pointer `DI` sur INTBASE (ligne 3940), adresse du premier vecteur d'interruption du DOS.

— On renseigne le vecteur d'interruption de l'interruption `20h` : il s'agit de l'adresse repérée par l'étiquette QUIT, qui se trouve à la ligne 266 du fichier source, comme nous le verrons (lignes 3941 et 3942).

Les segments CODE, CONSTANTS, DATA, DOSGROUP et SEGBIOS sont définis vers le début du fichier source :

```
148 : ;The following are all of the segments used
149 : ;They are declared in the order that they should be placed in the executable
150 :
151 : CODE    SEGMENT
152 : CODE    ENDS
153 :
154 : CONSTANTS       SEGMENT BYTE
155 : CONSTANTS       ENDS
156 :
157 : DATA    SEGMENT WORD
158 : DATA    ENDS
159 :
160 : DOSGROUP        GROUP   CODE,CONSTANTS,DATA
161 :
162 : SEGBIOS SEGMENT
163 : SEGBIOS ENDS
```

— Conformément au commentaire précédent, on place (ligne 3944) à l'adresse absolue `C0h` l'opcode pour un saut inconditionnel lointain direct, tenant sur un octet, puisqu'il s'agit de `EAh`. Il devra évidemment être suivi de quatre octets : IP bas, IP haut, CS bas et CS haut.

La constante ENTRYPOINT vaut INTBASE + `40h`, soit l'adresse absolue `C0h` :

```
68 : ENTRYPOINT      EQU     INTBASE+40H
```

La constante LONGJUMP vaut `0EAh`, soit l'opcode du saut inconditionnel lointain direct.

```
71 : LONGJUMP        EQU     0EAH
```

— Cet opcode doit être suivi du décalage et de l'adresse de segment. On commence par placer (ligne 3947) le décalage du début (ou presque) de la routine de service de l'interruption `21h` après cet opcode.

L'étiquette ENTRY repère le code de la routine de service de cette interruption :

```
277 : ENTRY:  ;System call entry point and dispatcher
```

— On place ensuite (lignes 3943 et 3946) la valeur du segment de code.

Les cinq octets commençant à l'adresse ENTRYPOINT constituent donc un saut à la routine de service de l'interruption `21h`.

— La routine de service du BIOS concernant l'interruption `00h`, division par zéro, est surchargée par MS-DOS (ligne 3947).

Cette routine de service, repérée par l'étiquette DIVOV, commence ligne 3619. Nous l'étudierons dans le chapitre consacré à l'implémentation des routines de service.

— On place, en une seule fois, l'adresse du segment de code comme mot de poids fort des 5 vecteurs d'interruption des interruptions `21h` à `26h` (lignes 3948–3950).

— On place le décalage de la routine de service de l'interruption principale `21h` comme mot de poids faible du vecteur d'interruption correspondant (ligne 3951).

Cette routine commence à l'étiquette COMMAND, repérant la ligne 270, un petit peu avant ENTRY.

— Momentanément les routines de service des interruptions `23h` et `24h` ne font rien. Elles seront surchargées par l'interpréteur de commande COMMAND.COM.

L'étiquette IRET repère une instruction IRET (*sic*) :

```
275 : IRET:   IRET
```

— On place le décalage de la routine de service de l'interruption `25h` comme vecteur d'interruption correspondant (lignes 3954–3959), seulement le décalage puisque l'adresse de segment a déjà été positionnée.

Le décalage de cette routine de service est celui de la pré-routine BIOSREAD :

```
184 : BIOSREAD        DB      3 DUP (?)       ;Disk read
```

— On place le décalage de la routine de service de l'interruption `26h` comme mot de poids faible du vecteur d'interruption correspondant (ligne 3960).

Ce décalage est celui de la pré-routine BIOSWRITE :

```
185 : BIOSWRITE       DB      3 DUP (?)       ;Disk write
```

— On termine par l'interruption `22h` (lignes 3961 et 3962).

La variable EXIT est initialisée à INTBASE + 8, puisqu'elle correspond au vecteur d'interruption de l'interruption `22h`, soit celui de l'interruption `20h` + 2×4 :

```
70 : EXIT    EQU     INTBASE+8
```

Affichage du message d'accueil

9^o) On affiche ensuite le message d'accueil (lignes 3963–3966).

HEADER repère le message affiché lors du démarrage de MS-DOS :

```
253 :         IF      NOT IBM
254 : HEADER  DB      13,10,"MS-DOS version 1.25"
255 :         IF      HIGHMEM
256 :         DB      "H"
257 :         ENDIF
258 :         IF      DSKTEST
259 :         DB      "D"
260 :         ENDIF
261 :
262 :         DB      13,10
263 :         DB      "Copyright 1981,82 Microsoft, Inc.",13,10,"$"
264 :         ENDIF
```

Préparation de l'emplacement des copies des FAT en mémoire

10^o) On confond les segments des données et supplémentaire avec le segment de code, on place le numéro de lecteur de disquette par défaut dans `AX`, que l'on sauvegarde dans `CX`, on fait pointer `DI` sur l'emplacement actuel de la copie de la FAT, `SI` sur la FAT dans la table des adresses des FAT, que l'on reporte et de même pour les autres FAT (lignes 3967–3980).

