Chapitre 6

Implémentation des routines de l'interpréteur de commande

6.1 La partie résidente de l'interpréteur de commande

6.1.1 La zone des données de la partie résidente

COMMAND.COM commence par la zone des données de la partie résidente de l'interpréteur de commande, que nous commenterons au fur et à mesure de leur utilisation :

```
96  : ;Data for resident portion
97  :
98  : DATARES SEGMENT BYTE
99  :         ORG     0
100 : ZERO    =       $
101 : MESBAS  DW      OFFSET RESGROUP:ERR0
102 :         DW      OFFSET RESGROUP:ERR2
103 :         DW      OFFSET RESGROUP:ERR4
104 :         DW      OFFSET RESGROUP:ERR6
105 :         DW      OFFSET RESGROUP:ERR8
106 :         DW      OFFSET RESGROUP:ERR10
107 :         DW      OFFSET RESGROUP:ERR12
108 : ERR0    DB      "Write protect$"
109 : ERR2    DB      "Not ready$"
110 : ERR4    DB      "Data$"
111 : ERR6    DB      "Seek$"
112 : ERR8    DB      "Sector not found$"
113 : ERR10   DB      "Write fault$"
114 : ERR12   DB      "Disk$"
115 : READ    DB      "read$"
116 : WRITE   DB      "writ$"
117 : ERRMES  DB      " error "
118 : IOTYP   DB      "writing"
119 : DRVNUM  DB      " drive "
120 : DRVLET  DB      "A"
121 : NEWLIN  DB      13,10,"$"
122 : REQUEST DB      "Abort, Retry, Ignore? $"
123 : BADFAT  DB      13,10,"File allocation table bad,$"
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,"$"
132 : NEEDBAT DB      13,10,"Insert disk with batch file$"
133 : ENDBATMES DB    13,10,"Terminate batch job (Y/N)? $"
134 : LOADING DB      0
135 : BATFCB  DB      1,"AUTOEXECBAT"
136 :         DB      21 DUP(?)
137 :         DW      0
138 :         DW      0               ;Initialize RR field to zero
139 : PARMTAB DW      10 DUP(-1)      ;No parameters initially
140 : BATCH   DB      1               ;Assume batch mode initially
141 : COMFCB  DB      COMDRV,"COMMAND COM"
142 :         DB      25 DUP(?)
143 : TRANS   DW      OFFSET TRANGROUP:COMMAND
144 : TRNSEG  DW      ?
145 : BATBYT  DB      ?
146 : MEMSIZ  DW      ?
147 : SUM     DW      ?
148 : INITADD DB      4 DUP(?)
149 : RESDATASIZE     EQU     $-ZERO
150 : DATARES ENDS
```

6.1.2 Routine de service de l'interruption 22h de terminaison d'un processus

6.1.2.1 Routine principale

La routine de service de l'interruption INT 22h, de terminaison d'un processus est, comme nous venons de le voir, repérée par l'étiquette LODCOM :

```
321 : LODCOM:
322 :         MOV     AX,CS
323 :         MOV     SS,AX
324 :         MOV     SP,OFFSET RESGROUP:RSTACK
325 :         MOV     DS,AX
326 :         CALL    SETVECT
327 :         CALL    CHKSUM
328 :         CMP     DX,[SUM]
329 :         JZ      HAVCOM
330 :         MOV     [LOADING],1
331 :         CALL    LOADCOM
332 : CHKSAME:
333 :         CALL    CHKSUM
334 :         CMP     DX,[SUM]
335 :         JZ      HAVCOM
336 :         CALL    WRONGCOM
337 :         JMP     SHORT CHKSAME
338 : HAVCOM:
339 :         MOV     [LOADING],0
340 :         MOV     SI,OFFSET RESGROUP:LTPA
341 :         MOV     DI,OFFSET TRANGROUP:TPA
342 :         MOV     ES,[TRNSEG]
343 :         CLD
344 :         MOVSW           ;Move TPA segment to transient storage
345 :         MOVSW           ;Move resident segment too
346 :         MOV     AX,[MEMSIZ]
347 :         MOV     WORD PTR ES:[2],AX
348 :         JMP     DWORD PTR [TRANS]
```

— On confond les segments de pile et des données avec le segment de code et on initialise le pointeur de pile au début du segment de code (lignes 322–325).

— On appelle la routine SETVECT pour initialiser les vecteurs d'interruption des interruptions INT 22h, INT 23h, INT 24h et INT 27h (ligne 326).

— On appelle la routine CHKSUM pour calculer la somme de contrôle de la partie transitoire et la placer dans DX (ligne 327).

— Si elle est différente de la valeur calculée lors de l'initialisation, on place 1 dans la variable LOADING, pour indiquer qu'on a besoin de charger (à nouveau) la partie transitoire de l'interpréteur de commande, on appelle la routine LOADCOM, pour charger à nouveau la partie transitoire de l'interpréteur de commande puis on appelle à nouveau la routine CHKSUM pour calculer la somme de contrôle de la partie transitoire et la placer dans DX. Si elle n'est toujours pas égale à la valeur calculée lors de l'initialisation, on appelle la routine WRONGCOM, partie de la routine LOADCOM, pour afficher un message destiné à l'utilisateur et réessayer (lignes 328–337).

La variable LOADING contient 1 pour indiquer qu'on a besoin de charger (à nouveau) la partie transitoire de l'interpréteur de commande, et 0 sinon :

```
134 : LOADING DB      0
```

Elle est initialisée à 0 puisque la partie résidente est chargée lors de l'initialisation.

— Lorsqu'on a trouvé que la somme de contrôle est égale à celle calculée lors de l'initialisation, on place 0 dans la variable LOADING, pour indiquer qu'on n'a plus besoin de charger la partie transitoire de l'interpréteur de commande, on fait pointer `SI` sur la zone de transfert depuis la disquette, `DI` sur sur le segment de la partie transitoire de l'interpréteur de commande, on prend comme segment supplémentaire celui de la partie transitoire de l'interpréteur de commande, les opérations sur les chaînes de caractères incrémenteront les registres `SI` et `DI` et on déplace la partie transitoire de l'interpréteur de commande depuis la zone de transfert de la disquette (lignes 329, 335 et 338-345).

La variable TPA est définie ligne 215 :

```
215 : TPA     DW      1 DUP(?)
```

TRANGROUP est le groupe de segments de la partie transitoire :

```
84 : TRANCODE        SEGMENT PARA
85 : TRANCODE        ENDS
86 :
87 : TRANDATA        SEGMENT BYTE
88 : TRANDATA        ENDS
89 :
90 : TRANSPACE       SEGMENT BYTE
91 : TRANSPACE       ENDS
92 :
93 : RESGROUP        GROUP   CODERES,DATARES,INIT,TAIL
94 : TRANGROUP       GROUP   TRANCODE,TRANDATA,TRANSPACE
```

— On place la taille de la mémoire comme deuxième octet et on redonne la main au processeur de commande (lignes 347–348).

La variable TRANS est définie ligne 215 :

```
143 : TRANS   DW      OFFSET TRANGROUP:COMMAND
```

Nous verrons que l'étiquette COMMAND repère le début du code de la partie transitoire du processeur de commande, que nous étudierons ci-après.

6.1.3 Routine de service de l'interruption 23h de CTRL-C

La routine de service de l'interruption INT `23h`, appelée par un appui sur la combinaison de touches CTRL-C, est, comme nous l'avons vu, repérée par l'étiquette CONTC :

```
298 : CONTC:
299 :         MOV     AX,CS
300 :         MOV     DS,AX
301 :         MOV     SS,AX
302 :         MOV     SP,OFFSET RESGROUP:RSTACK
303 :         STI
304 :         CALL    SETVECT
305 :         MOV     AH,DSKRESET
306 :         INT     33              ;Reset disks in case files were open
307 :         TEST    [BATCH],-1
308 :         JZ      LODCOM
309 : ASKEND:
310 :         MOV     DX,OFFSET RESGROUP:ENDBATMES
311 :         MOV     AH,PRINTBUF
312 :         INT     33
313 :         MOV     AX,0C00H+INCHAR
314 :         INT     33
315 :         AND     AL,5FH
316 :         CMP     AL,"N"
317 :         JZ      LODCOM
318 :         CMP     AL,"Y"
319 :         JNZ     ASKEND
320 :         MOV     [BATCH],0
321 : LODCOM:
```

— On confond les segments de données et supplémentaire avec le segment de code, on initialise le pointeur de pile au début de la pile de la partie résidente du processeur de commande, les opérations sur les chaînes de caractères incrémenteront les registres `SI` et `DI` et on appelle la routine SETVEC pour initialiser les vecteurs d'interruption des interruptions INT `22h`, INT `23h`, INT `24h` et INT `27h` (lignes 299–304).

— On réinialise la disquette, en particulier pour fermer les fichiers qui ne l'auraient pas été (lignes 305–306).

Le numéro de la fonction de l'interruption `21h` effectuant une réinitialistion de la disquette est repéré par la constante DSKREST :

```
41 : DSKRESET EQU    13
```

— Si on est en mode traitement par lot, on demande à l'utilisateur si on doit terminer entièrement le lot, la réponse saisie au clavier devant être l'une des majuscules 'Y' ou 'N', et on attend qu'un caractère soit tapé au clavier. Si l'utilisateur appuie sur un caractère autre que 'Y' ou 'N', on recommence jusqu'à ce qu'il appuie sur l'un de ces deux caractères. S'il appuie sur 'Y', on place 0 dans la variable BATCH pour clore le traitement par lot (lignes 307–320).

Le message ENDBATMES destiné à l'utilisateur est :

```
133 : ENDBATMES DB    13,10,"Terminate batch job (Y/N)? $"
```

Le numéro de la fonction de l'interruption `21h` de saisie d'un caractère au clavier est repéré par la constante INCHAR :

```
46 : INCHAR  EQU     1
```

— On exécute le code de la routine de service de l'interruption `22h` de terminaison d'un processus (ligne 321).

6.1.4 Routine de service de l'interruption 24h de traitement d'une erreur critique

6.1.4.1 Routine principale

La routine de service de l'interruption INT 24h de traitement d'une erreur critique est, comme nous l'avons vu, repérée par l'étiquette DSKERR :

```
359 : DSKERR:
360 :          ;*************************************************
361 :          ;       THIS IS THE DEFAULT DISK ERROR HANDLING CODE
362 :          ;       AVAILABLE TO ALL USERS IF THEY DO NOT TRY TO
363 :          ;       INTERCEPT INTERRUPT 24H.
364 :          ;*************************************************
365 :          STI
366 :          PUSH    DS
367 :          PUSH    CS
368 :          POP     DS              ;Set up local data segment
369 :          PUSH    DX
370 :          CALL    CRLF
371 :          POP     DX
372 :          ADD     AL,"A"          ;Compute drive letter
373 :          MOV     [DRVLET],AL
374 :          TEST    AH,80H          ;Check if hard disk error
375 :          JNZ     FATERR
376 :          MOV     SI,OFFSET RESGROUP:READ
377 :          TEST    AH,1
378 :          JZ      SAVMES
379 :          MOV     SI,OFFSET RESGROUP:WRITE
380 : SAVMES:
381 :          LODSW
382 :          MOV     WORD PTR [IOTYP],AX
383 :          LODSW
384 :          MOV     WORD PTR [IOTYP+2],AX
385 :          AND     DI,0FFH
386 :          CMP     DI,12
387 :          JBE     HAVCOD
388 :          MOV     DI,12
389 : HAVCOD:
390 :          MOV     DI,WORD PTR [DI+MESBAS] ;Get pointer to error message
391 :          XCHG    DI,DX           ;May need DX later
392 :          MOV     AH,PRINTBUF
393 :          INT     33              ;Print error type
394 :          MOV     DX,OFFSET RESGROUP:ERRMES
395 :          INT     33
396 :          CMP     [LOADING],0
397 :          JNZ     GETCOMDSK
398 : ASK:
399 :          MOV     DX,OFFSET RESGROUP:REQUEST
400 :          MOV     AH,PRINTBUF
401 :          INT     33
402 :          MOV     AX,0C00H+INCHAR
403 :          INT     33              ;Get response
404 :          CALL    CRLF
405 :          OR      AL,20H          ;Convert to lower case
406 :          MOV     AH,0            ;Return code for ignore
407 :          CMP     AL,"i"          ;Ignore?
408 :          JZ      EXIT
409 :          INC     AH
410 :          CMP     AL,"r"          ;Retry?
411 :          JZ      EXIT
412 :          INC     AH
```

```
413 :          CMP     AL,"a"          ;Abort?
414 :          JNZ     ASK
415 : EXIT:
416 :          MOV     AL,AH
417 :          MOV     DX,DI
418 :          POP     DS
419 :          IRET
420 :
421 : FATERR:
422 :          MOV     DX,OFFSET RESGROUP:BADFAT
423 :          MOV     AH,PRINTBUF
424 :          INT     33
425 :          MOV     DX,OFFSET RESGROUP:DRVNUM
426 :          INT     33
427 :          MOV     AL,2            ;Abort
428 :          POP     DS
429 :          IRET
430 :
431 : GETCOMDSK:
432 :          MOV     DX,OFFSET RESGROUP:NEEDCOM
433 :          MOV     AH,PRINTBUF
434 :          INT     33
435 :          MOV     AX,0C07H        ;Get char without testing or echo
436 :          INT     33
437 :          JMP     LODCOM
```

— On se met à l'écoute des interruptions masquables, on sauvegarde sur la pile l'adresse du segment des données et le contenu de `DX`, on confond le segment des données avec le segment de code et on appelle la routine CRLF, étudiée ci-dessous, pour passer à la ligne à l'écran (lignes 365–370).

— On restaure la valeur de `DX` et on place le caractère 'A' dans la variable DRVLET (lignes 371–373).

La variable DRVLET contient l'identificateur du lecteur de disquette ('A' ou 'B'), initialisée à 'A' :

```
120 : DRVLET  DB      "A"
```

— Si le code d'erreur, contenu dans `AH`, n'est pas égal à `80h`, on affiche un message d'erreur à l'écran, on place le code d'erreur `02h` dans `AL`, on restaure la valeur de `DS` et on termine la routine de service (lignes 374–375 et 421–429).

Le message d'erreur BADFAT est le suivant :

```
123 : BADFAT  DB      13,10,"File allocation table bad,$"
```

La chaîne de caractères DRVNUM est :

```
119 : DRVNUM  DB      " drive "
```

— Sinon on fait pointer `SI` sur la chaîne de caractères « read ». Si le contenu de `AH` n'est pas 1, on le fait pointer sur la chaîne de caractères « write » (lignes 376–379).

Les chaînes de caractères READ et WRITE sont :

```
115 : READ    DB      "read$"
116 : WRITE   DB      "writ$"
```

— On place les quatre caractères de la chaîne de caractères choisie dans la variable IOTYP (lignes 380–384).

La variable chaîne de caractères IOTYP contient le type entrée-sortie :

```
118 : IOTYP   DB      "writing"
```

— On place 12 dans `DI` s'il est supérieur à 12 (lignes 385–388).

— On fait pointer `DI` sur le message d'erreur, on en sauvegarde la valeur dans `DX` et on affiche le message d'erreur à l'écran, constitué du type d'erreur suivi de « error » (lignes 389–395).

Les messages d'erreur sont définis dans la zone des données de la partie résidente du processeur de commandes :

```
101 : MESBAS  DW      OFFSET RESGROUP:ERR0
102 :         DW      OFFSET RESGROUP:ERR2
103 :         DW      OFFSET RESGROUP:ERR4
104 :         DW      OFFSET RESGROUP:ERR6
105 :         DW      OFFSET RESGROUP:ERR8
106 :         DW      OFFSET RESGROUP:ERR10
107 :         DW      OFFSET RESGROUP:ERR12
108 : ERR0    DB      "Write protect$"
109 : ERR2    DB      "Not ready$"
110 : ERR4    DB      "Data$"
111 : ERR6    DB      "Seek$"
112 : ERR8    DB      "Sector not found$"
113 : ERR10   DB      "Write fault$"
114 : ERR12   DB      "Disk$"
[...]
117 : ERRMES  DB      " error "
```

— Si la variable LOADING n'est pas égale à 0, c'est-à-dire si on doit charger la partie transitoire de l'interpréteur de commandes, on demande à l'utilisateur de placer une disquette système dans le lecteur de disquette par défaut et, après que celui-ci ait appuyé sur une touche, on exécute le code de l'interruption `22h` de terminaison d'un processus (lignes 396–397 et 431–437).

— Sinon on lui demande comment continuer, on saisit la réponse, qui doit être constituée de l'un des caractères 'A', 'R' ou 'I', on va à la ligne à l'écran et on met en minuscule la lettre saisie (lignes 398–405).

Le message de continuation est REQUEST :

```
122 : REQUEST DB      "Abort, Retry, Ignore? $"
```

— S'il s'agit de 'i' (pour 'Ignore') on place le code d'erreur `00h` dans `AL`, la valeur de `DI` dans `DX`, on restaure la valeur de `DS` et on termine la routine de service (lignes 406–408 et 415–419).

— S'il s'agit de 'r' (pour 'Retry') on place le code d'erreur `01h` dans `AL`, la valeur de `DI` dans `DX`, on restaure la valeur de `DS` et on termine la routine de service (lignes 409–411 et 415–419).

— S'il s'agit de 'a' (pour 'Abort') on place le code d'erreur `02h` dans `AL`, la valeur de `DI` dans `DX`, on restaure la valeur de `DS` et on termine la routine de service (lignes 412–419).

— S'il ne s'agit d'aucune de ces trois lettres, on repose la question à l'utilisateur jusqu'à ce qu'il tape l'une de ces lettres (ligne 414).

6.1.4.2 Routine de passage à la ligne

La routine de passage à la ligne sur l'écran d'affichage est CRLF :

```
439 : CRLF:
440 :         MOV     DX,OFFSET RESGROUP:NEWLIN
441 :         PUSH    AX
442 :         MOV     AH,PRINTBUF
443 :         INT     33
444 :         POP     AX
445 : RET10:  RET
```

Rappelons qu'en MS-DOS un passage à la ligne est spécifié par les deux caractères de code ASCII 13 et 10. L'étiquette :

```
121 : NEWLIN  DB      13,10,"$"
```

repère la chaîne de caractères MS-DOS constituée de ces deux caractères.

On fait pointer `DX` sur l'étiquette NEWLIN, on sauvegarde la valeur de `AX` sur la pile, on appelle la fonction de l'interruption `21h` d'affichage de la chaîne de caractères pointée par `DX`, on restaure la valeur de `AX` et on termine la routine.

6.1.5 Routine de service de l'interruption 27h de terminaison d'un processus en laissant son code en mémoire

La routine de service de l'interruption INT `27h`, de terminaison d'un processus en laissant son code en mémoire, est, comme nous l'avons vu, repérée par l'étiquette RESIDENT :

```
350 : RESIDENT:
351 :         ADD     DX,15
352 :         MOV     CL,4
353 :         SHR     DX,CL              ;Number of paragraphs of new addition
354 :         ADD     CS:[LTPA],DX
355 :         XOR     AX,AX
356 :         MOV     DS,AX
357 :         JMP     DWORD PTR DS:[80H] ;Pretend user executed INT 20H
```

— On ajoute 15 au contenu de `DX`, que l'on divise par 16 pour obtenir le nombre de paragraphes, et on l'ajoute à la variable LPTA (lignes 351–354).

— On place 0 dans `DS` et on va à l'adresse absolue `80h` pour exécuter l'interruption `20h` de terminaison d'un programme (lignes 355–357).

6.2 La partie transitoire de l'interpréteur de commande

Nous avons vu que la partie initialisation de l'interpréteur de commande, ainsi que les routines de service des interruptions `22h` de terminaison d'un processus et `23h` renvoient à la partie étiquetée par COMMAND.

6.2.1 Fichier de traitement par lot ou commande en ligne ?

La partie étiquetée par COMMAND, qui se trouve au début de la partie transitoire de l'interpréteur de commande, affiche l'invite de commande, saisit ce qui est écrit et détermine s'il s'agit d'un fichier de traitement par lot ou s'il faut exécuter une commande en ligne :

```
618 : ;***************************************************************************
619 : ;START OF TRANSIENT PORTION
620 : ;This code is loaded at the end of memory and may be overwritten by
621 : ;memory-intensive user programs.
622 :
623 : TRANCODE        SEGMENT PARA
624 : ASSUME  CS:TRANGROUP,DS:TRANGROUP,ES:TRANGROUP,SS:TRANGROUP
625 :
626 : WSWITCH EQU     1               ;Wide display during DIR
627 : PSWITCH EQU     2               ;Pause (or Page) mode during DIR
628 : VSWITCH EQU     4               ;Verify during COPY
629 : ASWITCH EQU     8               ;ASCII mode during COPY
630 : BSWITCH EQU     10H             ;Binary mode during COPY
631 :
632 :         ORG     0
633 : ZERO    =       $
634 :
635 :         ORG     100H            ;Allow for 100H parameter area
636 :
637 : SETDRV:
638 :         MOV     AH,SELDRV
639 :         INT     21H
640 : COMMAND:
641 :         CLD
642 :         MOV     AX,CS
643 :         MOV     SS,AX
644 :         MOV     SP,OFFSET TRANGROUP:STACK
645 :         MOV     ES,AX
646 :         MOV     DS,AX
647 :         STI
648 :         MOV     AX,46*100H
649 :         MOV     DL,0
650 :         INT     33              ;Turn off verify after write
651 :         MOV     AX,CS           ;Get segment we're in
652 :         SUB     AX,[TPA]        ;AX=size ot TPA in paragraphs
653 :         MOV     DX,16
654 :         MUL     DX              ;DX:AX=size of TPA in bytes
655 :         OR      DX,DX           ;See if over 64K
656 :         JZ      SAVSIZ          ;OK if not
657 :         MOV     AX,-1           ;If so, limit to 65535 bytes
658 : SAVSIZ:
659 :         MOV     [BYTCNT],AX     ;Max no. of bytes that can be buffered
660 :         CALL    CRLF2
661 : GETCOM:
662 :         MOV     AH,GETDRV
663 :         INT     21H
664 :         MOV     [CURDRV],AL
665 :         ADD     AL,"A"
```

```
666 :          CALL    OUT             ;Print letter for default drive
667 :          MOV     AL,SYM
668 :          CALL    OUT
669 :          MOV     DS,[RESSEG]     ;All batch work must use resident seg.
670 : ASSUME   DS:RESGROUP
671 :          TEST    [BATCH],-1
672 :          JNZ     READBAT
673 :          PUSH    CS
674 :          POP     DS              ;Need local segment to point to buffer
675 : ASSUME   DS:TRANGROUP
676 :          MOV     DX,OFFSET TRANGROUP:COMBUF
677 :          MOV     AH,INBUF
678 :          INT     21H             ;Get a command
679 :          JMP     DOCOM
```

— Les opérations sur les chaînes de caractères décrémenteront les registres `SI` et `DI`, on confond les segments de pile, supplémentaire et des données avec le segment de code, on initialise le pointeur de pile à la fin de la pile, repérée par l'étiquette STACK et on se met à l'écoute des interruptions masquables (lignes 641–657).

L'étiquette STACK repère le sommet de la pile de la partie transitoire de l'interpréteur de commande, d'une taille de 128 octets :

```
262 :          DB      80H DUP(?)
263 : STACK    LABEL   WORD
264 :
265 : PRETRLEN         EQU     $-ZERO          ;Used later to compute TRNLEN
```

— On appelle la fonction 46 de l'interruption `21h` pour désenclencher la vérification lors de l'écriture sur une disquette (lignes 648–650).

— On défalque la taille, en paragraphes, de l'adresse du segment dans lequel on se trouve, que l'on multiplie par 16 pour obtenir la taille en octets, qu'on limite à 65 535 octets et que l'on place dans la variable BYTCNT (lignes 651–659).

La variable BYTCNT contient la taille en octets :

```
228 : BYTCNT   DW      1 DUP(?)
```

— On appelle la sous-routine CRLF2 pour passer à la ligne sur l'écran d'affichage (ligne 660).

Les routines CRLF2 et OUT n'ont pas besoin de commentaires :

```
1172 : CRLF2:
1173 :          MOV     AL,13
1174 :          CALL    OUT
1175 :          MOV     AL,10
1176 :          JMP     OUT
[...]
2018 : OUT:
2019 : ;Print char in AL without affecting registers
2020 :          XCHG    AX,DX
2021 :          PUSH    AX
2022 :          MOV     AH,OUTCH
2023 :          INT     33
2024 :          POP     AX
2025 :          XCHG    AX,DX
2026 :          RET
```

— On appelle la fonction 25 de l'interruption `21h` pour placer dans `AL` le numéro du lecteur de disquette par défaut (0 ou 1), puis le placer dans la variable CURDRV (lignes 661–664).

Le numéro de la fonction de l'interruption `21h` plaçant dans `AL` le numéro du lecteur de disquette par défaut (0 ou 1) est repéré par la constante GETDRV :

```
56 : GETDRV  EQU     25
```

La variable CURDRV contient le numéro du lecteur de disquette par défaut (0 ou 1) :

```
219 : CURDRV  DB      1 DUP(?)
```

— On ajoute 'A' au numéro de lecteur de disquette par défaut (pour obtenir 'A' ou 'B'), que l'on affiche à l'écran (lignes 665–666).

— On affiche ensuite le caractère ':' (lignes 667–668).

Le caractère affiché après l'identificateur du lecteur de disquette dépend de la version du système d'exploitation (PC-DOS ou MS-DOS) et est précisé par la constante SYM :

```
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
```

— On prend comme segment des données celui de la partie transitoire de l'interpréteur de commande. Si le contenu de la variable BATCH n'est pas `FFh`, on va à la partie READBAT, étudiée ci-dessous, pour exécuter les commandes du fichier AUTOEXEC.BAT (lignes 669–672).

La variable RESSEG contient l'adresse de segment de la partie transitoire du processeur de commandes :

```
216 : RESSEG  DW      1 DUP(?)
```

— Sinon on confond le segment des données avec le segment de code, on lit une chaîne de caractères au clavier, qui doit être une commande, on la place dans le tampon COMBUF, et on va à la partie DOCOM, étudiée ci-dessous, pour l'exécuter (lignes 673–679).

Le tampon COMBUF contient une chaîne de caractères, qui doit être une commande :

```
205 : COMBUF  DB      128,1,13
```

6.2.2 Routine du traitement de AUTOEXEC.BAT

6.2.2.1 Routine principale

La routine READBAT traite le contenu du fichier AUTOEXEC.BAT :

```
722 : READBAT:
723 :         MOV     DX,OFFSET RESGROUP:BATFCB
724 :         MOV     AH,OPEN
725 :         INT     33                      ;Make sure batch file still exists
726 :         OR      AL,AL
727 :         JNZ     PROMPTBAT               ;If OPEN fails, prompt for disk
728 :         MOV     WORD PTR [BATFCB+RECLEN],1
729 :         MOV     DX,OFFSET RESGROUP:BATBYT
730 :         MOV     AH,SETDMA
731 :         INT     33
732 :         MOV     DI,OFFSET TRANGROUP:COMBUF+2
733 : RDBAT:
734 :         CALL    GETBATBYT
735 :         CMP     AL,"%"                  ;Check for parameter
736 :         JZ      NEEDPARM
737 : SAVBATBYT:
738 :         STOSB
739 :         CALL    OUT                     ;Display batched command line
740 :         CMP     AL,0DH
741 :         JNZ     RDBAT
742 :         SUB     DI,OFFSET TRANGROUP:COMBUF+3
743 :         MOV     AX,DI
744 :         MOV     ES:[COMBUF+1],AL        ;Set length of line
745 :         CALL    GETBATBYT               ;Eat linefeed
746 :         PUSH    CS
747 :         POP     DS                      ;Go back to local segment
748 : ASSUME DS:TRANGROUP
749 : DOCOM:
```

— On essaie d'ouvrir le fichier AUTOEXEC.BAT (lignes 723–725).

— Si on n'y parvient pas, on va à la partie PROMPBAT, étudiée ci-dessous, pour demander à l'utilisateur d'insérer une disquette contenant un fichier AUTOEXEC.BAT (lignes 726–727).

— Sinon on place 1 dans le champ taille d'un enregistrement du FCB de AUTOEXEC.BAT, l'adresse de la variable BATBYT dans `DX` et on appelle la fonction de l'interruption `21h` d'initialisation de l'adresse de la zone de téléchargement (lignes 728–731).

La variable BATBYT sert de tampon lors de la lecture d'un octet du fichier AUTOEXEC.BAT :

```
145 : BATBYT  DB      ?
```

— On fait pointer `DI` sur le deuxième octet du tampon d'une ligne de commande et on appelle la sous-routine GETBATBYT, étudiée ci-dessous, pour placer dans `AL` cet octet du fichier AUTOEXEC.BAT (lignes 732–734).

— S'il s'agit du caractère '%', on va à la partie NEEDPARM, étudiée ci-dessous, de prise en compte d'un paramètre (lignes 735–736).

— Sinon on place le caractère dans le tampon, tout en l'affichant en écho à l'écran (lignes 738–739).

— S'il ne s'agit pas du caractère de code ASCII 13, donc du passage à la ligne, on revient à partir de l'étiquette RDBAT pour lire le caractère suivant (lignes 740–741).

— Sinon on place la taille de la ligne de commande dans DI et comme premier octet du tampon COMBUF, on appelle la sous-routine GETBATBYT pour écarter le second caractère de passage à la ligne (celui de code ASCII 10), on confond le segment des données avec le segment de code et on passe à la partie DOCOM, étudiée ci-dessous, de la partie transitoire de l'interpréteur de commande pour exécuter la commande qui vient d'être extraite du fichier AUTOEXEC.BAT (lignes 742–749).

6.2.2.2 Sous-routine de demande du fichier AUTOEXEC.BAT

La sous-routine de demande du fichier AUTOEXEC.BAT est PROMPTBAT :

```
709 : PROMPTBAT:
710 :          MOV     AH,PRINTBUF
711 :          MOV     DX,OFFSET RESGROUP:NEEDBAT
712 :          INT     33              ;Prompt for batch file
713 :          MOV     AH,PRINTBUF
714 :          MOV     DX,OFFSET RESGROUP:PROMPT
715 :          INT     33
716 :          MOV     AX,0C00H+INCHAR
717 :          INT     33
718 :          JMP     COMMAND
```

On demande à l'utilisateur d'insérer la disquette contenant le fichier AUTOEXEC.BAT et d'appuyer sur une touche lorsque ceci a été effectué. On attend qu'il ait frappé sur une touche, retournant alors au début de la partie transitoire de l'interpréteur de commande.

Le message destiné à l'utilisateur est NEEDBAT :

```
132 : NEEDBAT DB      13,10,"Insert disk with batch file$"
```

6.2.2.3 Sous-routine d'obtention d'un octet de AUTOEXEC.BAT

La sous-routine d'obtention d'un octet de AUTOEXEC.BAT est GETBATBYT :

```
1743 : GETBATBYT:
1744 : ;Get one byte from the batch file and return it in AL. End-of-file
1745 : ;returns <CR> and ends batch mode. DS must be set to resident segment.
1746 : ;AH, CX, DX destroyed.
1747 : ASSUME  DS:RESGROUP
1748 :         MOV     DX,OFFSET RESGROUP:BATFCB
1749 :         MOV     AH,RDBLK
1750 :         MOV     CX,1
1751 :         INT     33              ;Get one more byte from batch file
1752 :         JCXZ    BATEOF
1753 :         MOV     AL,[BATBYT]
1754 :         CMP     AL,1AH
1755 :         JNZ     RET70
1756 : BATEOF:
1757 :         MOV     AL,0DH          ;If end-of-file, then end of line
1758 :         MOV     [BATCH],0       ;And turn off batch mode
1759 : RET70:  RET
```

— On lit un octet du fichier AUTOEXEC.BAT et on le place dans la variable BATBYT (lignes 1747–1751).

Le numéro de la fonction de l'interruption `21h` de lecture directe sur un fichier spécifié par un FCB de plusieurs enregistrements est donné par la constante RDBLK :

```
52 : RDBLK   EQU     39
```

— Si on est arrivé à la fin du fichier, on place un caractère de fin de lignes (celui de code ASCII 13) dans `AL`, 0 dans la variable BATCH, pour indiquer qu'on a terminé de lire le fichier AUTOEXEC.BAT, et on termine la sous-routine (lignes 1752 et 1756–1759).

— Sinon on place le caractère lu dans `AL` et, s'il s'agit de `1Ah`, on place un caractère de fin de lignes (celui de code ASCII 13) dans `AL`, 0 dans la variable BATCH, pour indiquer qu'on a terminé de lire le fichier AUTOEXEC.BAT. Dans tous les cas, on termine la sous-routine (lignes 1753–1759).

6.2.2.4 Partie de prise en compte d'un paramètre

La partie de prise en compte d'un paramètre est NEEDPARM :

```
681 : ;All batch proccessing has DS set to segment of resident portion
682 : ASSUME  DS:RESGROUP
683 : NEEDPARM:
684 :         CALL    GETBATBYT
685 :         CMP     AL,"%"                          ;Check for two consecutive %
686 :         JZ      SAVBATBYT
687 :         CMP     AL,13                           ;Check for end-of-line
688 :         JZ      SAVBATBYT
689 :         SUB     AL,"0"
690 :         JB      RDBAT                           ;Ignore parameter reference if invalid
691 :         CMP     AL,9
692 :         JA      RDBAT
693 :         CBW
694 :         MOV     SI,AX
695 :         SHL     SI,1                            ;Two bytes per entry
696 :         MOV     SI,[SI+OFFSET RESGROUP:PARMTAB] ;Get pointer to corresponding parameter
697 :         CMP     SI,-1                           ;Check if parameter exists
698 :         JZ      RDBAT                           ;Ignore if it doesn't
699 :         MOV     AH,OUTCH
```

```
700 : RDPARM:
701 :         LODSB                                   ;From resident segment
702 :         CMP     AL,0DH                          ;Check for end of parameter
703 :         JZ      RDBAT
704 :         STOSB                                   ;To transient segment
705 :         MOV     DL,AL
706 :         INT     33                              ;Display paramters too
707 :         JMP     SHORT RDPARM
```

— On appelle la sous-routine GETBATBYT pour placer le caractère suivant du fichier AUTOEXEC.BAT dans `AL` (ligne 684).

— S'il s'agit du caractère '%' (on a donc deux '%' consécutifs puisque cette sous-routine a été appelée lorsqu'un '%' a été lu), on va à la partie SAVBATBYT de la routine principale pour le placer sur le tampon COMBUF d'une ligne de commandes (lignes 685–686).

— S'il s'agit du caractère de fin de ligne, de code ASCII 13, on va à la partie SAVBATBYT de la routine principale pour le placer sur le tampon COMBUF d'une ligne de commandes, placer la taille de la ligne de commande dans `DI` et comme premier octet du tampon COMBUF, appeler la sous-routine GETBATBYT pour écarter le second caractère de passage à la ligne (celui de code ASCII 10), confondre le segment des données avec le segment de code et passer à la partie DOCOM, étudiée ci-dessous, de la partie transitoire de l'interpréteur de commande pour exécuter la commande qui vient d'être extraite du fichier AUTOEXEC.BAT (lignes 687–688).

— S'il ne s'agit pas d'un chiffre, on va à la partie RDBAT pour lire le caractère suivant du fichier (lignes 689–692).

— S'il s'agit d'un chiffre, on le place, en tant que mot, dans `SI`, on le multiplie par deux et on fait pointer `SI` sur le paramètre correspondant dans la table PARMTAB (lignes 693–696).

La table des paramètres est PARMTAB :

```
139 : PARMTAB DW      10 DUP(-1)      ;No parameters initially
```

— Si le paramètre n'est pas initialisé, on l'ignore et on va à la partie RDBAT pour lire le caractère suivant du fichier (lignes 697–698).

— Sinon on lit un caractère depuis le segment de la partie résidente de l'interpréteur de commande, on regarde s'il s'agit du caractère de fin des paramètres (`0Dh`). Si ce n'est pas le as, on retourne à la partie étiquetée par RDBAT. Sinon on place le caractère dans la partie transitoire, on le place également dans `DL` et on appelle l'interruption `21h` (`AH` est toujours égal à ? ?) pour l'afficher également à l'écran et on revient à la partie étiquetée par RDPARM pour lire le paramètre suivant (lignes 701–707).

6.2.3 Routine du traitement d'une commande en ligne

Le traitement d'une commande en ligne est effectué par la routine DOCOM :

```
748 : ASSUME DS:TRANGROUP
749 : DOCOM:
750 : ;All segments are local for command line processing
751 :         MOV     AL,10
752 :         CALL    OUT
753 :         MOV     SI,OFFSET TRANGROUP:COMBUF+2
754 :         MOV     DI,OFFSET TRANGROUP:IDLEN
755 :         MOV     AX,2901H        ;Make FCB with blank scan-off
756 :         INT     21H
757 :         CMP     AL,1            ;Check for ambiguous command name
758 :         JZ      BADCOMJ1        ;Ambiguous commands not allowed
759 :         CMP     AL,-1
760 :         JNZ     DRVGD
761 :         JMP     DRVBAD
762 : DRVGD:
763 :         MOV     AL,[DI]
764 :         MOV     [SPECDRV],AL
765 :         MOV     AL," "
766 :         MOV     CX,9
767 :         INC     DI
768 :         REPNE   SCASB           ;Count no. of letters in command name
769 :         MOV     AL,9
770 :         SUB     AL,CL
771 :         MOV     [IDLEN],AL
772 :         MOV     DI,81H
773 :         MOV     CX,0
774 :         PUSH    SI
775 : COMTAIL:
776 :         LODSB
777 :         STOSB                   ;Move command tail to 80H
778 :         CMP     AL,13
779 :         LOOPNZ  COMTAIL
780 :         NOT     CL
781 :         MOV     BYTE PTR DS:[80H],CL
782 :         POP     SI
783 : ;If the command has 0 parameters must check here for
784 : ;any switches that might be present.
785 : ;SI -> first character after the command.
786 :         MOV     [FLAGER],0      ;Set error flag before any calls to switch
787 :         CALL    SWITCH          ;Is the next character a "/"
788 :         MOV     [COMSW],AX
789 :         MOV     DI,FCB
790 :         MOV     AX,2901H
791 :         INT     21H
792 :         MOV     [PARM1],AL      ;Save result of parse
793 :         CALL    SWITCH
794 :         MOV     [ARG1S],AX
795 :         MOV     DI,FCB+10H
796 :         MOV     AX,2901H
797 :         INT     21H             ;Parse file name
798 :         MOV     [PARM2],AL      ;Save result
799 :         CALL    SWITCH
800 :         MOV     [ARG2S],AX
801 :         MOV     AL,[IDLEN]
802 :         MOV     DL,[SPECDRV]
803 :         OR      DL,DL           ;Check if drive was specified
804 :         JZ      OK
805 :         JMP     DRVCHK
806 : OK:     DEC     AL              ;Check for null command
```

```
807 :         JNZ     FNDCOM
808 :         JMP     GETCOM
```

— On va à la ligne suivante de l'écran (lignes 751–752).

— On fait pointer `SI` sur le deuxième octet du tampon COMBUF de la ligne de commande et `DI` sur la variable IDLEN (lignes 753–754).

Les variables IDLEN, ID et COM contiennent le nombre de caractères de l'identificateur de commande, l'identificateur de commande et l'extension de l'identificateur de commande (BAT, COM ou EXE) :

```
234 : IDLEN   DB      1 DUP(?)
235 : ID      DB      8 DUP(?)
236 : COM     DB      3 DUP(?)
```

— On crée un FCB en appelant la fonction `29h` de l'interruption `21h` (lignes 755–756).

Avant d'appeler cette fonction, `DS:SI` doit pointer sur une ligne de commande, `ES:DI` sur la zone de mémoire devant contenir le FCB non ouvert et `AL` contenir l'action à effectuer, ici 1 pour retirer les séparateurs principaux de la ligne de commande se trouvant à l'adresse `DS:SI`.

La ligne de commande est analysée pour en extraire un nom de fichier écrit sous la forme '`d:filename.ext`'. Si on en trouve un, le FCB non ouvert de celui-ci est créé à l'emplacement `ES:DI`. Si aucun lecteur de disquette n'est spécifié, on doit comprendre qu'il s'agit du lecteur par défaut. Si aucune extension n'est spécifiée, on place des espaces dans le champ 'extension du nom de fichier' du FCB.

Après exécution de la fonction, `DS:SI` pointe sur le premier caractère suivant le nom de fichier et `ES:Dl` sur le premier octet du FCB. Si le nom de fichier n'est pas valide, `ES:DI` + 1 contient un espace.

— Si le code d'erreur renvoyé *via* `AL` est `01h`, indiquant que le caractère '*' apparaît dans le nom de fichier, il y a une erreur car le nom de la commande ne doit pas être ambigu, ce que l'on indique à l'utilisateur (lignes 757–758).

L'étiquette BADCOMJ1 renvoie à la partie BADCOM d'affichage d'un message d'erreur et attendant une commande valide :

```
157  : BADNAM  DB      "Bad command or file name",13,10,"$"
[...]
720  : BADCOMJ1:JMP    BADCOM
[...]
1001 : BADCOM:
1002 :         MOV     DX,OFFSET TRANGROUP:BADNAM
1003 : ERROR:
1004 :         MOV     AH,PRINTBUF
1005 :         INT     21H
1006 :         JMP     COMMAND
```

— Si le code d'erreur renvoyé *via* `AL` est `FFh`, indiquant que le numéro de lecteur de disquette n'est pas valide, on indique à l'utilisateur que les spécifications du lecteur de disquette ne sont pas valides et on attend une commande valide (lignes 759–762).

Le message BADDRV est défini ligne 169 :

```
169 : BADDRV  DB      "Invalid drive specification$"
[...]
846 : DRVBAD:
847 :         MOV     DX,OFFSET TRANGROUP:BADDRV
848 :         JMP     ERROR
```

— S'il n'y a pas d'erreur, on place le numéro de lecteur de disquette spécifié dans la variable SPECDRV (lignes 763–764).

La variable SPECDRV contient donc le numéro de lecteur de disquette spécifié :

```
227 : SPECDRV DB      1 DUP(?)
```

— On compte le nombre de caractères de l'identificateur de la commande et on le place dans la variable IDLEN (lignes 765–771).

— On fait pointer `DI` sur l'emplacement `81h`, deuxième octet de la ligne de commande du PSP, on initialise `CX` à zéro, on sauvegarde la valeur de `SI` sur la pile et on place l'identificateur de la commande sur le PSP (lignes 772–779).

— On place la valeur du complément de `CL` sur le premier octet du champ 'ligne de commande' du PSP (lignes 780–781).

— On restaure la valeur de `SI`, qui pointe donc maintenant sur le premier caractère suivant l'identificateur de la commande (lignes 782–785).

— On place le code d'erreur `00h` dans l'indicateur d'erreur FLAGER et on appelle la sous-routine SWITCH, étudiée ci-dessous, pour déterminer les commutateurs de la ligne de commande (lignes 786–787).

La variable FLAGER contient l'indicateur d'erreur :

```
225 : FLAGER  DB      1 DUP(?)
```

— On place les commutateurs, obtenus sous forme codée dans `AX`, dans la variable COMSW (ligne 788).