On récupère sur la pile le décalage de l'adresse de la copie en mémoire centrale de la première FAT, que l'on place dans `CX`, on fait pointer `SI` sur la zone actuelle des tables, `DI` sur la table

des blocs des paramètres des lecteurs de disquette et on place dans `CX` la taille de cette dernière table. Si on doit déplacer les tables en mémoire haute, on ajoute cette taille à `DI` et à `SI` et l'incrémentation sur les chaînes de caractères s'effectuera dans l'ordre croissant. Le segment supplémentaire est confondu avec le segment de code et on va à la partie MOVFAT, située au tout début du code, pour effectuer le déplacement (lignes 3981–3994).

La partie MOVFAT effectue le déplacement puis passe à la décrémentation des indices. On fait pointer `SI` sur le premier bloc des paramètres de disquette, on place `FFh` dans `AL`, le nombre de blocs de paramètres de disquette dans `CL`, on fait pointer `DI` sur l'octet précédant la copie de la première FAT et on y place `FFh` pour dire que cette copie n'a pas été sauvegardée sur la diquette depuis sa dernière modification, et de même pour les autres emplacements de FAT (lignes 3734–3748).

Construction du PSP pour COMMAND.COM

11^o) On place la valeur de la variable ENDMEM dans `AX` et on appelle la routine SETMEM, étudiée ci-dessous, pour construire un PSP (lignes 3748–3749).

Retour à IO.COM

12^o) Le retour à IO.COM est effectué par l'instruction RET (ligne 3752).

3.3.3 La routine de construction d'un PSP

La routine SETMEM construit le PSP d'un programme au début du segment spécifié en paramètre.

Avant de l'appeler, `AX` doit contenir la taille de la mémoire libre, en paragraphes, et `DX` une adresse de segment.

Après son exécution, les adresses des segments des données et supplémentaire ont comme adresse de segment celle spécifiée, et les 22 premiers octets de ce segment sont renseignés pour constituer un PSP :

```
3363 : SETMEM:
3364 :
3365 : ; Inputs:
3366 : ;       AX = Size of memory in paragraphs
3367 : ;       DX = Segment
3368 : ; Function:
3369 : ;       Completely prepares a program base at the
3370 : ;       specified segment.
3371 : ; Outputs:
3372 : ;       DS = DX
3373 : ;       ES = DX
3374 : ;       [0] has INT 20H
3375 : ;       [2] = First unavailable segment ([ENDMEM])
3376 : ;       [5] to [9] form a long call to the entry point
3377 : ;       [10] to [13] have exit address (from INT 22H)
3378 : ;       [14] to [17] have ctrl-C exit address (from INT 23H)
3379 : ;       [18] to [21] have fatal error address (from INT 24H)
3380 : ; DX,BP unchanged. All other registers destroyed.
3381 :
3382 :         XOR     CX,CX
3383 :         MOV     DS,CX
3384 :         MOV     ES,DX
3385 :         MOV     SI,EXIT
3386 :         MOV     DI,SAVEXIT
3387 :         MOVSW
3388 :         MOVSW
3389 :         MOVSW
3390 :         MOVSW
3391 :         MOVSW
3392 :         MOVSW
3393 :         MOV     ES:[2],AX
3394 :         SUB     AX,DX
3395 :         CMP     AX,MAXDIF
3396 :         JBE     HAVDIF
3397 :         MOV     AX,MAXDIF
3398 : HAVDIF:
3399 :         MOV     BX,ENTRYPOINTSEG
3400 :         SUB     BX,AX
3401 :         SHL     AX,1
3402 :         SHL     AX,1
3403 :         SHL     AX,1
3404 :         SHL     AX,1
3405 :         MOV     DS,DX
3406 :         MOV     DS:[6],AX
3407 :         MOV     DS:[8],BX
3408 :         MOV     DS:[0],20CDH    ;"INT INTTAB"
3409 :         MOV     DS:(BYTE PTR [5]),LONGCALL
3410 :         RET
```

— On initialise les adresses des segments des données et supplémentaire à 0, de façon à ce que les décalages correspondent aux adresses absolues, on fait pointer `SI` sur le vecteur

d'interruption de l'interruption INT 22h, on place 10 dans DI, contenu de la constante SAVEXIT, et on copie les 6 mots correspondants, donc les vecteurs des interruptions INT 22h, INT 23h et INT 24h sur les octets 0Ah à 15h du PSP (lignes 3382–3392).

La constante SAVEXIT donne le décalage dans un PSP de l'adresse de terminaison du programme :

```
74 : SAVEXIT EQU     10
```

— On place le contenu de AX, taille de la mémoire libre, sur le mot d'adresse 02h du PSP, c'est-à-dire l'adresse de la zone de mémoire allouée au programme (ligne 3393).

— On soustrait le contenu de DX à AX, pour obtenir la taille de la mémoire restante et on ne garde que le minimum entre la valeur trouvée et FFFh (lignes 3394–3398).

Le maximum de la différence est donné par la constante MAXDIF :

```
73 : MAXDIF  EQU     0FFFH
```

— On place 0Ch dans BX auquel on soustrait la taille que l'on vient de calculer, on divise cette taille par 16, pour l'obtenir en paragraphes, que l'on place comme mot de poids faible de l'adresse de l'appel long, le contenu de BX comme mot de poids fort (lignes 3399–3407).

La constante ENTRYPOINTSEG est définie ligne 67 :

```
67 : ENTRYPOINTSEG   EQU     0CH
```

— On place l'instruction INT 20h comme premier mot du PSP (ligne 3408).

— On place le préfixe d'une instruction d'appel long comme cinquième octet du PSP (ligne 3409).

Le code opération d'une instruction d'appel long est donné par la constante LONGCALL 72 :

```
72 : LONGCALL EQU    9AH
```

3.4 Appel de COMMAND.COM

Lorsqu'on revient de MSDOS.COM, on revient donc à la ligne 192 du code source de IO.ASM :

```
192 : ;
193 : ; Change disk read and write vectors (INT 37 and INT 38) to go to
194 : ; DIRECTREAD and DIRECTWRITE rather than READ and WRITE.
195 : ;
196 :       SUB   BP,BP
197 :       MOV   W,[BP+37*4],DIRECTREAD
198 :       MOV   W,[BP+38*4],DIRECTWRITE
199 :
200 :       MOV   DX,100H
201 :       MOV   AH,26       ;Set DMA address
202 :       INT   33
203 :       MOV   CX,[6]      ;Get size of segment
204 :       MOV   BX,DS       ;Save segment for later
205 : ;
206 : ; DS must be set to CS so we can point to the FCB.
207 : ;
208 :       MOV   AX,CS
209 :       MOV   DS,AX
210 :       MOV   DX,FCB      ;File Control Block for COMMAND.COM
211 :       MOV   AH,15
212 :       INT   33          ;Open COMMAND.COM
213 :       OR    AL,AL
214 :       JNZ   COMERR      ;Error if file not found
215 :       XOR   AX,AX
216 :       MOV   [FCB+33],AX ; Set 4-byte Random Record field to
217 :       MOV   [FCB+35],AX ;  beginning of file.
218 :       INC   AX
219 :       MOV   [FCB+14],AX ;Set record length field
220 :       MOV   AH,39       ;Block read (CX already set)
221 :       INT   33
222 :       JCXZ  COMERR      ;Error if no records read
223 :       TEST  AL,1
224 :       JZ    COMERR      ;Error if not end-of-file
225 : ;
226 : ; Make all segment registers the same.
227 : ;
228 :       MOV   DS,BX
229 :       MOV   ES,BX
230 :       MOV   SS,BX
231 :       MOV   SP,5CH      ;Set stack to standard value
232 :       XOR   AX,AX
233 :       PUSH  AX          ;Put zero on top of stack for return
234 :       MOV   DX,80H
235 :       MOV   AH,26
236 :       INT   33          ;Set default transfer address (DS:0080)
237 :       PUSH  BX          ;Put segment on stack
238 :       MOV   AX,100H
239 :       PUSH  AX          ;Put address to execute within segment on stack
240 :       RET   L           ;Jump to COMMAND
241 :
242 : COMERR:
243 :       MOV   DX,BADCOM
244 :       MOV   AH,9        ;Print string
245 :       INT   33
246 :       EI
247 : STALL:      JP    STALL
```

Commentaires.- 1^o) On commence par remplacer les routines de service des interruptions READ (`25h`) et WRITE (`26h`) par de nouvelles routines de service, repérées par les étiquettes DIRECTREAD et DIRECTWRITE (lignes 193–198).

2^o) On initialise l'adresse de la zone de transfert à 256 (lignes 200–202).

3^o) On récupère la taille du segment dans `CX` et on sauvegarde l'adresse du segment dans `BX` (lignes 203–204).

4^o) On confond le segment des données avec le segment de code, on fait pointer `DX` sur le FCB de COMMAND.COM, et on appelle la fonction 15 de l'interruption principale de MS-DOS, INT `21h`, pour ouvrir le fichier spécifié par ce FCB, donc COMMAND.COM (lignes 205–221).

Le FCB de COMMAND.COM est tout simplement repéré par FCB :

```
263 : FCB:    DB    1,"COMMAND COM"
264 :         DS    25
```

5^o) Si on n'est pas parvenu à ouvrir ce fichier, on affiche un message d'erreur, on se remet à l'écoute des interruptions masquables et on gèle le système (lignes 213–214 et 242–247).

Le message d'erreur est le suivant :

```
262 : BADCOM: DB 13,10,"Error in loading Command Interpreter",13,10,"$"
```

6^o) Si, par contre, on est parvenu à ouvrir le fichier COMMAND.COM, on initialise à zéro les quatre octets du champ RR (*Random Record*) de son FCB, pour indiquer que la lecture s'effectuera depuis le début du fichier, à 1 le champ 'taille d'un enregistrement' du FCB, pour indiquer que la lecture s'effectuera octet par octet, et on appelle la fonction 39 de l'interruption `21h` pour lire n enregistrements, et donc n octets, n étant le contenu de `CX`.

7^o) Si on n'est parvenu à lire aucun enregistrement ou si on n'est pas arrivé à la fin du fichier COMMAND.COM, on affiche le même message d'erreur que ci-dessus, on se remet à l'écoute des interruptions masquables et on gèle le système (lignes 215–224).