La variable COMSW contient les commutateurs, sous forme codée, de la commande :

```
222 : COMSW   DW      1 DUP(?)
```

— On fait pointer `DI` sur FCB et on appelle la fonction 29 de l'interruption `21h` pour renseigner le champ 'nom de fichier' du FCB, `DS:SI` indiquant l'adresse du nom de fichier sous le format d'une chaîne de caractères ASCIIZ, `ES:DI` popintant sur le FCB du fichier, `AL` étant égal à 1 pour indiquer que les séparateurs d'en-tête dans la chaîne de caractères doivent être vérifiés (lignes 789–791).

Après exécution de la fonction, `AL` indique le résultat de l'opération (0 pour dire que le nom du fichier n'avait aucun caractère générique, 1 pour indiquer que le nom du fichier avait des caractères génériques et 2 pour indiquer que l'unité de disquette spécifié n'est pas valide), `DS:SI` spécifie l'adresse du premier caractère du nom de fichier dans le tampon de nom de fichier spécifié et `ES:DI` l'adresse du tampon FCB.

La constante FCB est définie ligne 40 :

```
40 : FCB      EQU     5CH
```

— On place le résultat `AL` dans la variable PARM1 (ligne 792).

La variable PARM1 est définie ligne 220 :

```
220 : PARM1   DB      1 DUP(?)
```

— On appelle la routine SWITCH pour déterminer le premier commutateur de la ligne de commande et on place le place, obtenu sous forme codée dans `AX`, dans la variable ARG1S (lignes 793–794).

La variable ARG1S contient le premier commutateur de la ligne de commande :

```
223 : ARG1S   DW      1 DUP(?)
```

— On fait pointer `DI` sur FCB + 10h et on appelle la fonction 29 de l'interruption `21h` pour renseigner le champ 'nom du fichier' du FCB, `DS:SI` pointant sur le nom de fichier sous le format d'une chaîne de caractères ASCIIZ, `ES:DI` sur le FCB du fichier, `AL` étant égal à 1 pour indiquer que les séparateurs d'en-tête dans la chaîne de caractères doivent être vérifiés. On place le résultat obtenu `AL` dans la variable PARM2 (lignes 795–798).

La variable PARM2 est définie ligne 221 :

```
221 : PARM2   DB      1 DUP(?)
```

— On appelle la routine SWITCH pour déterminer le second commutateur de la ligne de commande et on le place, obtenu sous forme codée dans `AX`, dans la variable ARG2S (lignes 799–800).

La variable ARG2S contient le second commutateur de la ligne de commande :

```
224 : ARG2S   DW      1 DUP(?)
```

— On place la taille de l'identificateur de la commande dans `AL` et le lecteur de disquette spécifié dans `DL`. S'il s'agit du lecteur de disquette par défaut, on va à la partie DRVCHK, étudiée ci-dessous, de vérification du lecteur de disquette (lignes 801–805).

— Pour une commande non nulle on va à la partie FNDCOM, d'exćution de la commande, et sinon à la partie GETCOM, partie de la routine principale COMMAND déjà étudiée, pour passer à la commande suivante (lignes 806–808).

6.2.4 Routine de détermination des commutateurs d'une commande

6.2.4.1 Les commutateurs d'une commande

Un **commutateur** (*switch* en anglais) de commande commence toujours par le caractère '/' et permet de modifier l'action par défaut d'une commande.

Par exemple le commutateur '/ ?' permet de lister tous les commutateurs disponibles d'une commande. Pour la commande DIR :

```
C:\>dir /?
Affiche une liste de fichiers et de sous-repertoires dans un repertoire.

DIR [lecteur:][chemin][nom_de_fichier] [/A[[:]attributs]] [/B] [/C] [/D] [/L]
  [/N] [/O[[:]tri]] [/P] [/Q] [/R] [/S] [/T[[:]heure]] [/W] [/X] [/4]

  [lecteur:][chemin][nom_de_fichier]
              Specifie le lecteur, le repertoire et/ou fichiers a lister.

  /A Affiche les fichiers dotes des attributs specifies.
    attributs  D  Repertoires                R  Lecture seule
               H  Cache                      A  Archive
               S  Systeme                    I  Fichiers indexes sans contenu
               L  Points d'analyse           -  Prefixe de negation
  /B    Utilise le format abrege (noms des fichiers).
  /C    Affiche le separateur de milliers pour les tailles de fichiers.
        Ceci est la valeur par defaut. Utilisez /-C pour desactiver l'affichage
        du separateur.
  /D    Sur cinq colonnes avec fichiers tries par colonne.
  /L    Affiche en minuscules.
  /N    Nouveau format longue liste ou les noms de fichiers sont a droite.
  /O    Affiche les fichiers selon un tri specifie.
  tri   N  Nom (alphabetique)         S  Taille (ordre croissant)
        E  Extension (alphabetique)   D  Date et heure (chronologique)
        G  Repertoires en tete        -  Prefixe en ordre indirect
  /P    Arret apres l'affichage d'un ecran d'informations.
  /Q    Affiche le proprietaire du fichier.
  /R    Affiche les flux de donnees alternatifs du fichier.
  /S    Affiche les fichiers d'un repertoire et de ses sous-repertoires.
  /T    Controle le champ heure affiche ou utilise dans le tri.
  heure C  Creation
        A  Dernier acces
        W  Derniere ecriture
  /W    Affichage sur cinq colonnes.
  /X    Affiche les noms courts generes pour les noms de fichier non 8.3 car.
        Ce format est celui de /N avec le nom court insere avant le nom long.
        S'il n'y a pas de nom court, des espaces seront affiches a la place.
  /4    Affiche l'annee sur quatre chiffres.

Les commutateurs peuvent etre preconfigures dans la variable d'environnement
DIRCMD. Pour les ignorer, les prefixer avec un trait d'union. Par exemple /-W.

C:\>
```

6.2.4.2 La routine

La routine de détermination du commutateur suivant présent sur une ligne de commande, dont la position est pointée par `SI`, est SWITCH :

```
810 : RETSW:
811 :         XCHG    AX,BX           ;Put switches in AX
812 :         RET
813 :
814 : SWITCH:
815 :         XOR     BX,BX           ;Initialize - no switches set
816 : SWLOOP:
817 :         CALL    SCANOFF         ;Skip any delimiters
818 :         CMP     AL,"/"          ;Is it a switch specifier?
819 :         JNZ     RETSW           ;No -- we're finished
820 :         INC     SI              ;Skip over "/"
821 :         CALL    SCANOFF
822 :         INC     SI
823 : ;Convert lower case input to upper case
824 :         CMP     AL,"a"
825 :         JB      SAVCHR
826 :         CMP     AL,"z"
827 :         JA      SAVCHR
828 :         SUB     AL,20H          ;Lower-case changed to upper-case
829 : SAVCHR:
830 :         MOV     DI,OFFSET TRANGROUP:SWLIST
831 :         MOV     CX,SWCOUNT
832 :         REPNE   SCASB           ;Look for matching switch
833 :         JNZ     BADSW
834 :         MOV     AX,1
835 :         SHL     AX,CL           ;Set a bit for the switch
836 :         OR      BX,AX
837 :         JMP     SHORT SWLOOP
838 :
839 : BADSW:
840 :         MOV     [FLAGER],1      ;Record error in switch
841 :         JMP     SHORT SWLOOP
842 :
843 : SWLIST  DB      "BAVPW"
844 : SWCOUNT EQU     $-SWLIST
```

— On initialise `BX` pour indiquer qu'on n'a trouvé, pour l'instant, aucun commutateur de commande (ligne 815).

— On appelle la routine SCANOFF, étudiée ci-dessous, pour passer tous les délimiteurs et faire pointer `SI` sur le premier caractère qui n'en est pas un (ligne 817).

— S'il ne s'agit pas du caractère '/', on place le nombre de paramètres dans `AX` et on termine la routine (lignes 818–819 et 810–812).

— Sinon on incrémente `SI` pour passer également ce caractère (nous savons à présent que nous sommes en train d'étudier un commutateur de la commande), on appelle à nouveau la routine SCANOFF pour passer tous les délimiteurs suivants et faire pointer `SI` sur le premier caractère suivant qui n'en est pas un, on incrémente à nouveau `SI` et si ce caractère est une lettre, on le convertit en majuscule (lignes 820–829).

— On fait pointer `DI` sur la liste SWLIST des commutateurs, on place la taille SWLIST de cette liste dans `CX` et on compare le caractère paramètre trouvé avec ceux de cette liste (lignes 830–833).

La liste des commutateurs est SWLIST, sa taille SWCOUNT :

```
843 : SWLIST  DB      "BAVPW"
844 : SWCOUNT EQU     $-SWLIST
```

— Si aucun ne correspond, on place le code d'erreur `01h` dans la variable FLAGER et on recommence pour tester le commutateur suivant (lignes 833 et 839–841).

— Sinon on met le bit correspondant à ce numéro de commutateur à 1 dans `AX` et on recommence tant qu'il y a des commutateurs (lignes 834–837 et 819).

6.2.4.3 Routine de vérification du lecteur de disquette

La routine de vérification du lecteur de disquette est DRVCHK :

```
877 : SETDRV1:
878 :         JMP     SETDRV
879 :
880 : DRVCHK:
881 :         DEC     DL              ;Adjust for correct drive number
882 :         DEC     AL              ;Check if anything else is on line
883 :         JZ      SETDRV1
884 : EXTERNAL:
```

— Le numéro de lecteur de disquette étant passé en paramètres *via* le registre `DL`, on décrémente `DL` (car le numéro du lecteur A est 1 et qu'on veut qu'il soit 0) (ligne 881).

— S'il n'y a rien d'autre sur la ligne de commande, on va à la partie SETDRV, étudiée ci-dessous (lignes 882–883 et 878).

— Sinon on passe à la routine de recherche d'une commande externe, étudiée plus loin (ligne 884).

6.2.4.4 Routine pour ignorer les délimiteurs

La routine SCANOFF passe tous les délimiteurs d'une chaîne de caractères pointée par `SI` et fait pointer `SI` sur le premier caractère qui n'en est pas :

```
1762 : SCANOFF:
1763 :         LODSB
1764 :         CALL    DELIM
1765 :         JZ      SCANOFF
1766 :         DEC     SI              ;Point to first non-delimiter
1767 :         RET
```

— On place le premier caractère de la chaîne de caractères dans `AL` et on appelle la sous-routine DELIM, étudiée ci-dessous, pour savoir s'il s'agit d'un délimiteur, le drapeau de zéro étant levé si c'en est un (lignes 1763–1764).

— S'il s'agit d'un délimiteur, on revient au début pour étudier le caractère suivant de la chaîne de caractères (ligne 1765).

— Sinon on décrémente `SI` pour que celui-ci pointe sur le premier caractère qui n'est pas un délimiteur et on termine la routine (lignes 1766-1767).

La routine DELIM détermine si le caractère dont le code ASCII est placé dans `AL` est un délimiteur et lève le drapeau de zéro si c'en est un :

```
1769 : DELIM:
1770 :         CMP     AL," "
1771 :         JZ      RET80
1772 :         CMP     AL,"="
1773 :         JZ      RET80
1774 :         CMP     AL,","
1775 :         JZ      RET80
1776 :         CMP     AL,9            ;Check for TAB character
1777 : RET80:  RET
```

6.2.4.5 Routine de sélection du lecteur de disquette

La routine de sélection du lecteur de disquette est SETDRV :

```
637 : SETDRV:
638 :         MOV     AH,SELDRV
639 :         INT     21H
640 : COMMAND:
```

On appelle la fonction de l'interruption `21h` de sélection du lecteur de disquette par défaut, celui-ci étant désigné par le contenu de `DL` (avec 0 pour A et 1 pour B) et on revient au début de la routine de traitement d'une ligne de commande.

Le numéro de la fonction de l'interruption `21h` de sélection du lecteur de disquette par défaut est donné par la constante SELDRV :

```
55 : SELDRV  EQU     14
```

6.2.5 Routine de détermination d'une commande interne

Pour voir si la commande trouvée est une commande interne, on utilise la routine FNDCOM :

```
850 : FNDCOM:
851 :         MOV     SI,OFFSET TRANGROUP:COMTAB  ;Prepare to search command table
852 :         MOV     CH,0
853 : FINDCOM:
854 :         MOV     DI,OFFSET TRANGROUP:IDLEN
855 :         MOV     CL,[SI]
856 :         JCXZ    EXTERNAL
857 :         REPE    CMPSB
858 :         LAHF
859 :         ADD     SI,CX                       ;Bump to next position without affecting flags
860 :         SAHF
861 :         LODSB                               ;Get flag for drive check
862 :         MOV     [CHKDRV],AL
863 :         LODSW                               ;Get address of command
864 :         JNZ     FINDCOM
865 :         MOV     DX,AX
866 :         CMP     [CHKDRV],0
867 :         JZ      NOCHECK
868 :         MOV     AL,[PARM1]
869 :         OR      AL,[PARM2]                  ;Check if either parm. had invalid drive
870 :         CMP     AL,-1
871 :         JZ      DRVBAD
872 : NOCHECK:CALL    DX
873 : COMJMP: JMP     COMMAND
```

— On fait pointer `SI` au début de la table des commandes.

Les routines de service des onze commandes internes de MS-DOS 1.25 sont spécifiées par la table des commandes COMTAB :

```
181 : COMTAB  DB      4,"DIR",1
182 :         DW      OFFSET TRANGROUP:CATALOG
183 :         DB      7,"RENAME",1
184 :         DW      OFFSET TRANGROUP:RENAME
185 :         DB      4,"REN",1
186 :         DW      OFFSET TRANGROUP:RENAME
187 :         DB      6,"ERASE",1
188 :         DW      OFFSET TRANGROUP:ERASE
189 :         DB      4,"DEL",1
190 :         DW      OFFSET TRANGROUP:ERASE
191 :         DB      5,"TYPE",1
192 :         DW      OFFSET TRANGROUP:TYPEFIL
193 :         DB      4,"REM",1
194 :         DW      OFFSET TRANGROUP:COMMAND
195 :         DB      5,"COPY",1
196 :         DW      OFFSET TRANGROUP:COPY
197 :         DB      6,"PAUSE",1
198 :         DW      OFFSET TRANGROUP:PAUSE
199 :         DB      5,"DATE",0
200 :         DW      OFFSET TRANGROUP:DATE
201 :         DB      5,"TIME",0
202 :         DW      OFFSET TRANGROUP:TIME
203 :         DB      0               ;Terminate command table
```

le premier octet de chaque entrée de cette table spécifiant la taille n plus un de l'identificateur de la commande, les n octets suivants le nom de la commande, l'octet suivant le drapeau de vérification du lecteur de disquette, suivi du décalage de la routine de service de la commande.

— On initialise `CH` à zéro, pour que `CX` contienne le mot correspondant à la valeur placée dans l'octet `CL`, on fait pointer `DI` sur la taille IDLEN de l'identificateur de la commande et on place la longueur plus un de la première commande dans `CL` (lignes 852–855).

— On compare un à un les caractères de l'identificateur de la commande et du nom de la première commande de la table. On place le contenu du registre des drapeaux dans `AH`, on passe au champ suivant de l'entrée sans affecter les drapeaux, on restaure la valeur du registre des drapeux, on place dans `AL` le drapeau de vérification du lecteur de disquette, que l'on place dans la variable CHKDRV, on charge l'adresse de la routine de service de la commande, et on recommence avec le nom de commande suivant si les noms ne correspondent pas (lignes 857–864).

La variable CHKDRV contient le drapeau de vérification du lecteur de disquette :

```
55 : SELDRV  EQU     14
```

— Si l'identificateur de commande correspond à l'une des onze commandes internes, on place l'adresse de la routine de service de celle-ci dans `DX`. S'il y a besoin de vérifier le lecteur de disquette, on place les paramètres dans `AL` ; si tous sont levés, on indique à l'utilisateur que la spécification du lecteur de disquette n'est pas valide et on lit la ligne de commande suivante. Sinon on exécute la routine de service de la commande interne trouvée (lignes 865–872).

— Si la commande ne correspond à aucune des onze commandes internes, on va à la partie EXTERNAL, étudiée ci-dessous, de traitement des commandes externes (ligne 856).

6.2.6 Routine de détermination d'une commande externe

Si la commande n'est pas interne, elle doit être externe. On est donc renvoyé à la routine EXTERNAL :

```
875 : BADCOMJ:JMP     BADCOM
[...]
884 : EXTERNAL:
885 :         MOV     AL,[SPECDRV]
886 :         MOV     [IDLEN],AL
887 :         MOV     WORD PTR[COM],4F00H+"C"  ;"CO"
888 :         MOV     BYTE PTR[COM+2],"M"
889 :         MOV     DX,OFFSET TRANGROUP:IDLEN
890 :         MOV     AH,OPEN
891 :         INT     33                       ;Check if command to be executed
892 :         MOV     [FILTYP],AL              ;0 for COM files, -1 for EXE files
893 :         OR      AL,AL
894 :         JZ      EXECUTE
895 :         MOV     WORD PTR[COM],5800H+"E"  ;"EX"
896 :         MOV     BYTE PTR[COM+2],"E"
897 :         INT     33                       ;Check for EXE file
898 :         OR      AL,AL
899 :         JZ      EXECUTE
900 :         MOV     WORD PTR[COM],4100H+"B"  ;"BA"
901 :         MOV     BYTE PTR[COM+2],"T"
902 :         INT     33                       ;Check if batch file to be executed
903 :         OR      AL,AL
904 :         JNZ     BADCOMJ
905 : BATCOM:
```

— On place le lecteur de disquette spécifié par la commande dans la variable IDLEN, les trois caractères 'C', 'O' et 'M' dans la variable COM, on fait pointer `DX` sur la variable IDLEN et on essaie d'ouvrir le fichier `id`.COM, où `id` est l'identificateur de la commande (lignes 885–891).

— Si on arrive à ouvrir le fichier, on place 0 dans la variable FILTYP et on va à la partie EXECUTE, étudiée ci-dessous, pour en faire exécuter le code (lignes 892–894).

La variable FILTYP contient le type d'exécutable : 0 pour un fichier `.com`, `FFh` pour un fichier `.exe` :

```
218 : FILTYP  DB      1 DUP(?)
```

— Sinon on place 'EXE' dans la variable COM et on essaie d'ouvrir le fichier `id`.EXE, où `id` est l'identificateur de la commande. Si on y parvient, on place `FFh` dans la variable FILTYP et on va à la partie EXECUTE pour en faire exécuter le code (lignes 895–899).

— Sinon on place 'BAT' dans la variable COM et on essaie d'ouvrir le fichier `id`.BAT, où `id` est l'identificateur de la commande. Si on y parvient, on va à la partie BATCOM, étudiée ci-dessous, pour effectuer le traitement par lot (lignes 900–905).

— Si on ne peut ouvrir aucun de ces trois fichiers, on va à la partie BADCOM, déjà étudiée, pour indiquer à l'utilisateur que le nom de fichier ou de commande n'est pas valide et revenir à COMMAND pour saisir la commande suivante (lignes 903–904 et 875).

6.2.7 Routine d'exécution d'une commande externe

6.2.7.1 Routine principale : distinction entre `.com` et `.exe`

La routine d'exécution d'une commande externe est, comme nous l'avons vu, EXECUTE :

```
953  : EXECUTE:
954  :         MOV     AX,WORD PTR[IDLEN+16]
955  :         OR      AX,WORD PTR[IDLEN+18]           ;See if zero length
956  :         JZ      BADCOM                          ;If so, error
957  :         XOR     AX,AX
958  :         MOV     WORD PTR[IDLEN+RR],AX
959  :         MOV     WORD PTR[IDLEN+RR+2],AX         ;Set RR field to zero
960  :         INC     AX
961  :         MOV     WORD PTR[IDLEN+RECLEN],AX       ;Set record length field to 1
962  :         MOV     DX,[TPA]
963  :         MOV     BX,DX
964  :         MOV     AH,SETBASE
965  :         INT     21H
966  :         TEST    [FILTYP],-1                     ;Check if file is COM or EXE
967  :         JZ      COMLOAD
968  :         JMP     EXELOAD
```

— Si les dix-septième, dix-huitième, dix-neuvième et vingtième octets du FCB du fichier de la commande sont nuls, c'est que la taille de celui-ci est nulle, on va donc à la partie BADCOM, déjà étudiée, pour indiquer à l'utilisateur que le nom de fichier ou de commande n'est pas valide et revenir à COMMAND pour saisir la commande suivante (lignes 954–956).

— Sinon on initialise le champ de lecture directe du FCB du fichier à zéro et son champ 'taille d'un enregistrement' à 1 (lignes 957–961).

— On fait pointer `BX` sur la partie transitoire de l'interpréteur de commande et on appelle la fonction de l'interruption `21h` de création d'un nouveau PSP, commençant à cette adresse `BX` (lignes 962–965).

Le numéro de fonction de l'interruption `21h` de création d'un nouveau PSP est donné par la constante SETBASE :

```
42 : SETBASE EQU     38
```

— S'il s'agit d'un fichier `.com`, on va à la partie COMLOAD, étudiée ci-dessous, d'exécution d'un exécutable `.com` (lignes 966–967).

— S'il s'agit d'un fichier `.exe`, on va à la partie EXELOAD, étudiée ci-dessous, d'exécution d'un exécutable `.exe` (lignes 968).