8^o) Sinon on confond les segments de données, supplémentaire et de pile avec celui du segment de code, on pointe le début de la pile sur `5Ch`, qui est la valeur standard, on place 0 sur la pile pour le retour de la routine qui n'a pas été appelée, et donc qui ne possède pas d'adresse de retour sur la pile, on change l'adresse de la zone de transfert, commençant maintenant à l'adresse `DS:0080h`, on sauvegarde l'adresse du segment de code sur la pile, ainsi que l'adresse du début du programme (à savoir 256) et on termine la routine (qui n'a pas été appelée), ce qui nous amène à l'adresse 256, d'après ce que nous venons de placer dans la pile, et donc au début du programme COMMAND.COM pour le lancer (lignes 225–240).

On ne reviendra jamais à IO.COM, puisqu'on a le cycle infini : COMMAND.COM charge une commande puis l'exécute. Il n'y a donc pas besoin de code supplémentaire.

3.5 Initialisation de l'interpréteur de commandes

3.5.1 Un interpréteur de commandes en trois parties

L'interpréteur de commandes comprend trois parties :

— La **partie résidente** comprend les routines de service des interruptions `22h` (terminaison d'un processus), `23h` (appel de CTRL-C), `24h` (erreur fatale) et `27h` (terminaison d'un processus tout en laissant son code en mémoire), ainsi que le code permettant de tester si le code d'une commande transitoire se trouve en mémoire centrale et, le cas échéant, le charger.

— La **partie initialisation** est exécutée une seule fois, lors du démarrage du système d'exploitation. Elle peut donc être recouverte ensuite et elle le sera effectivement par le code de la partie transitoire.

— La **partie transitoire**, comprenant le code des commandes internes et le traitement des commandes en lignes, ne se trouve pas en permanence en mémoire centrale mais uniquement lorsqu'on en a besoin.

```
1  : ; COMMAND version 1.17
2  : ;
3  : ; This version of COMMAND is divided into three distinct parts. First
4  : ; is the resident portion, which includes handlers for interrupts
5  : ; 22H (terminate), 23H (Cntrl-C), 24H (fatal error), and 27H (stay
6  : ; resident); it also has code to test and, if necessary, reload the
7  : ; transient portion. Following the resident is the init code, which is
8  : ; overwritten after use. Then comes the transient portion, which
9  : ; includes all command processing (whether internal or external).
10 : ; The transient portion loads at the end of physical memory, and it may
11 : ; be overlayed by programs that need as much memory as possible. When
12 : ; the resident portion of command regains control from a user program,
13 : ; a checksum is performed on the transient portion to see if it must be
14 : ; reloaded. Thus programs which do not need maximum memory will save
15 : ; the time required to reload COMMAND when they terminate.
```

3.5.2 Lancement de l'interpréteur de commandes

L'exécution du code de COMMAND.COM commence à l'adresse `100h` :

```
280 : ;START OF RESIDENT PORTION
281 :
282 : CODERES SEGMENT
283 : ASSUME  CS:RESGROUP,DS:RESGROUP,ES:RESGROUP,SS:RESGROUP
284 :         ORG     0
285 : ZERO    =       $
286 : PARMBUF LABEL   WORD
287 :
288 :         ORG     100H
289 :
290 : RSTACK  LABEL   WORD
291 :
292 : PROGSTART:
293 :         JMP     CONPROC
```

Commentaires.- 1^o) Les segments, et donc le segment du groupe des segments de code et des données de la partie résidente, sont CODERES, DATARES, INIT, TAIL et RESGROUP :

```
70 : ;The following are all of the segments used in the load order
71 :
72 : CODERES SEGMENT
73 : CODERES ENDS
74 :
75 : DATARES SEGMENT BYTE
76 : DATARES ENDS
77 :
78 : INIT    SEGMENT BYTE
79 : INIT    ENDS
80 :
81 : TAIL    SEGMENT PARA
82 : TAIL    ENDS
[...]
93 : RESGROUP        GROUP   CODERES,DATARES,INIT,TAIL
```

2^o) COMMAND.COM étant un programme `.com`, son code commence au décalage 256, où l'on trouve un saut à la partie CONPROC du code, concernant la partie initialisation de l'interpréteur de commande.

3.5.3 Le code d'initialisation de l'interpréteur de commandes

3.5.3.1 Routine principale

Nous venons de voir que ce code commence à l'étiquette CONPROC :

```
515 : ;******************************************************************
516 : ;START OF INIT PORTION
517 : ;This code is overlayed the first time the TPA is used.
518 :
519 : INIT    SEGMENT BYTE
520 :
521 :         ORG     0
522 : ZERO    =       $
523 : CONPROC:
524 :         MOV     SP,OFFSET RESGROUP:RSTACK
525 :
526 :         IF      HIGHMEM
527 :         MOV     AX,WORD PTR DS:[2]
528 :         SUB     AX,((RESCODESIZE+RESDATASIZE)+15)/16            ;Subtract size of resident
529 :         MOV     WORD PTR DS:[2],AX
530 :         MOV     ES,AX
531 :         MOV     SI,100H
532 :         MOV     DI,SI
533 :         MOV     CX,((RESCODESIZE+RESDATASIZE)-100H+1)/2 ;Length of resident in words
534 :         REP     MOVSW                                           ;Move to end of memory
535 :         MOV     DS,AX
536 :         MOV     [LTPA],CS
537 :         ENDIF
538 :
539 :         IF      NOT HIGHMEM
540 :         MOV     AX,CS
541 :         ADD     AX,((RESCODESIZE+RESDATASIZE)+15)/16            ;Compute segment of TPA
542 :         MOV     [LTPA],AX
543 :         MOV     AX,WORD PTR DS:[2]
544 :         ENDIF
545 :
546 :         MOV     [MYSEG],DS
547 :         MOV     [MEMSIZ],AX
548 :         SUB     AX,TRNLEN                                       ;Subtract size of transient
549 :         MOV     [TRNSEG],AX
550 :         CALL    SETVECT
551 :         CALL    LOADCOM
552 :         CALL    CHKSUM
553 :         MOV     [SUM],DX
554 :
555 :         IF MSVER
556 :         IF      HIGHMEM
557 :         PUSH    DS
558 :         PUSH    CS
559 :         POP     DS
560 :         ENDIF
561 :         MOV     DX,OFFSET RESGROUP:HEADER
562 :         MOV     AH,PRINTBUF
563 :         INT     33
564 :         IF      HIGHMEM
565 :         POP     DS
566 :         ENDIF
567 :         ENDIF
568 :
569 :         MOV     DX,OFFSET RESGROUP:BATFCB
570 :         MOV     AH,OPEN
571 :         INT     33                                               ;See if AUTOEXEC.BAT exists
```

```
572 :          MOV     WORD PTR[BATFCB+RECLEN],1                       ;Set record length to 1
573 :          OR      AL,AL                                           ;Zero means file found
574 :          JZ      DRV0
575 :          MOV     [BATCH],0                                     ;Not found--turn off batch job
576 :          MOV     AX,OFFSET TRANGROUP:DATINIT
577 :          MOV     WORD PTR[INITADD],AX
578 :          MOV     AX,[TRNSEG]
579 :          MOV     WORD PTR[INITADD+2],AX
580 :          CALL    DWORD PTR DS:[INITADD]
581 :
582 :          IF IBMVER
583 :          MOV     DX,OFFSET RESGROUP:HEADER
584 :          MOV     AH,PRINTBUF
585 :          INT     33
586 :          ENDIF
587 :
588 : DRV0:
589 :          JMP     HAVCOM
```

Commentaires.- 1^o) La partie « initialisation » de l'interpréteur de commandes n'est nécessaire que la première fois qu'on y fait appel, c'est-à-dire lors du démarrage de MS-DOS. Elle peut donc être ensuite recouverte par autre chose, ce qui arrivera la première fois que la partie transitoire sera utilisée.

2^o) On initialise le pointeur de pile `SP` à l'adresse du segment de pile (ligne 524). On a vu que RSTACK est une étiquette (ligne 290).

3^o) Si l'interpréteur de commandes se trouve en mémoire basse, on calcule l'adresse de segment de l'emplacement de la partie transitoire de l'interpréteur de commandes, que l'on place dans la variable LPTA et on initialise `AX` avec la taille de la mémoire (lignes 539–544).

La variable LTPA contient l'adresse de segment de la partie transitoire de l'interpréteur de commandes :

```
295 : LTPA    DW      0               ;WILL STORE TPA SEGMENT HERE
```

Le choix de placer la partie résidente de l'interpréteur de commandes en-dessous ou au-dessus de sa partie transitoire, est effectué avant l'assemblage, en mettant à faux ou à vrai la constante HIGHMEM :

```
17 : ;Use the following booleans to set assembly flags
18 : FALSE   EQU     0
19 : TRUE    EQU     NOT FALSE
20 :
21 : IBMVER  EQU     FALSE   ;Switch to build IBM version of Command
22 : MSVER   EQU     TRUE    ;Switch to build MS-DOS version of Command
23 :
24 : HIGHMEM EQU     TRUE    ;Run resident part above transient (high memory)
```

La taille du code de la partie résidente de l'interpréteur de commandes est spécifiée dans la constante RESCODESIZE :

```
512 : RESCODESIZE     EQU     $-ZERO
```

l'étiquette ZERO se trouvant ligne 522.

La taille des données de la partie résidente de l'interpréteur de commandes est spécifiée par la constante RESDATASIZE :

```
149 : RESDATASIZE     EQU     $-ZERO
```

4^o) On place l'adresse du segment des données dans la variable MYSEG, la taille de la mémoire dans la variable MEMSIZ et la taille du segment transitoire dans la variable TRNSEG (lignes 546–549).