6.2.7.2 Cas d'une commande `.com`

La routine d'exécution d'une commande externe (ou d'un programme) `.com` est COMLOAD :

```
969  : COMLOAD:PUSH    DS
970  :         MOV     DS,DX
971  :         MOV     DX,100H
972  :         MOV     AH,SETDMA
973  :         INT     21H
974  :         POP     DS
975  :         MOV     CX,[BYTCNT]
976  :         SUB     CX,100H
977  :         MOV     DX,OFFSET TRANGROUP:IDLEN
978  :         MOV     AH,RDBLK
979  :         INT     21H
980  :         DEC     AL
981  :         MOV     DX,OFFSET TRANGROUP:TOOBIG
982  :         JNZ     ERROR
983  : ;Set up exit conditions
984  :         MOV     CX,[BYTCNT]
985  :         MOV     DS,BX
986  :         MOV     ES,BX
987  :         CLI
988  :         MOV     SS,BX
989  :         MOV     SP,CX
990  :         STI
991  :         SUB     CX,100H                         ;Allow some stack space
992  :         XOR     AX,AX
993  :         PUSH    AX
994  :         MOV     AX,100H
995  :         PUSH    BX
996  :         PUSH    AX
997  :         CALL    SETUP
998  : XXX     PROC    FAR
999  :         RET
1000 : XXX     ENDP
```

— On sauvegarde sur la pile l'adresse du segment des données, on prend comme segment des données celui de la partie transitoire de l'interpréteur de commande, on initialise `DX` à `100h`, début d'un programme `.com`, situé après le PSP, on appelle la fonction de l'interruption `21h` pour initialiser l'adresse de la zone de transfert à cette adresse, on restaure la valeur de `DS`, on place la taille du fichier dans `CX`, on lui soustrait `100h` et on lit le contenu du fichier que l'on place dans cette zone de transfert (lignes 966–979).

— Si le fichier est trop gros pour tenir en mémoire, on l'indique à l'utilisateur et on revient à COMMAND pour lire la commande suivante (lignes 980–982).

Le message à destination de l'utilisateur est TOOBIG :

```
168 : TOOBIG  DB      "Program too big to fit in memory$"
```

— On place la taille du fichier dans `CX`, on confond le segment des données et le segment supplémentaire avec le segment de code de la commande, on inhibe les interruptions masquables, on place la pile au-dessus du code, on se remet à l'écoute des interruptions masquables, on soustrait `100h` à `CX` pour laisser de la place pour la pile, on place 0 sur la pile, on initialise `AX` à `100h`, on sauvegarde les contenus de `BX` et de `AX` sur la pile et on appelle la sous-routine SETUP, étudiée ci-dessous, pour faire pointer `CX:DX` sur le FCB du fichier de cette commande externe, ce qui termine la routine (lignes 984–1000).

On a effectué un saut à cette routine, on ne l'a pas appelée. Terminer la routine a donc pour conséquence de passer la main au code d'adresse se trouvant au sommet de la pile, c'est-à-dire au début de la partie transitoire de l'interpréteur de commande, qui a été recouvert par le PSP et le programme.

6.2.7.3 Sous-routine de préparation

La sous-routine SETUP de préparation à l'exécution d'une commande externe fait pointer `CX:DX` sur le FCB du fichier de cette commande externe :

```
2140 : SETUP:
2141 :          AND     CL,0F0H                         ;Adjust to even paragraph boundary
2142 :          MOV     AX,WORD PTR DS:[6]              ;Get current memory size
2143 :          SUB     AX,CX                           ;Find out how much we're changing it
2144 :          MOV     WORD PTR DS:[6],CX
2145 :          MOV     CL,4
2146 :          SAR     AX,CL                           ;Convert to a segment address
2147 :          ADD     WORD PTR DS:[8],AX              ;Adjust long jump to go to same place
2148 :          MOV     DX,80H
2149 :          MOV     AH,SETDMA
2150 :          INT     33                              ;Set default disk transfer address
2151 :          MOV     AX,WORD PTR CS:[PARM1]          ;Pass on info about FCBs
2152 :          XOR     CX,CX
2153 :          MOV     DX,CX                           ;Assume no batch file
2154 : ASSUME   CS:RESGROUP
2155 :          TEST    CS:[BATCH],-1                   ;Batch file in progress?
2156 : ASSUME   CS:TRANGROUP
2157 :          JZ      RET120                          ;If not, all set up
2158 :          MOV     CX,CS:[RESSEG]
2159 :          MOV     DX,OFFSET RESGROUP:BATFCB       ;CX:DX points to batch FCB
2160 : RET120: RET
```

— On ajuste `CL` pour se trouver à la frontière d'un nombre pair de paragraphes (ligne 2141).

— On place la taille en cours de la mémoire disponible dans `AX`, on lui soustrait la valeur de `CX` et on place la valeur de `CX` comme nouvelle taille en cours de la mémoire disponible (lignes 2142–2144).

— On convertit la taille trouvée en nombre de paragraphes et on ajuste l'argument du saut long pour qu'il mène au même endroit (lignes 2145–2147).

— On initialise l'adresse de la zone de transfert à `80h` (lignes 2148–2150).

— On place les paramètres de la ligne de commande dans `AX` (ligne 2151).

— S'il y a un traitement par lot en cours, on initialise `CX` et `DX` à zéro et on termine la routine (lignes 2152–2157).

— Sinon, on fait pointer `CX:DX` sur le FCB du fichier et on termine la routine (lignes 2158–2160).

6.2.8 Routine de traitement par lot

La routine d'exécution d'un fichier de traitement par lot est, comme nous l'avons vu, BATCOM :

```
905 : BATCOM:
906 : ;Batch parameters are read with ES set to segment of resident part
907 :         MOV     ES,[RESSEG]
908 : ASSUME  ES:RESGROUP
909 :         MOV     DI,OFFSET RESGROUP:PARMTAB
910 :         MOV     AX,-1
911 :         MOV     CX,10
912 :         REP     STOSW                           ;Zero parameter pointer table
913 :         MOV     SI,OFFSET TRANGROUP:COMBUF+2
914 :         MOV     DI,OFFSET RESGROUP:PARMBUF
915 :         MOV     BX,OFFSET RESGROUP:PARMTAB
916 : EACHPARM:
917 :         CALL    SCANOFF
918 :         CMP     AL,0DH
919 :         JZ      HAVPARM
920 :         MOV     ES:[BX],DI                      ;Set pointer table to point to actual parameter
921 :         INC     BX
922 :         INC     BX
923 : MOVPARM:
924 :         LODSB
925 :         CALL    DELIM
926 :         JZ      ENDPARM                         ;Check for end of parameter
927 :         STOSB
928 :         CMP     AL,0DH
929 :         JZ      HAVPARM
930 :         JMP     SHORT MOVPARM
931 : ENDPARM:
932 :         MOV     AL,0DH
933 :         STOSB                                   ;End-of-parameter marker
934 :         CMP     BX,OFFSET RESGROUP:PARMTAB+20   ;Maximum number of parameters?
935 :         JB      EACHPARM
936 : HAVPARM:
937 :         MOV     SI,OFFSET TRANGROUP:IDLEN
938 :         MOV     DI,OFFSET RESGROUP:BATFCB
939 :         MOV     CX,16
940 :         REP     MOVSW                           ;Move into private batch FCB
941 :         XOR     AX,AX
942 :         PUSH    ES
943 :         POP     DS                              ;Simply batch FCB setup
944 : ASSUME  DS:RESGROUP
945 :         MOV     WORD PTR[BATFCB+RR],AX
946 :         MOV     WORD PTR[BATFCB+RR+2],AX        ;Zero RR field
947 :         INC     AX
948 :         MOV     WORD PTR[BATFCB+RECLEN],AX      ;Set record length to 1 byte
949 :         MOV     [BATCH],AL                      ;Flag batch job in progress
950 :         JMP     COMMAND
```

— On confond le segment supplémentaire avec la partie résidente de l'interpréteur de commande (ligne 907).

— On fait pointer `DI` au début de la table des paramètres, on place `FFh` dans `AX`, 10 dans `CX`, longueur de la table, et on place les paramètres dans la table (lignes 908–912).

— On fait pointer `SI` au début de l'identificateur de la commande dans le tampon d'une ligne de commande, `DI` au début du tampon des paramètres, `BX` au début de la table des paramètres et on appelle la sous-routine SCANOFF pour passer les délimiteurs et que `SI` pointe sur le premier caractère qui n'en est pas un (lignes 913–917).

L'étiquette PARMBUF repère le début de la partie résidente de l'interpréteur de commandes :

```
286 : PARMBUF LABEL   WORD
```

— Si le premier caractère qui n'est pas un délimiteur n'est pas `0Dh`, on a un paramètre :
  — On place dans la table des paramètres le pointeur du paramètre en cours et on incrémente deux fois `BX` (puisque `BX` pointe sur un mot et non un octet) (lignes 918–921).
  — On charge le caractère suivant et on appelle la sous-routine DELIM pour voir s'il s'agit d'un espace, du signe d'égalité, d'une virgule ou d'une tabulation horizontale. Si c'est le cas, c'est la fin du paramètre. Sinon on stocke le caractère chargé et on recommence jusqu'à ce qu'on trouve `0Dh` (lignes 923–930).
  — Lorsqu'on a trouvé la fin du paramètre, on place le marqueur de fin de paramètre `0Dh` et, si on n'est pas arrivé au maximum de paramètres, on recommence pour en trouver éventuellement un autre (lignes 931–935).
— Lorsqu'on a trouvé tous les paramètres, on fait pointer `SI` sur la commande, `DI` sur le FCB initialement utilisé pour le fichier AUTOEXEC.BAT, mais dont on n'a plus besoin, on initialise `CX` à 16 et on recopie les 16 caractères (lignes 936–940).
— On initialise `AX` à zéro, on confond le segment des données avec le segment supplémentaire, on initialise le champ 'lecture directe' du FCB à zéro, le champ 'taille d'un enregistrement' à 1, on place 1 dans la variable BATCH, pour indiquer qu'on est en train d'effectuer un traitement par lot, et on va à la partie COMMAND pour traiter la première commande du traitement par lot (lignes 941–950).

6.2.9 Routine d'exécution d'un exécutable `.exe`

6.2.9.1 Chargement du programme

La routine d'exécution d'un exécutable `.exe` est EXELOAD :

```
2028 : EXELOAD:
2029 :          MOV     AX,CS
2030 :          ADD     AX,LOADSEG
2031 :          MOV     [EXEEND],AX                 ;Store in EXEEND
2032 :          MOV     DX,OFFSET TRANGROUP:RUNVAR  ;Read header in here
2033 :          MOV     AH,SETDMA
2034 :          INT     33
2035 :          MOV     CX,RUNVARSIZ                ;Amount of header info we need
2036 :          MOV     DX,OFFSET TRANGROUP:EXEFCB
2037 :          MOV     AH,RDBLK
2038 :          INT     33                          ;Read in header
2039 :          OR      AL,AL
2040 :          JNZ     BADEXE                      ;Must not reach EOF
2041 :          MOV     AX,[HEADSIZ]                ;Size of header in paragraphs
2042 : ;Convert header size to 512-byte pages by multiplying by 32 & rounding up
2043 :          ADD     AX,31                       ;Round up first
2044 :          MOV     CL,5
2045 :          SHR     AX,CL                       ;Multiply by 32
2046 :          MOV     [EXEFCB+RR],AX              ;Position in file of program
2047 :          MOV     WORD PTR[EXEFCB+RECLEN],512 ;Set record size
2048 :          ADD     BX,10H                      ;First paragraph above parameter area
2049 :          MOV     DX,[PAGES]                  ;Total size of file in 512-byte pages
2050 :          SUB     DX,AX                       ;Size of program in pages
2051 :          MOV     [PSIZE],DX
2052 :          SHL     DX,CL                       ;Convert pages back to paragraphs
2053 :          MOV     AX,DX
2054 :          ADD     DX,BX                       ;Size + start = minimum memory (paragr.)
2055 :          MOV     CX,[EXEEND]                 ;Get memory size in paragraphs
2056 :          CMP     DX,CX                       ;Enough memory?
2057 :          JA      SHRTERR
2058 :          MOV     DX,[INITSP]
2059 :          ADD     DX,15
2060 :          SHR     DX,1
2061 :          SHR     DX,1
2062 :          SHR     DX,1
2063 :          SHR     DX,1
2064 :          ADD     DX,[INITSS]
2065 :          ADD     DX,BX                       ;Adjusted value of SP
2066 :          CMP     DX,CX                       ;Is it valid?
2067 :          JA      SHRTERR
2068 :          CMP     [LOADLOW],-1                ;Load low or high?
2069 :          JZ      LOAD                        ;If low, load at segment BX
2070 :          SUB     CX,AX                       ;Memory size - program size = load addr.
2071 :          MOV     BX,CX
2072 : LOAD:
2073 :          MOV     BP,BX                       ;Save load segment
2074 : LOAD1:
2075 : LOADSEG EQU      (LOAD1-ZERO)/16
2076 :          PUSH    DS
2077 :          MOV     DS,BX
2078 :          XOR     DX,DX                       ;Address 0 in segment
2079 :          MOV     AH,SETDMA
2080 :          INT     33                          ;Set load address
2081 :          POP     DS
2082 :          MOV     CX,[PSIZE]                  ;Number of records to read
2083 :          MOV     DX,OFFSET TRANGROUP:EXEFCB
2084 :          MOV     AH,RDBLK
```

```
2085 :          INT     33                      ;Read in up to 64K
2086 :          SUB     [PSIZE],CX              ;Decrement count by amount read
2087 :          JZ      HAVEXE                  ;Did we get it all?
2088 :          TEST    AL,1                    ;Check return code if not
2089 :          JNZ     BADEXE                  ;Must be zero if more to come
2090 :          ADD     BX,1000H-20H            ;Bump data segment 64K minus one record
2091 :          JMP     SHORT LOAD1             ;Get next 64K block
2092 :
2093 : BADEXE:
2094 :          MOV     DX,OFFSET TRANGROUP:EXEBAD
2095 :          JMP     ERROR
2096 :
2097 : SHRTERR:
2098 :          MOV     DX,OFFSET TRANGROUP:TOOBIG
2099 :          JMP     ERROR
```

— On place l'adresse de fin du code dans la variable EXEEND (lignes 2029–2031).

La variable EXEEND contient l'adresse de fin du code d'un programme `.exe` :

```
242 : EXEEND  DW      1 DUP(?)
```

— On prend comme début de la zone de transfert le début de la zone réservée aux variables d'en-tête d'une programme `.exe` (lignes 2032–2034).

La variable RUNVAR est la première de la zone réservée aux variables d'en-tête d'une programme `.exe` :

```
243 : ;Header variables for EXE file load
244 : ;These are overlapped with COPY variables, below
245 : RUNVAR  LABEL WORD
```

— On place la taille de l'en-tête dans `CX`, on fait pointer `DX` au début du FCB pour un programme `.exe` et on appelle la fonction de l'interruption `21h` de lecture directe de plusieurs enregistrements (lignes 2035–2038).

La taille de l'en-tête d'une programme `.exe` est spécifiée par la constante RUNVARSIZ :

```
260 : RUNVARSIZ       EQU     $-RUNVAR
```

Le FCB pour un programme `.exe` est repéré par EXEFCB :

```
233 : EXEFCB  LABEL WORD
```

— Si on n'est pas parvenu à les lire, on indique à l'utilisateur qu'il y a une erreur dans le fichier et on revient à COMMAND pour lire la commande suivante (lignes 2039–2040 et 2093–2095).

Le message destiné à l'utiliateur est EXEBAD :

```
161 : EXEBAD  DB      "Error in EXE file$"
```

— Si on est parvenu à les lire, on place la taille de l'en-tête dans `AX`, on lui ajoute 31 pour obtenir la partie entière désirée, et on divise par 32 pour obtenir la taille en paragraphes, que l'on place dans le champ RR de lecture directe du FCB du fichier (lignes 2041–2046).

La taille de l'en-tête est le champ HEADSIZ de l'en-tête du fichier `exe` :

```
251 : HEADSIZ DW      1 DUP(?)
252 :         DW      1 DUP(?)
```

— On initialise le champ 'taille d'un enregistrement' du FCB à 512 (ligne 2047).

— On ajoute `10h` à `BX` pour qu'il pointe sur le premier paragraphe se situant après la zone des paramètres (ligne 2048).

— On soustrait `AX` au nombre d'enregistrements (pages) de 512 octets du fichier pour obtenir la taille du programme en nombre de pages, que l'on place dans le champ PSIZE de l'en-tête (lignes 2049–2051).

Les tailles du fichier et du programme en nombre de pages constituent les champs PAGES et PSIZE de l'en-tête du fichier `exe` :

```
248 : PSIZE   LABEL   WORD
249 : PAGES   DW      1 DUP(?)
```

— On sauvegarde dans `AX` la taille du programme en nombre de paragraphes (lignes 2052–2053).

— Si le nombre de paragraphes nécessaire est supérieur à la place disponible en mémoire centrale, on indique à l'utilisateur que le programme est trop gros pour tenir en mémoire et on va à COMMAND pour lire la commande suivante (lignes 2054–2057 et 2097–2099).

— On ajoute 15 à la position initiale du pointeur de pile, que l'on divise par 16, pour obtenir sa taille en paragraphes, on lui ajoute l'adresse initiale du segment de pile, que l'on ajuste. Si la pile ne tient pas, après le code, en mémoire centrale, on indique à l'utilisateur que le programme est trop gros pour tenir en mémoire et on va à COMMAND pour lire la commande suivante (lignes 2058–267).

L'adresse initiale du segment de pile et la position initiale du pointeur de pile constituent les champs INITSS et INITSP de l'en-tête du fichier `exe` :

```
254 : INITSS  DW      1 DUP(?)
255 : INITSP  DW      1 DUP(?)
256 :         DW      1 DUP(?)
```

— Si le programme doit être chargé en partie haute de la mémoire, l'adresse de chargement est la taille de la mémoire moins la taille du programme. Dans les deux cas, chargement en partie haute ou non, on sauvegarde l'adresse de chargement dans `BP` (lignes 2068–2073).

Le champ LOADLOW de l'en-tête du fichier `exe` spécifie si le code doit être chargé en partie basse (il est alors égal à 1) de la mémoire centrale ou en partie haute :

```
253 : LOADLOW DW      1 DUP(?)
```

— On sauvegarde l'adresse du segment des données sur la pile, on initialise l'adresse du segment des données avec l'adresse de chargement, on initialise `DX` avec 0 et on appelle la fonction de l'interruption `21h` d'initialisation de l'adresse de transfert (lignes 2074–2080).

— On restaure l'adresse du segment des données, on place le nombre d'enregistrement à transférer dans `CX`, on fait pointer `DX` sur le FCB du fichier `.exe` et on appelle la fonction de l'interruption `21h` de lecture directe (lignes 2081–2085).

— On soustrait au nombre d'enregistrements à lire le nombre d'enregistrements effectivement lus. Si on tout lu, on va à la partie HAVEXE, étudiée ci-dessous, pour lancer le programme. Sinon on continue à lire (lignes 2086–2091).

6.2.9.2 Fin de la routine

Une fois le programme chargé, la fin de la routine d'exécution d'un exécutable `.exe` est HAVEXE :

```
2101 : HAVEXE:
2102 :          MOV     AX,[RELTAB]                     ;Get position of table
2103 :          MOV     [EXEFCB+RR],AX                  ;Set in random record field
2104 :          MOV     WORD PTR[EXEFCB+RECLEN],1       ;Set one-byte record
2105 :          MOV     DX,OFFSET TRANGROUP:RELPT       ;4-byte buffer for relocation address
2106 :          MOV     AH,SETDMA
2107 :          INT     33
2108 :          CMP     [RELCNT],0
2109 :          JZ      NOREL
2110 : RELOC:
2111 :          MOV     AH,RDBLK
2112 :          MOV     DX,OFFSET TRANGROUP:EXEFCB
2113 :          MOV     CX,4
2114 :          INT     33                              ;Read in one relocation pointer
2115 :          OR      AL,AL                           ;Check return code
2116 :          JNZ     BADEXE
2117 :          MOV     DI,[RELPT]                      ;Get offset of relocation pointer
2118 :          MOV     AX,[RELSEG]                     ;Get segment
2119 :          ADD     AX,BP                           ;Bias segment with actual load segment
2120 :          MOV     ES,AX
2121 :          ADD     WORD PTR ES:[DI],BP             ;Relocate
2122 :          DEC     [RELCNT]                        ;Count off
2123 :          JNZ     RELOC
2124 : ;Set up exit conditions
2125 : NOREL:
2126 :          MOV     AX,[INITSS]
2127 :          ADD     AX,BP
2128 :          CLI
2129 :          MOV     SS,AX                           ;Initialize SS
2130 :          MOV     SP,[INITSP]
2131 :          STI
2132 :          ADD     [INITCS],BP
2133 :          MOV     AX,[TPA]                        ;Get pointer to parameter area
2134 :          MOV     CX,[BYTCNT]                     ;Size of TPA segment
2135 :          MOV     ES,AX
2136 :          MOV     DS,AX                           ;Set segment registers to point to it
2137 :          CALL    SETUP
2138 :          JMP     DWORD PTR CS:[INITIP]           ;Long jump to program
```

— On initialise le champ de lecture directe du FCB du fichier avec l'emplacement de la table de translation, le champ 'taille d'un enregistrement' à 1 et on appelle la fonction de l'interruption `21h` de sélection de l'adresse de transfert (lignes 2102–2107).

Le champ RELTAB de l'en-tête du fichier `exe` spécifie l'emplacement de la table de translation :

```
259 : RELTAB  DW      1 DUP(?)
```

La variable RELPT contient une adresse sur 4 octets (décalage et segment) :

```
246 : RELPT   DW      1 DUP(?)
247 : RELSEG  DW      1 DUP(?)
```

— Si le nombre d'adresses de translation est non nul, on essaie de lire sur le fichier la première adresse de translation, pour la placer dans le tampon. Si on n'y parvient pas, on indique à l'utilisateur qu'il y a une erreur dans le fichier et on revient à COMMAND pour lire la commande suivante (lignes 2108–2116 et 2093–2095).

Le champ RELCNT de l'en-tête d'un fichier `exe` spécifie le nombre d'adresses de translation :

```
250 : RELCNT  DW      1 DUP(?)
```

— Si on y parvient, on place le décalage de celle-ci dans `DI`, son adresse de segment dans `AX`, à laquelle on ajoute la valeur de `BP` pour effectuer la translation, que l'on sauvegarde dans `ES` et on ajoute la valeur de `BP` à `DI` (lignes 2117–2121).