L'adresse du segment en cours est spécifiée par la variable MYSEG :

```
296 : MYSEG   DW      0               ;Put our own segment here
```

La taille de la mémoire est spécifiée par la variable MEMSIZ :

```
146 : MEMSIZ  DW      ?
```

La taille, en paragraphes, de la partie transitoire de l'interpréteur de commandes est spécifiée par la constante TRNLEN :

```
2164 : TRNLEN  EQU     (PRETRLEN+TRANCODESIZE+TRANDATASIZE+15)/16
             ;Length of transient in paragraphs
```

les constantes PRETRLEN, TRANCODESIZE et TRANDATASIZE étant définies auparavant :

```
207  : TRANDATASIZE    EQU     $-ZERO
[...]
265  : PRETRLEN        EQU     $-ZERO          ;Used later to compute TRNLEN
[...]
2161 : TRANCODESIZE    EQU     $-ZERO
```

L'adresse du segment de la partie transitoire de l'interpréteur de commandes est spécifiée par la variable TRNSEG :

```
144 : TRNSEG  DW      ?
```

5^o) On appelle la sous-routine SETVECT, étudiée ci-dessous, pour initialiser les vecteurs d'interruption des interruptions INT 22h, INT 23h, INT 24h et INT 27h (ligne 550).

6^o) On appelle la routine LOADCOM, étudiée ci-dessous, de chargement de la partie transitoire de l'interpréteur de commandes (ligne 551).

7^o) On appelle la routine CHKSUM, étudiée ci-dessous, pour calculer la somme de contrôle de la partie transitoire de l'interpréteur de commandes et la placer dans la variable SUM (lignes 552–553).

La variable SUM contient la somme de contrôle de la partie transitoire de l'interpréteur de commandes :

```
147 : SUM     DW      ?
```

8^o) On affiche le prompteur de l'interpréteur de commandes (lignes 555-567).

Le prompteur HEADER, affiché lors du démarrage de l'interpréteur de commandes, dépend de la version, PC-DOS ou MS-DOS, du système d'exploitation :

```
592 :         IF MSVER
593 : HEADER  DB      13,10,"Command v. 1.17"
594 :         IF      HIGHMEM
595 :         DB      "H"
596 :         ENDIF
597 :         DB      13,10,"$"
598 :         ENDIF
599 :
600 :         IF IBMVER
601 : HEADER  DB      13,10,13,10,"The IBM Personal Computer DOS",13,10
602 :         DB      "Version 1.10 (C)Copyright IBM Corp 1981, 1982",13,10,"$"
603 :         DB      "Licensed Material - Program Property of IBM"
604 :         ENDIF
```

9^o) Il n'y a pas nécessairement de fichier AUROEXEC.BAT mais il peut y en avoir un. On essaie donc d'ouvrir le fichier AUTOEXEC.BAT (lignes 569–571).

Le FCB du fichier AUTOEXEC.BAT est repéré par BATFCB :

```
135 : BATFCB  DB      1,"AUTOEXECBAT"
136 :         DB      21 DUP(?)
137 :         DW      0
138 :         DW      0               ;Initialize RR field to zero
```

Le numéro de la fonction de l'interruption `21h` d'ouverture d'un fichier spécifié par un FCB est donné par la constante OPEN :

```
48 : OPEN    EQU     15
```

10^o) On initialise le champ 'taille d'un enregistrement' à un octet dans le FCB de ce fichier (ligne 572).

11^o) Si on n'a pas trouvé le fichier AUTOEXEC.BAT, on clôt le traitement par lot en plaçant 0 dans la variable BATCH, on place le décalage du code DATINIT, étudié ci-dessous, de l'initialisation de la date et de l'heure ainsi que l'adresse de segment de la partie transitoire de l'interpréteur de commandes dans la variable INITADD, et on y fait appel, c'est-à-dire qu'on l'initialise la date et l'heure (lignes 575–580).

La variable BATCH indique si on est en train d'effectuer un traitement par lot (1) ou non (0) :

```
140 : BATCH   DB      1               ;Assume batch mode initially
```

La variable INITADD contient le décalage du code DATINIT de l'initialisation de la date et de l'heure ainsi que l'adresse de segment de la partie transitoire de l'interpréteur de commandes :

```
148 : INITADD DB      4 DUP(?)
```

12^o) On va, que l'on ait trouvé le fichier AUTOEXEC.BAT ou non, à la partie HAVCOM de la routine de service de l'interruption `22h` de terminaison d'un processus pour placer 0 dans la variable LOADING, l'adresse de segment de l'emplacement de la partie transitoire de l'interpréteur de commandes dans `SI`, l'adresse ? ? dans `DI`, prendre comme segment supplémentaire celui de la partie transitoire de l'interpréteur de commandes, faire que les opérations sur les chaînes de caractères incrémentent les registres `SI` et `DI`, déplacer la partie transitoire de l'interpréteur de commandes depuis la zone de transfert de la disquette puis donner la main à la partie principale COMMAND de l'interpréteur de commandes, que nous étudierons évidemment avec le processeur de commandes (lignes 573–574 et 588–589).

3.5.3.2 Initialisation de la date et de l'heure

L'initialisation de la date et de l'heure par l'interpréteur de commandes s'effectue grâce à la routine DATINIT :

```
1787 : ;Date and time are set during initialization and use
1788 : ;this routines since they need to do a long return
1789 :
1790 : DATINIT:
1791 :          PUSH    ES
1792 :          PUSH    DS                    ;Going to use the previous stack
1793 :          MOV     AX,CS                 ;Set up the appropriate segment registers
1794 :          MOV     ES,AX
1795 :          MOV     DS,AX
1796 :          MOV     WORD PTR DS:[81H],13 ;Want to prompt for date during initialization
1797 :          CALL    DATE
1798 :          CALL    TIME
1799 :          POP     DS
1800 :          POP     ES
1801 : YYY      PROC    FAR
1802 :          RET
1803 : YYY      ENDP
```

On sauvegarde sur la pile les adresses des segments supplémentaire et des données en cours, on confond les segments supplémentaire et des données avec le segment de code, on place 13 au décalage `81h`, on appelle les routines DATE puis TIME, routines de service des commandes DATE et TIME, on restaure les adresses des segments des données et supplémentaire puis on termine la routine.

3.5.4 Routines annexes

3.5.4.1 Initialisation des vecteurs d'interruption des interruptions 22h, 23h, 24h et 27h

La routine SETVECT initialise les vecteurs d'interruption des interruptions INT 22h, INT 23h, INT 24h et INT 27h :

```
498 : SETVECT:
499 :         MOV     DX,OFFSET RESGROUP:LODCOM
500 :         MOV     AX,2522H        ;Set Terminate address
501 :         INT     21H
502 :         MOV     DX,OFFSET RESGROUP:CONTC
503 :         MOV     AX,2523H        ;Set Ctrl-C address
504 :         INT     21H
505 :         MOV     DX,OFFSET RESGROUP:DSKERR
506 :         MOV     AX,2524H        ;Set Hard Disk Error address
507 :         INT     33
508 :         MOV     DX,OFFSET RESGROUP:RESIDENT
509 :         MOV     AX,2527H        ;Set Terminate and Stay Resident address
510 :         INT     33
511 :         RET
```

— On place l'adresse de la routine de service de l'interruption 22h, repérée par l'étiquette LODCOM, dans DX, le numéro d'interruption, 22h, dans AL, le numéro de fonction, 25h, d'initialisation d'un vecteur d'interruption dans AH et on appelle l'interruption 21h.

— De même, on place l'adresse de la routine de service de l'interruption 23h, repérée par l'étiquette CONTC, dans DX, le numéro d'interruption, 23h, dans AL, le numéro de fonction, 25h, d'initialisation d'un vecteur d'interruption dans AH et on appelle l'interruption 21h.

— De même, on place l'adresse de la routine de service de l'interruption 24h, repérée par l'étiquette DSKERR, dans DX, le numéro d'interruption, 24h, dans AL, le numéro de fonction, 25h, d'initialisation d'un vecteur d'interruption dans AH et on appelle l'interruption 21h.

— Enfin, on place l'adresse de la routine de service de l'interruption 27h, repérée par l'étiquette RESIDENT, dans DX, le numéro d'interruption, 27h, dans AL, le numéro de fonction, 25h, d'initialisation d'un vecteur d'interruption dans AH, on appelle l'interruption 21h et on termine la routine.

3.5.4.2 Routine de chargement de la partie transitoire de l'interpréteur de commandes

On charge la partie transitoire de l'interpréteur de commandes grâce à la routine LOADCOM :

```
447 : LOADCOM:
448 :         PUSH    DS
449 :         MOV     DS,[TRNSEG]
450 :         MOV     DX,100H
451 :         MOV     AH,SETDMA
452 :         INT     33
453 :         POP     DS
454 :         MOV     DX,OFFSET RESGROUP:COMFCB
455 :         MOV     AH,OPEN
456 :         INT     33              ;Open COMMAND.COM
457 :         OR      AL,AL
458 :         JZ      READCOM
459 :         MOV     DX,OFFSET RESGROUP:NEEDCOM
460 : PROMPTCOM:
461 :         MOV     AH,PRINTBUF
462 :         INT     33
463 :         MOV     AX,0C07H         ;Get char without testing or echo
464 :         INT     33
465 :         JMP     SHORT LOADCOM
466 : READCOM:
467 :         MOV     WORD PTR[COMFCB+RR],OFFSET RESGROUP:TRANSTART
468 :         XOR     AX,AX
469 :         MOV     WORD PTR[COMFCB+RR+2],AX
470 :         MOV     [COMFCB],AL      ;Use default drive
471 :         INC     AX
472 :         MOV     WORD PTR[COMFCB+RECLEN],AX
473 :         MOV     CX,COMLEN
474 :         MOV     DX,OFFSET RESGROUP:COMFCB
475 :         MOV     AH,RDBLK
476 :         INT     33
477 :         OR      AL,AL
478 :         JZ      RET10
479 : WRONGCOM:
480 :         MOV     DX,OFFSET RESGROUP:COMBAD
481 :         JMP     SHORT PROMPTCOM
```

— On sauvegarde l'adresse du segment des données en cours sur la pile, on confond le segment des données avec le segment de la partie transitoire de l'interpréteur de commandes, on initialise l'adresse de la zone de transfert depuis la disquette au décalage 256 du segment de la partie transitoire de l'interpréteur de commandes et on restaure la valeur de `DS` (lignes 448–453).

Le numéro de la fonction de l'interruption `21h` d'initialisation de la zone de transfert en mémoire centrale depuis la disquette est spécifié par la constante SETDMA :

```
54 : SETDMA  EQU     26
```

— On ouvre le fichier COMMAND.COM (lignes 454–456).