— On décrémente le nombre d'adresses de translation et on recommence pour la suivante si on n'est pas arrivé à zéro (lignes 2122-2123).

— On place l'adresse initiale du segment de pile dans `AX`, à laquelle on ajoute `BP`, on inhibe les interruptions masquables et on initialise le segment de pile avec cette adresse (lignes 2126–2129).

— On initialise le pointeur de pile avec le pointeur initial et on se remet à l'écoute des interruptions masquables (lignes 2130–2131).

— On ajoute `BP` à INITCS pour obtenir l'adresse absolue du point de départ du programme (ligne 2132).

Les champs INITCS et INITIP de l'en-tête d'un fichier `.exe` contiennent le point de départ du programme, décalage et adresse de segment, à la translation près :

```
257 : INITIP  DW      1 DUP(?)
258 : INITCS  DW      1 DUP(?)
```

— On initialise le segment supplémentaire et le segment des données à l'adresse de la zone des paramètres, `CX` avec la longueur de cette zone et on appelle la routine SETUP pour faire pointer `CX:DX` sur le FCB du fichier de la commande (lignes 2133–2137).

— On effectue un saut long vers le point de départ du programme (ligne 2138).

6.3 Les routines de service des commandes internes

Nous venons de voir comment sont appelées les routines de service des onze commandes internes. Étudions les routines de service de ces dernières.

6.3.1 Routine de service de la commande DIR de listage des fichiers

6.3.1.1 Début de la routine de service

Comme nous l'avons vu, la routine associée à la commande DIR commence à l'étiquette CATALOG :

```
1029 : CATALOG:
1030 :          MOV     AL,"?"                          ;*.* is default file spec.
1031 :          MOV     DI,5DH
1032 :          MOV     CX,11
1033 :          REP     STOSB
1034 :          MOV     SI,81H
1035 :          CALL    SWITCH
1036 :          MOV     DI,5CH
1037 :          MOV     AX,41*100H+0DH                  ;Parse with default name and extension
1038 :          INT     33
1039 :
1040 : ;Begin by processing any switches that may have been specified.
1041 : ;BITS will contain any information about switches that was
1042 : ;found when the command line was parsed.
1043 :
1044 : SETSWT:
1045 :          MOV     AX,[COMSW]                      ;Get switches from command
1046 :          OR      AX,[ARG1S]                      ;OR in switches from first parameter
1047 :          MOV     [BITS],AX
1048 :          MOV     BYTE PTR[FULLSCR],LINPERPAG
1049 :          TEST    AL,1                            ;Look for /W
1050 :          MOV     AL,NORMPERLIN
1051 :          JZ      DIR
1052 :          MOV     AL,WIDEPERLIN
1053 : DIR:
1054 :          MOV     [LINLEN],AL                     ;Set number of entries per line
1055 :          MOV     [LINCNT],AL
1056 :          MOV     [FILECNT],0                     ;Keep track of how many files found
1057 :          MOV     DX,OFFSET TRANGROUP:DIRBUF      ;Set Disk transfer address
1058 :          MOV     AH,SETDMA
1059 :          INT     21H
1060 :          MOV     AH,SRCHFRST
1061 : SHOWDIR:
1062 :          MOV     DX,5CH                          ;DX -> Unopened FCB
1063 :          INT     21H                             ;Search for a file to match FCB
1064 :          INC     AL                              ;FF = file not found
1065 :          JNZ     AGAIN                           ;Either an error or we are finished
1066 :          JMP     CHKCNT
1067 : AGAIN:
1068 :          INC     [FILECNT]                       ;Keep track of how many we find
1069 :          MOV     SI,OFFSET TRANGROUP:DIRBUF+1    ;SI -> information returned by sys call
1070 :          CALL    SHONAME
1071 :          TEST    BYTE PTR[BITS],1                ;/W set?
1072 :          JNZ     NEXENT                          ;If so, no size, date, or time
1073 :          CALL    DISPSIZE                        ;Print size of file
1074 :          CALL    TWOSPC
1075 :          MOV     AX,WORD PTR[DIRBUF+25]          ;Get date
1076 :          OR      AX,AX
1077 :          JZ      NEXENT                          ;Skip if no date
```

```
1078 :          MOV     DX,AX
1079 :          MOV     CL,5
1080 :          SHR     AX,CL                   ;Align month
1081 :          AND     AL,0FH
1082 :          MOV     BH,"0"-" "              ;Enable zero suppression
1083 :          CALL    OUT2
1084 :          MOV     AL,"-"
1085 :          CALL    OUT
1086 :          MOV     AL,DL
1087 :          AND     AL,1FH                  ;Mask to day
1088 :          CALL    OUT2
1089 :          MOV     AL,"-"
1090 :          CALL    OUT
1091 :          MOV     AL,DH
1092 :          SHR     AL,1                    ;Align year
1093 :          ADD     AX,80                   ;Relative 1980
1094 :          CMP     AL,100
1095 :          JB      MILLENIUM
1096 :          SUB     AL,100
1097 : MILLENIUM:
1098 :          CALL    OUT2
1099 :          MOV     BX,WORD PTR[DIRBUF+23]  ;Get time
1100 :          OR      BX,BX                   ;Time field present?
1101 :          JZ      NEXENT
1102 :          CALL    TWOSPC
1103 :          SHR     BX,1
1104 :          SHR     BX,1
1105 :          SHR     BX,1
1106 :          SHR     BL,1
1107 :          SHR     BL,1                    ;Hours in BH, minutes in BL
1108 :          MOV     AL,BH
1109 :          MOV     DH,"a"                  ;Assume A.M.
1110 :          CMP     AL,12                   ;In the afternoon?
1111 :          JB      MORN
1112 :          MOV     DH,"p"
1113 :          JE      MORN
1114 :          SUB     AL,12                   ;Keep it to 12 hours or less
1115 : MORN:
1116 :          OR      AL,AL                   ;Before 1 am?
1117 :          JNZ     SHOHOURS
1118 :          MOV     AL,12
1119 : SHOHOURS:
1120 :          MOV     BH,"0"-" "              ;Enable zero suppression
1121 :          CALL    OUT2
1122 :          MOV     AL,":"
1123 :          CALL    OUT
1124 :          MOV     AL,BL                   ;Output minutes
1125 :          CALL    OUT2
1126 :          MOV     AL,DH                   ;Get "a" or "p"
1127 :          CALL    OUT
1128 : NEXENT:
1129 :          DEC     [LINCNT]
1130 :          JNZ     SAMLIN
1131 : NEXLIN:
1132 :          MOV     AL,[LINLEN]
1133 :          MOV     [LINCNT],AL
1134 :          CALL    CRLF2
1135 :          TEST    BYTE PTR[BITS],2        ;/P switch present?
1136 :          JZ      SCROLL                  ;If not, just continue
1137 :          DEC     BYTE PTR[FULLSCR]
1138 :          JNZ     SCROLL
1139 :          MOV     BYTE PTR[FULLSCR],LINPERPAG
```

```
1140 :          MOV     AH,PRINTBUF
1141 :          MOV     DX,OFFSET TRANGROUP:PAUSMES
1142 :          INT     33
1143 :          MOV     AX,0C08H                    ;Wait for any character to be typed
1144 :          INT     21H
1145 :          CALL    CRLF2
1146 : SCROLL:
1147 :          MOV     AH,SRCHNXT
1148 :          JMP     SHOWDIR
1149 :
1150 : SAMLIN:
1151 :          MOV     AL,9                        ;Output a tab
1152 :          CALL    OUT
1153 :          JMP     SHORT SCROLL
```

— On place le caractère ambigu ' ?' dans `AL`, `5Dh` dans `DI`, 11 dans `CX`, nombre de caractères d'un nom de fichier, et on place 11 caractères ambigus ' ?' dans le champ 'nom de fichier' du FCB (lignes 1030–1033).

Prise en compte du commutateur

— On initialise `SI` à `81h`, début du champ 'ligne de commande' du PSP, et on appelle la routine SWITCH pour déterminer les commutateurs éventuels de la commande et les placer, sous forme codée, dans `AX` (lignes 1034–1035).

— On initialise `DI` à `5Ch`, adresse de début du FCB que l'on veut construire, on place l'octet de l'action à effectuer dans `AL`, à savoir `0Dh` pour que le champ 'extension du nom de fichier' du FCB ne soit modifié que si la ligne de commande contient une extension de nom de fichier, le champ 'nom de fichier' du FCB se soit modifié que si la ligne de commande contient un nom de fichier et pour retirer de la ligne de commande les séparateurs principaux. On appelle la fonction 41 de l'interruption `21h` pour construire un nouveau FCB (lignes 1036–1038).

— On place les commutateurs, sous forme codée, de l'instance de commande DIR dans la variable BITS (lignes 1045–1047).

La variable BITS contient les commutateurs, sous forme codée :

```
240 : BITS    DW      1 DUP(?)
```

— On place le nombre de lignes par page écran dans la variable FULLSCR (lignes 1048).

Le nombre de lignes par page écran est spécifié par la constante LINPERPAG :

```
26 : LINPERPAG      EQU     23
```

Le nombre de lignes encore disponibles avant d'arriver en bas de l'ecran est spécifié par la variable FULLSCR :

```
241 : FULLSCR DW      1 DUP(?)
```

— Si le commutateur '/W' est présent sur la ligne de commande, le nombre d'entrées par lignes est 5 , sinon 1 (lignes 1049–1054).

Les constantes NORMPERLIN et WIDEPERLIN spécifient le nombre d'entrées par ligne :

```
27 : NORMPERLIN      EQU     1
28 : WIDEPERLIN      EQU     5
```

Le nombre d'items par ligne est spécifié par la variable LINLEN :

```
231 : LINLEN  DB      1 DUP(?)
```

Recherche de la première entrée du répertoire

— On initialise LINCNT avec le nombre d'entrées par ligne et FILECNT à zéro (lignes 1055-1056).

Le nombre d'items pouvant encore être écrit sur la ligne est spécifié par la variable LINCNT :

```
230 : LINCNT  DB      1 DUP(?)
```

La variable FILECNT contient le nombre d'entrées trouvées dans le répertoire :

```
232 : FILECNT DW      1 DUP(?)
```

— On initialise l'adresse du tampon de transfert à celle du tampon DIRBUF en mémoire centrale pour une entrée du répertoire (lignes 1057–1059).

DIRBUF est un tampon en mémoire pour une entrée du répertoire :

```
239 : DIRBUF  DB      37 DUP(?)
```

— On place `5Ch`, adresse du nouveau FCB, non ouvert, dans `DX` et on recherche la première entrée du répertoire dont le champ 'nom de fichier' concondre avec celui du FCB (lignes 1060–1063).

Le numéro de la fonction de l'interruption `21h` de recherche de la première entrée du répertoire concordant avec un nom de fichier donné, pouvant être ambigu, est donné par la constante SRCHFRST :

```
43 : SRCHFRST EQU    17
```

— Si on n'a trouvé aucune entrée, on va à CHKCNT pour afficher le nombre d'entrées trouvées et terminer la routine (lignes 1064–1066).

Affichage du nom du fichier de l'entrée trouvée

— Par contre, si on a trouvé une entrée, on incrémente le nombre d'entrées trouvées, on fait pointer `SI` sur le nom du fichier dans le tampon DIRBUF de l'entrée du répertoire trouvée et on appelle la routine SHONAME, étudiée ci-dessous, pour afficher à l'écran le nom du fichier trouvé (lignes 1067–1070).

— Si le commutateur '/W' n'est pas présent sur la ligne de commande, on appelle la sous-routine DISPSIZE pour afficher la taille du fichier sur 9 chiffres décimaux, la sous-routine TWOSPC pour afficher deux espaces, on récupère la date du fichier depuis le tampon de l'entrée de répertoire pour la placer dans `AX` et, si elle existe, c'est-à-dire si elle est non nulle, on l'affiche (lignes 1071–1127).

La sous-routine d'affichage de la taille du fichier sur 9 chiffres décimaux est DISPSIZE :

```
1178 : DISPSIZE:
1179 :          MOV     SI,WORD PTR[DIRBUF+29]
1180 :          MOV     DI,WORD PTR[DIRBUF+31]
1181 : DISP32BITS:
```

la sous-routine DISP32BITS d'affichage d'un nombre de 32 bits étant étudiée ci-dessous.

Les sous-routines TWOSPC et ONESPC d'affichage de deux espaces et d'un espace sont définies lignes 1166–1170 :

```
1166 : TWOSPC:
1167 :          CALL    ONESPC
1168 : ONESPC:
1169 :          MOV     AL," "
1170 :          JMP     OUT
```

Recherche des entrées suivantes du répertoire

— Une fois l'affichage de l'entrée trouvée effectué, on décrémente le nombre d'items pouvant encore être écrits sur la ligne et s'il est non nul, on effectue une tabulation horizontale à l'écran et on revient à SHOWDIR pour rechercher l'entrée suivante et la traiter de la même façon jusqu'à ce qu'on ne trouve plus de nouvelle entrée (lignes 1072, 1077, 1101, 1128–1130, 1150–1153 et 1146–1148).

Le numéro de la fonction de l'interruption `21h` de recherche de l'entrée suivante du répertoire concordant avec un nom de fichier donné, pouvant être ambigu, est donné par la constante SRCHNXT :

```
44 : SRCHNXT EQU     18
```

— Si, par contre, il ne reste plus de place sur la ligne pour un nouvel item, on réinitialise LINCNT avec le nombre d'items par ligne et on va à la ligne suivante à l'ecran. Si le commutateur '/P' n'était pas présent sur la ligne de commande, on revient à SHOWDIR pour rechercher l'entrée suivante et la traiter de la même façon jusqu'à ce qu'on ne trouve plus de nouvelle entrée (lignes 1131–1136).

— Si, par contre, le commutateur '/P' était présent sur la ligne de commande, on décrémente le nombre de lignes encore disponibles à l'écran. Si on n'est pas arrivé à la dernière ligne, on revient à SHOWDIR pour rechercher l'entrée suivante et la traiter de la même façon jusqu'à ce qu'on ne trouve plus de nouvelle entrée (lignes 1136–1138).

— Si, par contre, on en est arrivé à la dernière ligne, on réinitialise FULLSCR avec le nombre de lignes par page, on laisse à l'utilisateur le temps de lire le contenu de ce qui vient d'être affiché et on lui demande d'appuyer sur une touche quelconque pour continuer (lignes 1139–1142).

Le message de pause est PAUSMES :

```
170 : PAUSMES DB      "Strike a key when ready . . . $"
```

— Lorsque l'utilisateur a appuyé sur une touche, on va à la ligne (ce qui a pour effet de tout faire défiler vers le haut) et on revient à SHOWDIR pour rechercher l'entrée suivante et la traiter de la même façon jusqu'à ce qu'on ne trouve plus de nouvelle entrée (lignes 1143–1148).

6.3.1.2 Fin de la routine de service

Comme nous l'avons vu dans le début de la routine de service, lorsqu'il n'y a plus d'entrée dans le répertoire, on est renvoyé à la partie CHKCNT, constituant la fin de la routine de service :

```
1003 : ERROR:
1004 :          MOV     AH,PRINTBUF
1005 :          INT     21H
1006 :          JMP     COMMAND
1007 :
1008 : CHKCNT:
1009 :          TEST    [FILECNT],-1
1010 :          JNZ     ENDDIR
1011 :          MOV     DX,OFFSET TRANGROUP:NOTFND
1012 :          JMP     ERROR
1013 :
1014 : ENDDIR:
1015 : ;Make sure last line ends with CR/LF
1016 :          MOV     AL,[LINLEN]
1017 :          CMP     AL,[LINCNT]     ;Will be equal if just had CR/LF
1018 :          JZ      MESSAGE
1019 :          CALL    CRLF2
1020 : MESSAGE:
1021 :          MOV     SI,[FILECNT]
1022 :          XOR     DI,DI
1023 :          CALL    DISP32BITS
1024 :          MOV     DX,OFFSET TRANGROUP:DIRMES
1025 :          MOV     AH,PRINTBUF
1026 :          INT     21H
1027 :          RET
```

— Si le nombre de fichiers indiqué par la variable FILECNT est `FFh`, il y a une erreur (puisque celui-ci ne peut pas dépasser 114). On affiche donc un message d'erreur et on retourne à COMMAND pour attendre la commande suivante (lignes 1008–1012 et 1003–1006).

Le message de fichier non trouvé est NOTFND :

```
160 : NOTFND  DB      "File not found$"
```

— Sinon on s'assure que la dernière ligne se termine par un retour à la ligne. Pour cela, si le contenu des variables LINLEN et LINCNT ne sont pas égaux, on appelle la sous-routine CRLF2 pour ajouter un passage à la ligne (lignes 1010 et 1014–1020).

— On place le nombre d'entrées du répertoire dans `SI`, on initialise `DI` à zéro et on appelle la sous-routine DISP32BITS pour afficher le nombre d'entrées à l'écran, sous forme décimale, en utilisant 9 chiffres, éventuellement complétés à gauche par des espaces (lignes 1020–1023).

— On le fait suivre de 'File(s)' et on termine la routine (lignes 1024–1027).

Le message final est DIRMES :

```
167 : DIRMES  DB      " File(s)$"
```

6.3.1.3 Routine d'affichage d'un nom de fichier

Le registre `SI` pointant sur un nom de fichier (8 caractères constituant le nom proprement dit, suivi de 3 caractères constituant l'extension), la routine SHONAME affiche à l'écran le nom pointé, en séparant le nom et l'extension par un espace :

```
1155 : SHONAME:
1156 :          MOV     CX,8
1157 :          CALL    OUTCNT
1158 :          CALL    ONESPC
1159 :          MOV     CX,3
1160 : OUTCNT:
1161 :          LODSB
1162 :          CALL    OUT
1163 :          LOOP    OUTCNT
1164 :          RET
```

— On initialise `CX` à 8, la longueur du nom du fichier, et on appelle la routine OUTCNT pour afficher un à un les 8 caractères du nom du fichier (lignes 1156–1157).

— On appelle la sous-routine ONESPC pour afficher un espace (ligne 1158).

— On initialise `CX` à 3, la longueur de l'extension du fichier, et on exécute la routine OUTCNT pour afficher un à un les 3 caractères de l'extension du fichier (lignes 1159–1164).

6.3.1.4 Routine d'affichage d'un nombre de 32 bits

La routine DISP32BITS affiche à l'écran le nombre de 32 bits contenu dans `DI:SI` sous forme décimale, en utilisant 9 chiffres, éventuellement complétés à gauche par des espaces :

```
1181 : DISP32BITS:
1182 : ;Prints the 32-bit number DI:SI on the console in decimal. Uses a total
1183 : ;of 9 digit positions with leading blanks.
1184 :          XOR     AX,AX
1185 :          MOV     BX,AX
1186 :          MOV     BP,AX
1187 :          MOV     CX,32
1188 : CONVLP:
1189 :          SHL     SI,1
1190 :          RCL     DI,1
1191 :          XCHG    AX,BP
1192 :          CALL    CONVWRD
1193 :          XCHG    AX,BP
1194 :          XCHG    AX,BX
1195 :          CALL    CONVWRD
1196 :          XCHG    AX,BX
1197 :          ADC     AL,0
1198 :          LOOP    CONVLP
1199 : ; Conversion complete. Print 9-digit number.
1200 :          MOV     CX,1810H        ;Allow leading zero blanking for 8 digits
1201 :          XCHG    DX,AX
1202 :          CALL    DIGIT
1203 :          XCHG    AX,BX
1204 :          CALL    OUTWORD
1205 :          XCHG    AX,BP
1206 : OUTWORD:
```

Conversion

On utilise la méthode de Hörner pour convertir le nombre de 32 bits en 9 chiffres décimaux, codés en BCD, le chiffre le plus à gauche étant placé dans `AH`, les quatre suivants dans `BX` et les quatre derniers dans `BP`.

— On initialise `AX`, `BX` et `BP` à zéro et `CX` à 32, le nombre de bits constituant le nombre (lignes 1184–1187).

— On effectue une rotation des bits de `DI:SI` vers la gauche en réinsérant le bit dans l'indicateur de retenue (CF) ainsi que dans le bit le plus à droite libéré (lignes 1188–1190). Le i-ième bit se trouve donc dans l'indicateur de retenue CF, lors du i-ième passage dans le corps de la boucle contrôlée de 32 étapes

— On sauvegarde le contenu de `AX` dans `BP`, ce qui n'a pas d'importance la première fois mais en a pour les fois suivantes, et on appelle la sous-routine CONVWRD pour l'étape de la méthode de Hörner pour le i-ième bit (lignes 1188–1192).

— On replace l'ancienne valeur de `AX` depuis `BP`, on la sauvegarde dans `BX` et on appelle la sous-routine CONVWRD toujours pour l'étape de la méthode de Hörner pour le i-ième bit (lignes 1193–1195).

— On échange les contenus de `AX` et `BX`, on ajoute la retenue éventuelle à `AL` (lignes 1196–1197).

— On recommence 31 fois depuis CONVLP pour traiter tous les bits (ligne 1198).

Affichage

Une fois la conversion achevée, on affiche le nombre de 9 chiffres décimaux, sous forme BCD, le chiffre le plus à gauche étant placé dans `CH`, les quatre suivants dans `BX` et les quatre derniers dans `BP`.

— On place 24 dans `CL` et 16 dans `CH`, on échange les contenus de `DX` et `AX` et on appelle la sous-routine DIGIT pour afficher le premier chiffre, celui de droite, du nombre à neuf chiffres décimaux (lignes 1199–1202).

— On échange les contenus de `AX` et `BX` et on appelle la sous-routine OUTWORD pour afficher les quatre chiffres suivants (lignes 1203–1204).

— On échange les contenus de `AX` et `BP` et on exécute la sous-routine OUTWORD pour afficher les quatre derniers chiffres et terminer la rotine (lignes 1205–1206).

Routine de conversion.- La routine de conversion est CONVWRD :

```
1232 : CONVWRD:
1233 :         ADC     AL,AL
1234 :         DAA
1235 :         XCHG    AL,AH
1236 :         ADC     AL,AL
1237 :         DAA
1238 :         XCHG    AL,AH
1239 : RET20:  RET
```

— On multiplie `AL` par 2 et on lui ajoute le bit de retenue, ce qui revient à effectuer une étape de la méthode de Hörner :

$$n = 2 \times n + d_i$$

— On corrige après coup les retenues de la manipulation de la valeur DCB effectuée.

— On échange les contenus de `AL` et de `AH` pour effectuer la même chose sur `AH` pour traiter le bit de retenue éventuel, on échange les contenus de `AL` et de `AH` et on termine la routine.

Routine d'affichage d'un chiffre décimal.- La routine d'affichage d'un chiffre décimal, contenu dans DL, est DIGIT :

```
1219 : DIGIT:
1220 :          AND     DL,0FH
1221 :          JZ      BLANKZER
1222 :          MOV     CL,0
1223 : BLANKZER:
1224 :          DEC     CH
1225 :          AND     CL,CH
1226 :          OR      DL,30H
1227 :          SUB     DL,CL
1228 :          MOV     AH,OUTCH
1229 :          INT     21H
1230 :          RET
```

— On ne conserve que le chiffre DCB de poids faible de DL contenant deux chiffres DCB (ligne 1220).

— S'il est non nul on place 0 dans CL (lignes 1221–1223).

— On décrémente CH et on filtre CL avec la nouvelle valeur de CH (lignes 1224–1225).

— On ajoute 30h au nombre contenu dans DL pour obtenir le code ASCII de ce chiffre (ligne 1226).

— On soustrait la valeur de CL à DL pour que ce dernier contienne le code ASCII d'un espace s'il s'agit d'un 0 de tête (ligne 1227).

— On appelle la fonction 2 de l'interruption 21h pour afficher le caractère de code ASCII contenu dans DL à l'écran et on termine la routine (lignes 1228–1230).

Le numéro de la fonction de l'interruption 21h d'affichage d'un caractère à l'écran est donnée par la constante OUTCH :

```
58 : OUTCH   EQU     2
```

Routine d'affichage de quatre chiffres décimaux BCD contenus dans un mot.- Ce mot étant contenu dans AX, la routine est OUTWORD :

```
1206 : OUTWORD:
1207 :          PUSH    AX
1208 :          MOV     DL,AH
1209 :          CALL    OUTBYTE
1210 :          POP     DX
1211 : OUTBYTE:
```

— On sauvegarde le contenu de AX sur la pile, on place son octet de poids fort dans DL et on appelle la sous-routine OUTBYTE pour afficher les deux chiffres de gauche (lignes 1207–1209).

— On restaure la valeur de AX passée en paramètre dans DX, ce qui revient à placer les deux chiffres de droite dans DL, et on passe à OUTBYTE pour les afficher et terminer la routine (lignes 1210–1211).

Routine d'affichage de deux chiffres décimaux BCD contenus dans un octet.- Cet octet étant contenu dans `DL`, la routine est OUTBYTE :

```
1211 : OUTBYTE:
1212 :          MOV     DH,DL
1213 :          SHR     DL,1
1214 :          SHR     DL,1
1215 :          SHR     DL,1
1216 :          SHR     DL,1
1217 :          CALL    DIGIT
1218 :          MOV     DL,DH
1219 : DIGIT:
```

— On sauvegarde le contenu de `DL` dans `DH`, on effectue une rotation de quatre bits à droite sur `DL` pour placer le premier chiffre sur le demi-octet de poids faible et on appelle la sous-routine DIGIT pour l'afficher (lignes 1212–1217).

— On restaure l'ancienne valeur de `DL`, le second chiffre se trouvant sur le demi-octet de poids faible et on passe à la sous-routine DIGIT pour l'afficher et terminer la routine (lignes 1218–1219).

6.3.2 Routine de service de la commande RENAME ou REN de renommage d'un fichier

Comme nous l'avons vu, la routine de service des commandes RENAME et REN, de renommage d'un fichier, est repérée par l'étiquette RENAME (*sic*) :

```
1270 : RENAME:
1271 :          MOV     AH,RENAM
1272 :          MOV     BX,OFFSET TRANGROUP:RENERR
1273 :          CMP     BYTE PTR DS:[FCB+16+1]," "  ;Check if parameter exists
1274 : OPFILE:
1275 :          MOV     DX,OFFSET TRANGROUP:MISNAM
1276 :          JZ      ERRJ                        ;Error if missing parameter
1277 :          MOV     DX,FCB
1278 :          INT     21H
1279 :          INC     AL
1280 :          JNZ     RET20
1281 :          MOV     DX,BX
1282 : ERRJ:    JMP     ERROR
```

— On place dans `AH` le numéro de la fonction de l'interruption `21h` de renommage d'un fichier et on fait pointer `BX` vers un message d'erreur (lignes 1271–1272).

Le numéro de la fonction de l'interruption `21h` de renommage d'un fichier est donné par la constante RENAM :

```
45 : RENAM   EQU     23
```

Le message RENERR d'erreur de renommage est :

```
159 : RENERR  DB      "Duplicate file name or "
160 : NOTFND  DB      "File not found$"
```

— Si la commande ne comprend pas de paramètre, spécifiant le nouveau nom du fichier, on l'indique à l'utilisateur et on retourne à COMMAND pour lire la commande suivante (lignes 1273–1276).

Le message MISNAM d'absence du nouveau nom de fichier est :

```
158 : MISNAM  DB      "Missing file name$"
```

— Sinon on fait pointer `DX` sur le FCB spécifiant le nouveau nom du fichier et on appelle la fonction de l'interruption `21h` de renommage d'un fichier (lignes 1277–1278).

— S'il n'y a pas eu d'erreur, on termine la routine. Sinon on indique à l'utilisateur qu'il s'agit d'un nom de fichier déjà existant ou qu'on n'a pas trouvé le fichier à renommer et on retourne à COMMAND pour lire la commande suivante (lignes 1279–1282).

6.3.3 Routine de service de la commande ERASE ou DEL de suppression d'un fichier

Comme nous l'avons vu, la routine de service des commandes ERASE ou DEL de suppression d'un fichier est repérée par l'étiquette ERASE (*sic*) :

```
1241 : ERASE:
1242 :          MOV     CX,11
1243 :          MOV     SI,FCB+1
1244 : AMBSPEC:
1245 :          LODSB
1246 :          CMP     AL,"?"
1247 :          JNZ     ALLFIL
1248 :          LOOP    AMBSPEC
1249 : ALLFIL:
1250 :          CMP     CX,0
1251 :          JNZ     NOPRMPT
1252 : ASKAGN:
1253 :          MOV     DX,OFFSET TRANGROUP:SUREMES       ;"Are you sure (Y/N)?"
1254 :          MOV     AH,PRINTBUF
1255 :          INT     21H
1256 :          MOV     AX,0C00H+INCHAR
1257 :          INT     21H
1258 :          AND     AL,5FH
1259 :          CMP     AL,"N"
1260 :          JZ      RET20
1261 :          CMP     AL,"Y"
1262 :          CALL    CRLF2
1263 :          JZ      NOPRMPT
1264 :          JMP     SHORT ASKAGN
1265 : NOPRMPT:
1266 :          MOV     AH,DELETE
1267 :          MOV     BX,OFFSET TRANGROUP:NOTFND
1268 :          CMP     BYTE PTR DS:[FCB+1]," "           ;Check if parameter exists
1269 :          JMP     SHORT OPFILE
1270 : RENAME:
1271 :          MOV     AH,RENAM
1272 :          MOV     BX,OFFSET TRANGROUP:RENERR
1273 :          CMP     BYTE PTR DS:[FCB+16+1]," "        ;Check if parameter exists
1274 : OPFILE:
1275 :          MOV     DX,OFFSET TRANGROUP:MISNAM
1276 :          JZ      ERRJ                              ;Error if missing parameter
1277 :          MOV     DX,FCB
1278 :          INT     21H
1279 :          INC     AL
1280 :          JNZ     RET20
1281 :          MOV     DX,BX
1282 : ERRJ:    JMP     ERROR
```

— On initialise `CX` à 11, nombre de caractères d'un nom de fichier, on fait pointer `SI` sur le champ 'nom de fichier' du FCB et on place ce nom de fichier dans le tampon pointé par `DI` (lignes 1242–1248).

— Si ce nom de fichier ne comporte pas le caractère ambigu ' ?', on demande à l'utilisateur s'il est sûr de vouloir supprimer ce fichier et de saisir un caractère. On le saisit, en ayant certainement à l'attendre (lignes 1249–1257).

Le message SUREMES de confirmation de la suppression est :

```
179 : SUREMES DB      "Are you sure (Y/N)? $"
```

— S'il répond 'N' ou 'n', on termine la routine (lignes 1258–1260).

— S'il ne répond pas par 'N', 'n', 'Y' ou 'y', on appelle la sous-routine CRLF2 pour aller à la ligne et on lui pose à nouveau la question (lignes 1261–1264).

— S'il répond 'Y' ou 'y', on place le numéro de fonction de l'interruption `21h` de suppression d'un fichier dans `AH`, on fait pointer `BX` sur le message de fichier non trouvé et, si le nom du fichier n'est pas spécifié, on indique à l'utilisateur qu'il manque le nom du fichier et on va à COMMAND pour lire la commande suivante (lignes 1265–1269 et 1274–1276).

Le numéro de la fonction de l'interruption `21h` de suppression d'un fichier est donné par la constante DELETE :

```
51 : DELETE  EQU     19
```

— Si le nom de fichier est spécifié, on fait pointer `DX` sur le FCB du fichier à supprimer et on appelle la fonction de l'interruption `21h` de suppression d'un fichier (lignes 1277–1278).

— S'il n'y a pas eu d'erreur, on termine la routine. Sinon on indique à l'utilisateur qu'on n'a pas trouvé le fichier à supprimer et on retourne à COMMAND pour lire la commande suivante (lignes 1279–1282).

6.3.4 Routine de service de la commande TYPE d'affichage d'un fichier texte

Comme nous l'avons vu, la routine de service de la commande TYPE d'affichage d'un fichier texte est repérée par l'étiquette TYPEFIL :

```
1284 : TYPEFIL:
1285 :          MOV     DS,[TPA]
1286 :          XOR     DX,DX
1287 :          MOV     AH,SETDMA
1288 :          INT     21H
1289 :          PUSH    CS
1290 :          POP     DS
1291 :          MOV     DX,FCB
1292 :          MOV     AH,OPEN
1293 :          INT     21H
1294 :          OR      AL,AL
1295 :          MOV     DX,OFFSET TRANGROUP:NOTFND
1296 :          JNZ     ERRJ
1297 :          XOR     AX,AX
1298 :          MOV     WORD PTR DS:[FCB+RR],AX         ;Set RR field
1299 :          MOV     WORD PTR DS:[FCB+RR+2],AX
1300 :          INC     AX
1301 :          MOV     WORD PTR DS:[FCB+RECLEN],AX     ;Set record length
1302 :          MOV     ES,[TPA]
1303 : TYPELP:
1304 :          MOV     DX,FCB
1305 :          MOV     CX,[BYTCNT]
1306 :          MOV     AH,RDBLK
1307 :          INT     21H
1308 :          JCXZ    RET30
1309 :          XOR     SI,SI                           ;Start at 0 in TPA
1310 : OUTLP:
1311 :          LODS    BYTE PTR ES:[SI]                ;In TPA segment
1312 :          CMP     AL,1AH
1313 :          JZ      RET30
1314 :          MOV     AH,OUTCH
1315 :          MOV     DL,AL
1316 :          INT     21H
1317 :          LOOP    OUTLP
1318 :          JMP     SHORT TYPELP
1319 :
```

```
1320 : RET30:  RET                                    ;Need a nearby RET
```

— On prend comme segment des données celui de la partie transitoire de l'interpréteur de commande (ligne 1285).

— On initialise `DX` à zéro et on appelle la fonction de l'interruption `21h` de sélection de la zone de transfert (lignes 1286–1288).

— On confond le segment des données avec le segment de code, on fait pointer `DX` sur le FCB et on appelle la fonction de l'interruption `21h` d'ouverture du fichier dont le contenu est à afficher (lignes 1289–1293).

— Si on n'y parvient pas, on indique à l'utilisateur qu'on n'a pas trouvé le fichier et on retourne à COMMAND pour lire la commande suivante (lignes 1294–1296).

— On initialise le champ 'lecture directe' du FCB à zéro, le champ 'taille d'un enregistrement' à un, on prend comme segment supplémentaire celui de la partie transitoire de l'interprétur de commande, on fait pointer `DX` sur le FCB du fichier dont le contenu est à afficher, on place la taille du fichier dans `CX` et on appelle la fonction de l'interruption `21h` de lecture de plusieurs enregistrements (lignes 1297–1307).

— Si on est parvenu à tout lire, on termine la routine (ligne 1308).

— Sinon on initialise `SI` à zéro et on charge un caractère dans la zone transitoire. S'il s'agit de la fin de fichier, on termine la routine. Sinon on affiche le caractère à l'écran et on recommence (lignes 1309–1320).

6.3.5 Routine de service de la commande REM de commentaire

Comme nous l'avons vu, la routine de service de la commande REM, permettant d'ignorer la fin de la ligne, surtout utile évidemment dans un fichier de traitement par lot, est repérée par l'étiquette COMMAND, c'est-à-dire qu'on retourne à COMMAND pour lire la commande suivante.

6.3.6 Routine de service de la commande COPY de copie d'un fichier

6.3.6.1 Routine principale

Comme nous l'avons vu, la routine de service de la commande COPY de copie d'un fichier est repérée par l'étiquette COPY (*sic*!) :

```
1322 : COPY:
1323 :          XOR     AX,AX
1324 :          MOV     [PLUS],AL               ;Will keep track of "+"s
1325 :          MOV     [FILECNT],AX
1326 :          MOV     SI,81H                  ;Point to input line
1327 :          CALL    SWITCH                  ;Skip over switches on command
1328 :          MOV     BP,AX
1329 :          MOV     DI,FCB
1330 :          CALL    PARSNAM                 ;Scan first source
1331 :          MOV     [PARM1],DL              ;Save ambiguous flag
1332 :          MOV     [SRCPT],SI              ;Save pointer to command line
1333 : ;Parse each name to find destination and check for /V switch
1334 : SCANNAM:
1335 :          CALL    PARSE
1336 :          JNZ     SCANNAM
1337 : GETDEST:
1338 :          MOV     DI,OFFSET TRANGROUP:DEST
1339 :          MOV     BX,BP                   ;Remeber switches so far
1340 :          XOR     BP,BP                   ;Must have dest. swtiches alone
1341 :          CALL    PARSNAM
1342 :          MOV     [ARG2S],BP              ;Remember switches on destination
1343 :          JNZ     HAVDESTNAM              ;File name present?
1344 :          INC     DI                      ;Point to file name spot
1345 :          MOV     AL,"?"                  ;Substitute *.*
1346 :          MOV     CX,11
1347 :          REP     STOSB
1348 : HAVDESTNAM:
1349 :          OR      BX,BP                   ;BX = all switches combined
1350 :          AND     BL,VSWITCH              ;Verify requested?
1351 :          JZ      NOVER
1352 :          MOV     AX,46*100H+1            ;Set verify
1353 :          MOV     DL,0
1354 :          INT     33
1355 : NOVER:
1356 :          MOV     DI,OFFSET TRANGROUP:DESTNAME
1357 :          MOV     SI,OFFSET TRANGROUP:DEST+1
1358 :          MOV     BX,FCB+1
1359 :          CALL    BUILDNAME               ;See if we can make it unambiguous
1360 :          MOV     DI,OFFSET TRANGROUP:DESTNAME
1361 :          MOV     AL,"?"
1362 :          MOV     CX,11
1363 :          REPNE   SCASB                   ;Scan for "?" to see if ambiguous
1364 :          MOV     AL,1                    ;Flag if ambig.
1365 :          JZ      AMBIG
1366 :          DEC     AX                      ;AL=0 if unambig.
1367 : AMBIG:
1368 :          MOV     DL,AL
1369 :          MOV     AH,[PLUS]               ;1=found "+"
1370 :          XOR     AL,1                    ;0=ambig, 1=unambig destination
1371 :          AND     AL,[PARM1]              ;Source ambig. AND dest unambig.
1372 :          OR      AL,AH                   ;OR found "+" means concatenation
1373 :          MOV     [ASCII],AL              ;Concatenation implies ASCII mode
1374 :          MOV     [INEXACT],AL            ;ASCII implies inexact copy
1375 :          SHL     AL,1
1376 :          OR      AL,DL                   ;Combine multiple and concat flags
1377 :          MOV     [PARM2],AL
```

```
1378 :          MOV     AL,BYTE PTR[COMSW]
1379 :          CALL    SETASC                          ;Check /A,/B on command
1380 :          MOV     AL,BYTE PTR[ARG1S]
1381 :          CALL    SETASC                          ;Check for ASCII on first filename
1382 :          MOV     BYTE PTR[COMSW],AL              ;Save starting switch values
1383 :          MOV     AH,SRCHFRST
1384 :          CALL    SEARCH                          ;Search for first source name
1385 : MULTDEST:
1386 :          JZ      FIRSTSRC                        ;Find a first source name?
1387 :          TEST    [PARM2],1                       ;If multiple, we're done
1388 :          JNZ     ENDCOPY
1389 :          XOR     AX,AX
1390 :          MOV     [NXTADD],AX
1391 :          MOV     [CFLAG],AL                      ;Flag nothing read yet
1392 : NEXTSNG:
1393 :          MOV     DI,FCB
1394 :          MOV     SI,[SRCPT]
1395 :          CALL    PARSESRC                        ;Parse next file name into FCB
1396 :          MOV     [PARM1],DL                      ;Remember if it's ambiguous
1397 :          MOV     [SRCPT],SI
1398 :          JZ      SNGCLOS
1399 :          MOV     AH,SRCHFRST
1400 :          CALL    SEARCH                          ;Search for new file name
1401 :          JNZ     NEXTSNG                         ;If none, skip it and move to next name
1402 : READSNG:
1403 :          CALL    CHECKREAD
1404 : SNGLOOP:
1405 :          CALL    SEARCHNEXT                      ;See if any more of this name
1406 :          JZ      READSNG
1407 :          JMP     SHORT NEXTSNG
1408 :
1409 : SNGCLOS:
1410 :          CALL    CLOSEFIL
1411 : ENDCOPY:
1412 :          MOV     SI,[FILECNT]
1413 :          XOR     DI,DI
1414 :          CALL    DISP32BITS
1415 :          MOV     DX,OFFSET TRANGROUP:COPIED
1416 :          MOV     AH,PRINTBUF
1417 :          INT     21H
1418 :          JMP     COMMAND                         ;Stack could be messed up
1419 :
1420 : FIRSTSRC:
1421 :          MOV     SI,OFFSET TRANGROUP:DIRBUF+1
1422 :          MOV     DI,OFFSET TRANGROUP:SOURCE
1423 :          MOV     CX,11
1424 :          REP     MOVSB                           ;Copy first source name to SOURCE
1425 :          MOV     SI,OFFSET TRANGROUP:DESTNAME
1426 :          MOV     DI,OFFSET TRANGROUP:DEST+1
1427 :          MOV     BX,OFFSET TRANGROUP:SOURCE
1428 :          CALL    BUILDNAME                       ;Build destination name
1429 :          XOR     AX,AX
1430 :          MOV     [NXTADD],AX
1431 :          MOV     [CFLAG],AL
1432 :          MOV     [APPEND],AL
1433 :          MOV     [NOWRITE],AL
1434 :          TEST    [PARM2],1                       ;Multiple destinations?
1435 :          JZ      NOPRT
1436 :          MOV     SI,OFFSET TRANGROUP:DIRBUF+1
1437 :          CALL    SHONAME                         ;If so, show first source
1438 :          CALL    CRLF2
1439 : NOPRT:
```

```
1440 :          CALL    COMPNAME                        ;Source and dest. the same?
1441 :          JNZ     DOREAD                          ;If not, read source in
1442 :          TEST    [PARM2],2                       ;Concatenation?
1443 :          MOV     DX,OFFSET TRANGROUP:OVERWR
1444 :          JZ      COPERRJ                         ;If not, overwrite error
1445 :          MOV     [APPEND],1                      ;Set physical append
1446 :          MOV     AH,OPEN
1447 :          MOV     DX,OFFSET TRANGROUP:DEST
1448 :          INT     33                              ;Open (existing) destination
1449 :          CMP     [ASCII],0                       ;ASCII flag set?
1450 :          JZ      BINARYAPP
1451 : ;ASCII append. Must find logical EOF, then seek there with dest. FCB
1452 :          MOV     [NOWRITE],1
1453 :          CALL    READIN                          ;Find EOF
1454 :          CALL    FLSHFIL                         ;Seek there
1455 :          MOV     [NOWRITE],0
1456 :          CALL    FLSHFIL                         ;Truncate file
1457 :          JMP     SHORT SNGLCHK
1458 :
1459 : SNGLOOPJ:JMP     SNGLOOP
1460 :
1461 : COPERRJ:JMP     COPERR
1462 :
1463 : BINARYAPP:
1464 :          MOV     WORD PTR[DEST+RECLEN],1         ;Set record length to 1
1465 :          MOV     SI,OFFSET TRANGROUP:DEST+16     ;Point to file size
1466 :          MOV     DI,OFFSET TRANGROUP:DEST+RR
1467 :          MOVSW
1468 :          MOVSW                                   ;Seek to end of file
1469 :          MOV     [CFLAG],1
1470 :          JMP     SHORT SNGLCHK
```

— On initialise les variables PLUS, indiquant qu'il faut effectuer une concaténation, et FILECNT, le nombre de fichiers copiés, à zéro, `SI` à `81h`, c'est-à-dire sur le champ 'ligne de commande' du PSP, et on appelle la sous-routine SWITCH pour déterminer les commutateurs de la commande (lignes 1323–1327).