La constante COMDRV dépend de la version du système d'exploitation (PC-DOS ou MS-DOS) :

```
30 :         IF      IBMVER
31 : SYM     EQU     ">"
32 : COMDRV  EQU     1
33 :         ENDIF
34 :
35 :         IF      MSVER
36 : SYM     EQU     ":"
```

```
37 : COMDRV  EQU     0
38 :         ENDIF
```

La version du système d'exploitation, MS-DOS ou PC-DOS, est choisie avant l'assemblage, comme nous l'avons vu.

Le FCB du fichier COMMAND.COM est repéré, dans le fichier COMMAND.ASM, par l'étiquette COMFCB :

```
141 : COMFCB  DB      COMDRV,"COMMAND COM"
142 :         DB      25 DUP(?)
```

— Si le numéro de lecteur de disquette n'est pas celui par défaut, on affiche un message à l'écran jusqu'à ce que l'utilisateur ait placé la disquette système dans le lecteur de disquette par défaut (lignes 457–465).

Le message destiné à l'utilisateur est repéré par l'étiquette NEEDCOM :

```
125 : NEEDCOM DB      13,10,"Insert DOS disk in "
126 :         IF      IBMVER
127 :         DB      "drive A"
128 :         ELSE
129 :         DB      "default drive"
130 :         ENDIF
131 : PROMPT  DB      13,10,"and strike any key when ready",13,10,"$"
```

Le numéro de la fonction d'affichage d'une chaîne de caractères à l'écran de l'interruption `21h` est donné par la constante PRINTBUF :

```
57 : PRINTBUF EQU    9
```

— On lit le fichier COMMAND.COM et on en place le contenu dans la zone de transfert définie ci-dessus, c'est-à-dire à partir du décalage 256 de la zone de la partie transitoire de l'interpréteur de commandes. Pour cela on place l'adresse du début de la zone de la partie transitoire de l'interpréteur de commandes dans le champ de lecture directe du FCB de COMMAND.COM, 0 comme premier octet du FCB pour utiliser le lecteur de disquette par défaut, 1 dans le champ 'taille d'un enregistrement', la taille de la partie transitoire dans `CX`, le décalage du FCB dans `DX` et on appelle la fonction de lecture directe de l'interruption `21h` (lignes 466–476).

L'étiquette TRANSTART repère le début du code de la partie transitoire :

```
609 : ;This TAIL segment is used to produce a PARA aligned label in the resident
610 : ; group which is the location where the transient segments will be loaded
611 : ; initialy.
612 :
613 : TAIL    SEGMENT PARA
614 :         ORG     0
615 : TRANSTART       LABEL   WORD
616 : TAIL    ENDS
```

Les décalages dans le FCB des champs 'lecture directe' et 'taille d'un enregistrement' d'un FCB sont donnés par les constantes RR et RELEN :

```
64 : RR      EQU     33
65 : RECLEN  EQU     14
```

— On termine la routine s'il n'y a pas eu d'erreur de lecture sur la disquette. Sinon on indique à l'utilisateur qu'il s'agit d'un interpréteur de commandes non valide et on recommence jusqu'à ce que l'utilisateur ait introduit une disquette système dans le lecteur de disquette par défaut (lignes 477–481).

Le message d'erreur COMBAD est :

```
124 : COMBAD  DB      13,10,"Invalid COMMAND.COM"
125 : NEEDCOM DB      13,10,"Insert DOS disk in "
126 :         IF      IBMVER
127 :         DB      "drive A"
```

```
128 :         ELSE
129 :         DB      "default drive"
130 :         ENDIF
131 : PROMPT  DB      13,10,"and strike any key when ready",13,10,"$"
```

3.5.4.3 Routine de calcul de la somme de contrôle de la partie transitoire

La routine CHKSUM calcule la somme de contrôle de la partie transitoire de l'interpréteur de commandes et la place dans `DX` :

```
483 : CHKSUM:
484 :         CLD
485 :         PUSH    DS
486 :         MOV     DS,[TRNSEG]
487 :         MOV     SI,100H
488 :         MOV     CX,COMLEN
489 :         SHR     CX,1
490 :         XOR     DX,DX
491 : CHK:
492 :         LODSW
493 :         ADD     DX,AX
494 :         LOOP    CHK
495 :         POP     DS
496 :         RET
```

— Les opérations sur les chaînes de caractères décrémenteront les registres `SI` et `DI` (ligne 484).

— On sauvegarde la valeur de `DS` sur la pile, on prend comme segment des données le segment de la partie transitoire de l'interpréteur de commandes, on initialise `SI` à `100h`, début du code de la partie transitoire, `CX` avec la moitié de la taille de la partie transitoire de l'interpréteur de commandes, puisqu'on va travailler avec des mots et non des octets, `DX` à zéro et on lui ajoute les mots constituant ce code, sans tenir compte des retenues (lignes 485–494).

La taille de la partie transitoire de l'interpréteur de commandes est spécifiée par la constante COMLEN :

```
2163 : COMLEN  EQU     TRANDATASIZE+TRANCODESIZE-102H
          ;End of COMMAND load. ZERO Needed to make COMLEN absolute
```

— On restaure la valeur de `DS` et on termine la routine (lignes 495–496).