Un 1 dans la variable PLUS indique qu'il faut effectuer une concaténation :

```
274 : PLUS    DB      1 DUP(?)
```

La variable FILECNT spécifie le nombre de fichiers copiés :

```
232 : FILECNT DW      1 DUP(?)
```

— On initialise `BP` à zéro, on fait pointer `DI` sur le FCB du fichier source et on appelle la sous-routine PARSNAM, étudée ci-dessous, pour y placer le nom du fichier à copier (lignes 1328–1330).

— On sauvegarde l'indicateur d'ambiguïté du nom de fichier dans la variable PARM1 et la position du pointeur sur la ligne de commande dans la variable SRCPT (lignes 1331–1332).

La position du pointeur sur la ligne de commande est spécifiée par la variable SRCPT :

```
269 : SRCPT   DW      1 DUP(?)
```

— On appelle la sous-routine PARSE, étudiée ci-dessous, pour déterminer le nom du fichier de destination (lignes 1333-1335).

— On poursuit l'analyse de la ligne de commande jusqu'à ce qu'on trouve le nom du fichier de destination (ligne 1336).

— On fait pointer `DI` sur le FCB du fichier de destination, on sauvegarde les commutateurs dans `BX`, on initialise `BP` à zéro et on appelle la sous-routine PARSNAM pour placer le nom du fichier de destination dans son FCB (lignes 1337–1341).

Le FCB du fichier de destination est repéré par l'étiquette DEST :

```
237 : DEST    DB      37 DUP(?)
```

— On place les commutateurs trouvés dans la variable ARG2S (ligne 1342).

— Si on n'a pas trouvé le nom du fichier de destination, on incrémente `DI` et on place onze caractères ambigus ' ?' pour celui-ci (lignes 1343–1348).

— On place l'ensemble de tous les commutateurs trouvés, sous forme codée, dans `BX`. Si le commutateur '/V' de vérification est présent, on appelle la fonction 46 de l'interruption `21h` pour demander à ce que soit effectuée une vérification lors de l'écriture dans un fichier (lignes 1349–1355).

La position du commutateur '/V' dans la forme codée des commutateurs est donnée par la constante VSWITCH :

```
628 : VSWITCH EQU     4               ;Verify during COPY
```

— On fait pointer `DI` sur le tampon du nom du fichier de destination, `SI` sur le début du champ 'nom de fichier de destination' du FCB, `BX` sur la première lettre du nom du fichier à copier et on appelle la sous-routine BUILDNAME, étudiée ci dessous, pour remplacer les caractères ambigus du nom du fichier de destination par ceux de même position du nom du fichier à copier (lignes 1356–1359).

Le tampon du nom du fichier de destination est DESTNAME :

```
238 : DESTNAME DB     11 DUP(?)
```

— On place 1 dans `AL` si le nom du fichier de destination est ambigu, 0 sinon (lignes 1360–1367).

— On sauvegarde cette valeur de `AL` dans `DL`, on récupère la valeur de la variable PLUS (égale à 1 si on a trouvé le caractère '+' pour indiquer qu'il faut effectuer une concaténation) dans `AH`, on place 0 dans `AL` si le nom de fichier de destination est ambigu, 0 sinon, puis 1 dans `AL` si les noms des fichiers source et but sont tous les deux ambigus (lignes 1368–1371).

— Si on a trouvé le caractère '+', indiquant qu'il faut effectuer une concaténation, on place 1 dans la variable ASCII (ligne 1373).

La variable booléenne ASCII indique si on a une chaîne de caractères :

```
273 : ASCII   DB      1 DUP(?)
```

— On place également 1 dans la variable INEXACT pour indiquer que le nom du fichier n'est pas exact (ligne 1374).

La variable INEXACT est égale à 1 si le nom de fichier n'est pas totalement défini :

```
270 : INEXACT DB      1 DUP(?)
```

— On décale ce drapeau d'une position à gauche, on l'ajoute aux drapeaux contenus dans `DL` et on sauvegarde le résultat dans la variable PARM2 (lignes 1375–1377).

— On place les commutateurs de la ligne de commande, sous forme codée, dans `AL` et on appelle la sous-routine SETASC, étudiée ci-dessous, pour voir si l'un des commutateurs '/A' ou '/B' est présent (lignes 1378–1379).

— On place le contenu de ARG1S dans `AL` et on appelle la sous-routine SETASC pour voir si l'un des commutateurs '/A' ou '/B' est présent (lignes 1380–1381).

— On sauvegarde les commutateurs de la ligne de commande, sous forme codée, dans la variable COMSW (ligne 1382).

— On appelle la sous-routine SEARCH, étudiée ci-dessous, pour rechercher la première entrée du répertoire dont le nom de fichier concorde avec le nom de fichier source, en sachant qu'iI peut y en avoir plusieurs lorsque le nom de fichier est ambigu (lignes 1383–1384).

— Si on a trouvé une première instance du nom de fichier source dans le répertoire (lignes 1386 et 1420–1438) :
  — On fait pointer `SI` sur le second octet du FCB, c'est-à-dire au début du champ 'nom du fichier', `DI` sur la variable SOURCE, nom du fichier source, on initialise `CX` à 11, nombre de caractères d'un nom de fichier, et on copie le nom du premier fichier source dans la variable SOURCE.

    La variable chaîne de caractères SOURCE contient le nom du fichier source :

```
275 : SOURCE  DB      11 DUP(?)
```

  — On fait pointer `SI` sur le tampon du fichier de destination, `DI` sur le FCB du fichier de destination, `BX` sur le tampon du nom de fichier source et on appelle la sous-routine BUILDNAME pour remplacer les caractères ambigus du nom du fichier de destination par ceux de même position du nom du fichier à copier.
  — On place zéro dans les variables NXTADD, CFLAG, APPEND et NOWRITE.

    La variable CFLAG est un drapeau, vrai si le fichier a été créé, NXTADD donne la position dans la zone de transfert en mémoire centrale, APPEND est un drapeau, vrai si on doit ajouter quelque chose à un fichier, et NOWRITE est un drapeau, vrai si on ne doit pas écrire sur le fichier :

```
226 : CFLAG   DB      1 DUP(?)
[...]
229 : NXTADD  DW      1 DUP(?)
[...]
271 : APPEND  DB      1 DUP(?)
272 : NOWRITE DB      1 DUP(?)
```

  — S'il y a plusieurs instances de fichier de destination, on fait pointer `SI` sur le second octet du FCB, c'est-à-dire sur son champ 'nom du fichier', on appelle la sous-routine SHONAME pour afficher le nom du fichier à l'écran, en séparant le nom et l'extension par un espace, et on passe à la ligne (lignes 1434–1439).
  — On appelle la sous-routine COMPNAME pour déterminer si les noms des fichiers source et destination sont les mêmes. Si ce n'est pas le cas, on va à la sous-routine DOREAD, étudiée ci-dessous, pour lire le fichier source (lignes 1440–1441).
  — Sinon on regarde s'il faut effectuer une concaténation. Si ce n'est pas le cas, on fait pointer `DX` sur un message d'erreur, on affiche ce message et on termine la routine (lignes 1442–1444, 1461, 1733–1736 et 1411–1418).

    Le message d'erreur est OVERWR :

```
164 : OVERWR  DB      "File cannot be copied onto itself",13,10,"$"
```

    Le code de traitement d'une erreur de copie est :

```
1733 : COPERR:
1734 :          MOV     AH,9
1735 :          INT     21H
1736 :          JMP     ENDCOPY
```

  — S'il faut effectuer une concaténation, on place 1 dans la variable APPEND, on ouvre le fichier de destination et si le drapeau ASCII n'est pas levé on va à BINARYAPP, étudié ci-dessous (lignes 1445–1450).
  — S'il s'agit d'une concaténation de fichiers texte, on place 1 dans la variable NOWRITE, on appelle la sous-routine READIN, étudiée ci-dessous, pour ouvrir le fichier source, le lire et y chercher le premier caractère CTRL-Z, dénotant la fin d'un fichier pour MS-DOS, puis on appelle la sous-routine FLSHFIL, étudiée ci dessous, pour écrire toutes les données restant en mémoire et, enfin, on va à SNGLCHK, partie de DOREAD, pour ? ? (lignes 1451–1457).

— Si on n'a pas trouvé le premier nom de fichier source dans le répertoire :
  — S'il n'y a pas de copie multiple à effectuer, on a terminé. On place le nombre de fichiers copiés dans `SI`, on initialise `DI` à zéro, on appelle la sous-routine DISP32BITS d'affichage d'un nombre de 32 bits pour indiquer le nombre de fichiers copiés, on fait pointer `DX` sur le message COPIED, on l'affiche et on retourne à COMMAND pour lire la commande suivante (lignes 1387–1388 et 1411–1418).

    Le message COPIED précise la signification du nombre affiché avant lui :

```
166 : COPIED  DB      " File(s) copied$"
```

  — Sinon on place 0 dans la variable NXTADD, pour indiquer qu'il n'y a plus de copie à effectuer, 0 dans la variable CFLAG pour indiquer qu'on n'a encore rien lu, on fait pointer `DI` sur le FCB du fichier source, on place le pointeur sur la ligne de commande dans `SI` et on appelle la sous-routine PARSESRC, partie de la sous-routine PARSE, pour placer le nom de fichier suivant dans le champ 'nom de fichier' du FCB du fichier source (lignes 1389–1395).
  — On place le contenu de `DL` dans la variable PARM1 et la position atteinte sur la ligne de commande dans la variable SRCPT (lignes 1396–1397).
  — Si ? ?, on appelle la sous-routine CLOSEFIL, étudiée ci-dessous, pour fermer le fichier, puis on place le nombre de fichiers copiés dans `SI`, on initialise `DI` à zéro, on appelle la sous-routine DISP32BITS d'affichage d'un nombre de 32 bits pour indiquer le nombre de fichiers copiés, on fait pointer `DX` sur le message COPIED, on l'affiche et on retourne à COMMAND pour lire la commande suivante (lignes 1398 et 1409–1411).
  — Sinon on place le numéro de l'interruption `21h` de rechercherche de la première entrée du répertoire dans `AH` et on appelle la sous-routine SEARCH pour obtenir le nom de fichier suivant (lignes 1399–1400).
  — S'il n'y en a pas, on recommence pour traiter le nom de fichier source suivant (ligne 1401).
  — Sinon on appelle la sous-routine CHECKREAD, étudiée ci-dessous, pour lire le fichier source si son nom n'est pas identique à celui du fichier de destination (ligne 1403).
  — On appelle la sous-routine SEARCHNEXT, étudiée ci-dessous, pour voir s'il y a d'autres instances de fichier de destination correspondant à ce fichier source (ligne 1405).
  — S'il y en a, on revient à READSNG pour le traiter. Sinon on revient à NEXTSNG pour chercher le nom de fichier source suivant (lignes 1406–1407).
— S'il s'agit de fichiers binaires, on place 1 dans le champ 'taille d'un enregistrement' du FCB de destination, on initialise `SI` avec la taille du fichier, `DI` avec l'emplacement dans le fichier, on déplace les deux mots, on place 1 dans la variable CFLAG et on retourne à SNGLCHK (lignes 1463–1470).

6.3.6.2 Sous-routine de détermination d'un nom de fichier

La sous-routine de détermination d'un nom de fichier est PARSNAM :

```
1503 : PARSNAM:
1504 :          MOV     AX,2901H
1505 :          INT     33                      ;Parse file name
1506 :          CMP     AL,-1                   ;Illegal?
1507 :          MOV     DX,OFFSET TRANGROUP:BADDRV
1508 :          JZ      COPERRJ
1509 :          XCHG    AX,DX                   ;Save parse flag in DL
1510 :          MOV     AL,BYTE PTR[DI]         ;Get drive number
1511 :          OR      AL,AL                   ;Is it default?
1512 :          JNZ     PARSW
1513 :          MOV     AL,[CURDRV]             ;Substitute actual drive
1514 :          INC     AX
1515 :          MOV     BYTE PTR[DI],AL
1516 : PARSW:
1517 :          PUSH    BX
1518 :          PUSH    DI
1519 :          CALL    SWITCH                  ;Process switches
1520 :          OR      BP,AX                   ;Combine all switches
1521 :          CALL    SETASC                  ;Check for /A or /B
1522 :          POP     DI
1523 :          POP     BX
1524 :          CMP     BYTE PTR[DI+1]," "      ;Did we even get a file name?
1525 :          RET
```

— On place `29h` dans `AH` pour spécifier la fonction de l'interruption `21h` de création d'un FCB, `01h` dans `AL` pour indiquer de retirer les séparateurs de la ligne de commande se trouvant en `DS:DI`, `ES:DI` pointant sur la zone de mémoire à laquelle on va créer le FCB et on appelle l'interruption `21h` de création d'un FCB (lignes 1504–1505).

La fonction renvoie le code d'erreur `FFh` dans `AL` si le numéro de lecteur de disquette n'est pas valide.

— Si le numéro de lecteur de disquette n'est pas valide, on fait pointer `DX` sur le message BADDRV et on va en COPERRJ pour afficher le message d'erreur (lignes 1506–1508).

Rappelons que l'étiquette COPERRJ repère une instruction de saut à COPERR :

```
1461 : COPERRJ:JMP      COPERR
```

— Sinon on sauvegarde le drapeau d'analyse dans `DL` et on place le numéro de lecteur de disquette dans `AL` (lignes 1509–1510)

— S'il indique le lecteur de disquette par défaut (à savoir 0), on lui substitue dans le FCB le numéro de lecteur de disquette par défaut (incrémenté pour avoir 1 pour A) (lignes 1511–1516).

— On sauvegarde sur la pile les contenus de `BX` et de `DI` et on appelle la sous-routine SWITCH pour déterminer les commutateurs de la ligne de commande (lignes 1517–1519).

— On combine les commutateurs, sous forme codée et on appelle la sous-routine SETASC pour voir si l'un des commutateurs '/A' ou '/B' est présent (lignes 1520–1521).

— On restaure les contenus de `BX` et de `DI`, on lève le drapeau ZF s'il ne s'agit pas d'un nom de fichier et on termine la routine (lignes 1522-1525).

6.3.6.3 Sous-routine de détermination d'une concaténation de noms de fichiers partiels

La sous-routine est PARSE :

```
1495 : PARSE:
1496 :          MOV     DI,OFFSET TRANGROUP:DIRBUF
1497 : PARSESRC:
1498 :          CALL    SCANOFF
1499 :          CMP     AL,"+"
1500 :          JNZ     RETZF
1501 :          MOV     [PLUS],1                ;Keep track of "+" signs
1502 :          INC     SI                      ;Skip over it
1503 : PARSNAM:
```

— On fait pointer `DI` sur la copie en mémoire centrale du secteur du répertoire et on appelle la sous-routine SCANOFF pour passer les délimiteurs (lignes 1495–1498).

— S'il ne s'agit pas de '+', on n'aura pas à effectuer de concaténation des noms de fichier partiels. On termine donc la routine (lignes 1499–1500)

— S'il s'agit de '+', il faut effectuer la concaténation des noms de fichier partiels. On place 1 dans la variable PLUS pour s'en souvenir, on incrémente `SI` pour passer au caractère suivant et on poursuit par PARSNAM, qui vient juste d'être étudié (lignes 1499–1500).

6.3.6.4 Sous-routine de désambiguïsation d'un nom de fichier

La sous-routine est BUILDNAME :

```
1563 : BUILDNAME:
1564 : ; [SI] = Ambiguous input file name
1565 : ; [BX] = Source of replacement characters
1566 : ; [DI] = Destination
1567 : ; File name is copied from [SI] to [DI]. If "?"s are encountered,
1568 : ; they are replaced with the character in the same position at [BX].
1569 :          MOV     CX,11
1570 : BUILDNAM:
1571 :          LODSB
1572 :          CMP     AL,"?"
1573 :          JNZ     NOTAMBIG
1574 :          MOV     AL,BYTE PTR[BX]
1575 : NOTAMBIG:
1576 :          STOSB
1577 :          INC     BX
1578 :          LOOP    BUILDNAM
1579 :          RET
```

— Avant d'appeler cette routine, `SI` pointe sur un nom de fichier source ambigu, c'est-à-dire contenant le caractère ' ?', `BX` sur un nom de fichier intermédiaire permettant de désambiguïser les caractères ambigus et `DI` sur le nom de fichier de destination. Le nom du fichier est copié en remplaçant chaque instance du caractère ambigu du nom de fichier source par le caractère se trouvant à la même position dans le nom de fichier intermédiaire (lignes 1564–1568).

— On initialise `CX` à 11, nombre de caractères d'un nom de fichier (ligne 1569).

— On regarde les caractères un à un. S'il s'agit du caractère ambigu ' ?', on place le caractère du nom de fichier intermédiaire de la même position dans le nom de fichier de destination. S'il ne s'agit pas du caractère ambigu ' ?', on place le caractère du nom de fichier source dans le nom de fichier de destination (lignes 1571–1579).

6.3.6.5 Sous-routine de détermination du mode ASCII

La sous-routine est SETASC :

```
1547 : SETASC:
1548 : ;Given switch vector in AX,
1549 : ;       Set ASCII switch if /A is set
1550 : ;       Clear ASCII switch if /B is set
1551 : ;       Leave ASCII unchanged if neither or both are set
1552 : ; Also sets INEXACT if ASCII is ever set. AL = ASCII on exit, flags set
1553 :         AND     AL,ASWITCH+BSWITCH
1554 :         JPE     LOADSW                  ;PE means both or neither are set
1555 :         AND     AL,ASWITCH
1556 :         MOV     [ASCII],AL
1557 :         OR      [INEXACT],AL
1558 : LOADSW:
1559 :         MOV     AL,[ASCII]
1560 :         OR      AL,AL
1561 :         RET
```

— Les commutateurs sous forme codée se trouvant dans `AX`, on place 1 dans les variables ASCII et INEXACT si le commutateur '/A' est présent sur la ligne de commande, 0 dans la variable ASCII si le commutateur '/B' est présent sur la ligne de commande et on n'y touche pas sinon. On place le contenu de la variable ASCII dans `AL` (lignes 1548–1552).

— Si `AL` n'indique pas la présence simultanée sur la ligne de commande des deux commutateurs '/A' et '/B', on place la valeur de `AL` dans la variable ASCII et on l'ajoute à la variable INEXACT. (lignes 1553–1558).

Les constantes ASWITCH et BSWITCH spécifient les positions des commutateurs '/A' et '/B' dans la forme codée des commutateurs :

```
629 : ASWITCH EQU     8               ;ASCII mode during COPY
630 : BSWITCH EQU     10H             ;Binary mode during COPY
```

— On lève le drapeau ZF si le contenu de ASCII est nul et on termine la routine (lignes 1559–1561).

6.3.6.6 Sous-routines de recherche d'une entrée dans le répertoire

La sous-routine SEARCH lève le drapeau ZF si le nom du fichier du FCB `FCB` apparaît dans le répertoire :

```
1536 : SEARCH:
1537 :          PUSH    AX
1538 :          MOV     AH,SETDMA
1539 :          MOV     DX,OFFSET TRANGROUP:DIRBUF
1540 :          INT     33                      ;Put result of search in DIRBUF
1541 :          POP     AX                      ;Restore search first/next command
1542 :          MOV     DX,FCB
1543 :          INT     33                      ;Do the search
1544 :          OR      AL,AL
1545 :          RET
```

— On sauvegarde le contenu de `AX` sur la pile, on place le numéro de la fonction de l'interruption `21h` de sélection de la zone de transfert dans `AH`, on fait pointer `DX` sur le tampon en mémoire centrale d'un secteur du répertoire et on appelle l'interruption `21h` pour que ce tampon devienne la zone de transfert (lignes 1537–1540).

— On restaure la valeur de `AX`, et donc en `AH` le numéro de la fonction de l'interruption `21h` soit de recherche de la première entrée du répertoire concordant avec le nom de fichier du FCB (ligne 1399), soit de l'entrée suivante (ligne 1535), on fait pointer `DX` sur le FCB et on appelle l'interruption `21h` pour savoir si le nom du fichier apparaît dans le répertoire. On lève le drapeau ZF si c'est le cas (lignes 1541–1545).

Si le nom est ambigu on utilise la sous-routine SEARCHNEXT pour rechercher l'entrée suivante du répertoire dont le nom de fichier concorde avec le nom ambigu :

```
1531 : SEARCHNEXT:
1532 :          MOV     AL,[PARM1]              ;Is name ambiguous?
1533 :          DEC     AL
1534 :          JNZ     RET35                   ;Don't perform search if not
1535 :          MOV     AH,SRCHNXT
1536 : SEARCH:
```

Si le nom n'est pas ambigu, on termine la recherche. Sinon on recherche l'entrée de répertoire suivante dont le nom de fichier concorde avec le nom ambigu.

6.3.6.7 Sous-routine de comparaison de deux noms de fichiers

La sous-routine COMPNAME compare deux noms de fichiers, y compris leurs octets d'attribut :

```
1581 : COMPNAME:
1582 :          MOV     SI,OFFSET TRANGROUP:DEST
1583 :          MOV     DI,OFFSET TRANGROUP:DIRBUF
1584 :          MOV     CX,6
1585 :          REPE    CMPSW
1586 :          RET
```

On fait pointer `SI` sur le nom du fichier de destination, `DI` sur le nom du fichier de l'entrée de répertoire et on compare les 12 caractères.

6.3.6.8 Sous-routine de lecture

La routine READIN ouvre et lit le fichier source :

```
1597 : RET40:  RET
1598 :
1599 : READIN:
1600 : ;Open source file and read it in. If memory fills up, flush it out to
1601 : ;destination and keep reading. If /A switch set, chop file at first ^Z.
1602 : ; Inputs/Outputs:
1603 : ;       [NXTADD] has current pointer in buffer
1604 : ;       [CFLAG] <>0 if destination has been created
1605 :
1606 :         MOV     DX,OFFSET TRANGROUP:DIRBUF
1607 :         MOV     AH,OPEN
1608 :         INT     21H
1609 :         OR      AL,AL                   ;Successful open?
1610 :         JNZ     RET40                   ;If not, just ignore it
1611 :         XOR     AX,AX
1612 :         MOV     WORD PTR[DIRBUF+RR],AX
1613 :         MOV     WORD PTR[DIRBUF+RR+2],AX
1614 :         INC     AX
1615 :         MOV     WORD PTR[DIRBUF+RECLEN],AX
1616 : COPYLP:
1617 :         MOV     DX,[NXTADD]
1618 :         MOV     AH,SETDMA
1619 :         PUSH    DS
1620 :         MOV     DS,[TPA]
1621 :         INT     33
1622 :         POP     DS
1623 :         MOV     CX,[BYTCNT]
1624 :         SUB     CX,DX                   ;Compute available space
1625 :         MOV     DX,OFFSET TRANGROUP:DIRBUF
1626 :         MOV     AH,RDBLK                ;Read in source file
1627 :         INT     21H
1628 :         JCXZ    RET40
1629 :         CMP     [ASCII],0
1630 :         JZ      BINREAD
1631 :         MOV     DX,CX
1632 :         MOV     DI,[NXTADD]
1633 :         MOV     AL,1AH
1634 :         PUSH    ES
1635 :         MOV     ES,[TPA]
1636 :         REPNE   SCASB                   ;Scan for EOF
1637 :         POP     ES
1638 :         JNZ     USEALL
1639 :         INC     CX
1640 : USEALL:
1641 :         SUB     DX,CX
1642 :         MOV     CX,DX
1643 : BINREAD:
1644 :         ADD     CX,[NXTADD]
1645 :         MOV     [NXTADD],CX
1646 :         CMP     CX,[BYTCNT]             ;Is buffer full?
1647 :         JB      RET40                   ;If not, we must have found EOF
1648 :         CALL    FLSHFIL
1649 :         JMP     SHORT COPYLP
```

— On fait pointer `DX` sur l'entrée de répertoire, on place dans `AH` le numéro de fonction de l'interruption `21h` d'ouverture d'un fichier et on appelle l'interruption `21h` pour ouvrir le fichier source (lignes 1606–1608).

— Si on n'y parvient pas, on termine la routine (lignes 1609–1610).

— Sinon on place 0 dans le champ RR du FCB, 1 dans son champ 'taille d'un enregistrement', l'adresse suivante dans le tampon de transfert dans `DX`, on sauvegarde l'adresse du segment des données, on prend comme segment des données le segment de la partie transitoire de l'interpréteur de commande, on appelle la fonction de l'interruption `21h` de sélection de la zone de transfert et on restaure la valeur de `DS` (lignes 1611–1622).

— On place le nombre d'octets à lire dans `CX`. En lui soustrayant l'adresse dans le tampon, on obtient dans `CX` la longueur nécessaire (lignes 1623–1624).

— On fait pointer `DX` sur le tampon, on place dans `AH` le numéro de la fonction `21h` de lecture directe de plusieurs enregistrements et on appelle l'interruption `21h` (lignes 1625–1627).

— Si le nombre d'octets restant à lire est nul, on termine la routine (lignes 1628 et 1597).

— Sinon, s'il s'agit d'un fichier texte, on place le nombre d'octets restant à lire dans `DX`, on fait pointer `DI` sur l'adresse en cours dans le tampon, on place `1Ah` (indicateur de fin de fichier) dans `AL`, on sauvegarde l'adresse du segment supplémentaire sur la pile, on prend comme nouveau segment supplémentaire celui de la partie transitoire de l'interpréteur de commande et on recherche l'indicateur de fin de fichier (lignes 1629–1636).

— On revient au segment supplémentaire précédent, si on trouvé l'indicateur de fin de fichier on incrémente `CX`, on soustrait `DX` à `CX` et on place le résultat dans `CX` (lignes 1637–1642).

— Qu'il s'agisse d'un fichier texte ou non, on ajoute la position en cours dans le tampon à `CX` pour obtenir la nouvelle position en cours. Si le tampon est plein, on termine la routine (lignes 1643–1647).

— Sinon on appelle la sous-routine FLSHFIL, étudiée ci-dessous, d'envoi de la fin du tampon et on revient à COPYLP pour continuer la copie (lignes 1648–1649).

6.3.6.9 Sous-routine de lecture

La sous-routine est DOREAD :

```
1459 : SNGLOOPJ:JMP    SNGLOOP
[...]
1471 : DOREAD:
1472 :         CALL    READIN
1473 : SNGLCHK:
1474 :         TEST    [PARM2],1               ;Single or multiple destinations?
1475 :         JZ      SNGLOOPJ
1476 :         MOV     SI,[SRCPT]
1477 : MULTAPP:
1478 :         CALL    PARSE
1479 :         JZ      MULTCLOS
1480 :         PUSH    SI
1481 :         MOV     SI,OFFSET TRANGROUP:DIRBUF+1
1482 :         MOV     DI,SI
1483 :         MOV     BX,OFFSET TRANGROUP:SOURCE
1484 :         CALL    BUILDNAME
1485 :         CALL    CHECKREAD
1486 :         POP     SI
1487 :         JMP     SHORT MULTAPP
1488 : MULTCLOS:
1489 :         CALL    CLOSEFIL
1490 :         MOV     AL,BYTE PTR[COMSW]
1491 :         MOV     [ASCII],AL              ;Restore ASCII flag
1492 :         CALL    SEARCHNEXT
1493 :         JMP     MULTDEST
```

— On appelle la routine READIN pour ouvrir un fichier source et le lire (ligne 1471).

— S'il n'existe qu'un seul fichier de destination, on va à SNGLOOP, partie de la routine principale (lignes 1473–1475 et 1459).

— Sinon on place la position du pointeur sur la ligne de commande dans `SI`, on appelle la sous-routine PARSE pour déterminer le fichier but suivant (lignes 1476–1478).

— S'il en existe un, on sauvegarde la valeur de `SI` sur la pile, on fait pointer `SI` et `DI` sur le champ 'nom de fichier' de l'entrée du répertoire, `BX` sur le nom de fichier source et on appelle la sous-routine BUILDNAME pour désambiguïser le nouveau nom de fichier but (lignes 1479–1484).

— On appelle la sous-routine CHECKREAD, étudiée ci-dessous, pour, si les noms des fichiers source et but ne sont pas identiques, aller à la sous-routine READIN pour ouvrir et lire le fichier source (ligne 1485).

— On restaure la valeur de `SI` et on recommence tant qu'il y a de nouveaux fichiers buts sur lesquels copier le fichier source (lignes 1486–1487).

— On appelle la sous-routine CLOSEFIL, étudiée ci-dessous, pour fermer le fichier (ligne 1489).

— On restaure les commutateurs de la ligne de commande, sous forme codée, dans `AL` et dans la variable ASCII, on appelle la routine SEARCHNEXT de l'entrée suivante du répertoire dont le nom de fichier concorde avec le nom (ambigu) du fichier source et on revient à l'étiquette MULTDEST de la routine principale (lignes 1490–1493).

6.3.6.10 Sous-routine d'enregistrement de la fin du tampon

La sous-routine d'enregistrement de la fin du tampon en mémoire centrale est FLSHFIL (pour *FLuSh FILe*) :

```
1685 : FLSHFIL:
1686 : ;Write out any data remaining in memory.
1687 : ; Inputs:
1688 : ;       [NXTADD] = No. of bytes to write
1689 : ;       [CFLAG] <>0 if file has been created
1690 : ; Outputs:
1691 : ;       [NXTADD] = 0
1692 :
1693 :         MOV     AL,1
1694 :         XCHG    [CFLAG],AL
1695 :         OR      AL,AL
1696 :         JNZ     EXISTS
1697 :         CMP     [NOWRITE],0
1698 :         JNZ     SKPMAK                  ;Don't actually create if NOWRITE set
1699 :         MOV     DX,OFFSET TRANGROUP:DEST
1700 :         MOV     AH,MAKE
1701 :         INT     21H
1702 :         MOV     DX,OFFSET TRANGROUP:FULDIR
1703 :         OR      AL,AL
1704 :         JNZ     COPERR
1705 : SKPMAK:
1706 :         XOR     AX,AX
1707 :         MOV     WORD PTR[DEST+RR],AX
1708 :         MOV     WORD PTR[DEST+RR+2],AX
1709 :         INC     AX
1710 :         MOV     WORD PTR[DEST+RECLEN],AX
1711 : EXISTS:
1712 :         XOR     CX,CX
1713 :         XCHG    CX,[NXTADD]
1714 :         CMP     [NOWRITE],0             ;If NOWRITE set, just seek CX bytes
1715 :         JNZ     SEEKEND
1716 :         XOR     DX,DX
1717 :         PUSH    DS
1718 :         MOV     DS,[TPA]
1719 :         MOV     AH,SETDMA
1720 :         INT     33
1721 :         POP     DS
1722 :         MOV     DX,OFFSET TRANGROUP:DEST
1723 :         MOV     AH,WRBLK
1724 :         INT     21H
1725 :         OR      AL,AL
1726 :         JZ      RET60
1727 :         MOV     DX,OFFSET TRANGROUP:DEST
1728 :         MOV     AH,CLOSE
1729 :         INT     21H
1730 :         MOV     AH,DELETE
1731 :         INT     33
1732 :         MOV     DX,OFFSET TRANGROUP:NOSPACE
1733 : COPERR:
1734 :         MOV     AH,9
1735 :         INT     21H
1736 :         JMP     ENDCOPY
1737 :
1738 : SEEKEND:
1739 :         ADD     WORD PTR[DEST+RR],CX
1740 :         ADC     WORD PTR[DEST+RR+2],0           ;Propagate carry
1741 : RET60:  RET
```

— On place 1 dans `AL` et on l'échange avec le contenu de ma variable CFLAG (lignes 1693–1694).

— Si l'ancienne valeur de CFLAG est nulle, c'est-à-dire si le fichier de destination n'existe pas encore, et si la valeur de la variable NOWRITE est nulle, c'est-à-dire si on peut écrire, on fait pointer `DX` sur le FCB du fichier de destination, on place dans `AH` le numéro de la fonction de l'interruption `21h` de création d'un fichier spécifié par un FCB, on appelle l'interruption `21h` pour créer le fichier de destination. On fait ensuite pointer `DX` sur le message FULDIR et, si on n'est pas parvenu à le créer, on l'indique à l'utilisateur et on va à la partie ENDCOPY de la routine principale (lignes 1695–1704).

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

```
50 : MAKE    EQU     22
```

  Le message FULDIR est :

```
163 : FULDIR  DB      "File creation error",13,10,"$"
```

— On place 0 dans le champ d'opération directe du FCB du fichier de destination, 1 dans son champ 'taille d'un enregistrement', et 0 dans `CX`, valeur que l'on échange avec le contenu de NXTADD (lignes 1705–1713).

— Si on ne doit pas écrire, on place le nombre d'octets dans le champ de lecture directe du FCB du fichier de destination et on termine la routine (lignes 1714–1715 et 1738–1741).

— On place 0 dans `DX`, on sauvegarde sur la pile l'adresse du segment des données, on prend comme segment des données la partie transitoire de l'interpréteur de commande, on place dans `AH` le numéro de la fonction de l'interruption `21h` de sélection de l'adresse de transfert en mémoire centrale, on appelle l'interruption `21h` pour que l'adresse de transfert soit le début de la partie transitoire de l'interpréteur de commande et on restaure la valeur de `DS`. On fait pointer `DX` sur le FCB du fichier de destination, on place dans `AH` le numéro de la fonction de l'interruption `21h` d'écriture directe de plusieurs enregistrements et on appelle l'interruption `21h` pour enregistrer la copie sur le fichier de destination (lignes 1716–1724).

  Le numéro de la fonction de l'interruption `21h` d'écriture directe de plusieurs enregistrements est donné par la constante WRBLK :

```
53 : WRBLK   EQU     40
```

— On termine la routine si on y est parvenu. Sinon on fait pointer `DX` sur le FCB du fichier de destination, on place dans `AH` le numéro de la fonction de l'interruption `21h` de fermeture d'un fichier spécifié par son FCB, on appelle l'interruption `21h` pour fermer le fichier de destination, on place dans `AH` le numéro de la fonction de l'interruption `21h` de suppression d'un fichier spécifié par son FCB, on appelle l'interruption `21h` pour supprimer le fichier de destination, on fait pointer `DX` sur le message de manque de place, on l'affiche et on va à la partie ENDCOPY de la routine principale (lignes 1725–1736).

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

```
49 : CLOSE   EQU     16
```

  Le message de manque de place est NOSPACE :

```
162 : NOSPACE DB      "Insufficient disk space",13,10,"$"
```

6.3.6.11 Sous-routine de fermeture du fichier

La sous-routine est CLOSEFIL :

```
1651 : CLOSEFIL:
1652 :          MOV     AX,[NXTADD]
1653 :          MOV     BX,AX
1654 :          OR      AL,AH                           ;See if any data is loaded
1655 :          OR      AL,[CFLAG]                      ;   or file was created
1656 :          JZ      RET50                           ;Don't close or count if not created
1657 :          MOV     AL,BYTE PTR[ARG2S]
1658 :          CALL    SETASC                          ;Check for /B or /A on destination
1659 :          JZ      BINCLOS
1660 :          CMP     BX,[BYTCNT]                     ;Is memory full?
1661 :          JNZ     PUTZ
1662 :          CALL    FLSHFIL                         ;Empty it to make room for 1 lousy byte
1663 :          XOR     BX,BX
1664 : PUTZ:
1665 :          PUSH    DS
1666 :          MOV     DS,[TPA]
1667 :          MOV     WORD PTR[BX],1AH                ;Add End-of-file mark (Ctrl-Z)
1668 :          POP     DS
1669 :          INC     [NXTADD]
1670 : BINCLOS:
1671 :          CALL    FLSHFIL
1672 :          CMP     [INEXACT],0                     ;Copy not exact?
1673 :          JNZ     NODATE                          ;If so, don't copy date & time
1674 :          MOV     SI,OFFSET TRANGROUP:DIRBUF+OFFDATE
1675 :          MOV     DI,OFFSET TRANGROUP:DEST+OFFDATE   ;Make date & time same as original
1676 :          MOVSW                                   ;Copy date
1677 :          MOVSW                                   ;Copy time
1678 : NODATE:
1679 :          MOV     DX,OFFSET TRANGROUP:DEST
1680 :          MOV     AH,CLOSE
1681 :          INT     21H
1682 :          INC     [FILECNT]
1683 : RET50:  RET
```

— On place la position du prochain octet à mettre dans le tampon de transfert dans `AX` et dans `BX`. Si toutes les données pnt été chargées ou si le fichier de destination a été créé, on termine la routine (lignes 1652–1656 et 1683).

— On place les commuteurs de la ligne de commande, sous forme codée dans `AL` et on appelle la sous-routine SETASC pour savoir si l'un des contrôleurs '/B' ou '/A' est présent pour le fichier de destination (lignes 1657–1659).

— S'il ne s'agit pas d'un fichier texte, on va à BINCLOS. Sinon si la mémoire est pleine, on appelle la sous-routine FLSHFIL pour enregistrer dans le fichier de destination ce qui a été copié depuis le fichier source en mémoire centrale et on place 0 dans `BX`. On sauvegarde sur la pile l'adresse du segment des données, on prend comme segment des données la zone de transfert en mémoire centrale, on place un indicateur de fin de fichier (CTRL-Z, de code ASCII `1Ah`) à l'emplacement pointé par `BX`, on restaure la valeur de `DS` et on incrémente `BX`, puisqu'on vient d'ajouter un caractère (lignes 1659–1670).

— On appelle la sous-routine FLSHFIL pour enregistrer dans le fichier de destination ce qui a été copié depuis le fichier source. Si la copie est exacte, on copie les champs 'date' et 'heure' de l'entrée du répertoire dans le FCB. On ferme le fichier de destination, on incrémente le nombre de fichiers copiés et on termine la routine (lignes 1671–1683).

6.3.6.12 Sous-routine de lecture si les noms de fichier source et but ne sont pas identiques

La sous-routine de lecture si les noms de fichier source et but ne sont pas identiques est CHECKREAD :

```
1588 : CHECKREAD:
1589 : ;Read file in (with READIN) if not identical to destination
1590 :          CALL    COMPNAME                        ;See if source and destination the same
1591 :          JNZ     READIN
1592 :          CMP     [APPEND],0                      ;If physical append, it's OK
1593 :          JNZ     RET40
1594 :          MOV     DX,OFFSET TRANGROUP:LOSTERR     ;Tell him he's not going to get it
1595 :          MOV     AH,PRINTBUF
1596 :          INT     33
1597 : RET40:   RET
```

— On appelle la routine COMPNAME pour comparer les noms des fichiers source et but (ligne 1590).

— Si les deux noms sont différents on va à la routine READIN pour ouvrir et lire le fichier source, en plaçant le contenu en mémoire centrale (ligne 1591).

— Sinon, s'il ne s'agit pas d'ajouter quelque chose, on termine la routine (lignes 1592–1593).

— Sinon on indique à l'utilisateur qu'il y a une erreur (lignes 1594–1597).

Le message d'erreur est LOSTERR :

```
165 : LOSTERR DB      "Content of destination lost before copy",13,10,"$"
```

6.3.7 Routine de service de la commande PAUSE d'insertion d'une pause

Comme nous l'avons vu, la routine de service de la commande PAUSE, d'insertion d'une pause, est repérée par l'étiquette PAUSE :

```
1779 : PAUSE:
1780 :          MOV     DX,OFFSET TRANGROUP:PAUSMES
1781 :          MOV     AH,PRINTBUF
1782 :          INT     33
1783 :          MOV     AX,0C00H+INCHAR ;Get character with KB buffer flush
1784 :          INT     33
1785 : RET90:   RET
```

On demande à l'utilisateur de frapper sur une touche, lorsqu'il aura considéré que la pause est suffisante, et on lit un caractère au clavier, instruction bloquante qui attend que l'utilisateur ait frappé sur une touche.

Le message est PAUSMES :

```
170 : PAUSMES DB      "Strike a key when ready . . . $"
```

6.3.8 Routine de service de la commande DATE de modification ou de lecture de la date

6.3.8.1 Routine principale

Comme nous l'avons vu, la routine de service de la commande DATE, de modification ou de lecture de la date, est repérée par l'étiquette DATE :

```
1785 : RET90:  RET
[...]
1805 : ; DATE - Gets and sets the time
1806 :
1807 : DATE:
1808 :         MOV     SI,81H          ;Accepting argument for date inline
1809 :         CALL    SCANOFF
1810 :         CMP     AL,13
1811 :         JZ      PRMTDAT
1812 :         MOV     BX,2F00H+"-"    ;"/-"
1813 :         CALL    INLINE
1814 :         JMP     COMDAT
1815 :
1816 : PRMTDAT:
1817 :         MOV     DX,OFFSET TRANGROUP:CURDAT
1818 :         MOV     AH,PRINTBUF
1819 :         INT     33              ;Print "Current date is "
1820 :         MOV     AH,GETDATE
1821 :         INT     33              ;Get date in CX:DX
1822 :         CBW
1823 :         MOV     SI,AX
1824 :         SHL     SI,1
1825 :         ADD     SI,AX           ;SI=AX*3
1826 :         ADD     SI,OFFSET TRANGROUP:WEEKTAB
1827 :         MOV     BX,CX
1828 :         MOV     CX,3
1829 :         CALL    OUTCNT
1830 :         MOV     AL," "
1831 :         CALL    OUT
1832 :         MOV     AX,BX
1833 :         MOV     CX,DX
1834 :         MOV     DL,100
1835 :         DIV     DL
1836 :         XCHG    AL,AH
1837 :         XCHG    AX,DX
1838 :         MOV     BL,"-"
1839 :         CALL    SHOW
1840 : GETDAT:
1841 :         MOV     DX,OFFSET TRANGROUP:NEWDAT
1842 :         MOV     BX,2F00H+"-"    ;"/-" in BX
1843 :         CALL    GETBUF
1844 : COMDAT: JZ      RET90
1845 :         JC      DATERR
1846 :         LODSB
1847 :         CMP     AL,BL
1848 :         JZ      SEPGD
1849 :         CMP     AL,BH
1850 :         JNZ     DATERR
1851 : SEPGD:  CALL    GETNUM
1852 :         JC      DATERR
1853 :         MOV     CX,1900
1854 :         CMP     BYTE PTR[SI],13
1855 :         JZ      BIAS
1856 :         MOV     AL,100
```

```
1857 :          MUL     AH
1858 :          MOV     CX,AX
1859 :          CALL    GETNUM
1860 :          JC      DATERR
1861 : BIAS:
1862 :          MOV     AL,AH
1863 :          MOV     AH,0
1864 :          ADD     CX,AX
1865 :          LODSB
1866 :          CMP     AL,13
1867 :          JNZ     DATERR
1868 :          MOV     AH,SETDATE
1869 :          INT     33
1870 :          OR      AL,AL
1871 :          JNZ     DATERR
1872 :          JMP     RET90
1873 : DATERR:
1874 :          MOV     DX,OFFSET TRANGROUP:BADDAT
1875 :          MOV     AH,PRINTBUF
1876 :          INT     33
1877 :          JMP     GETDAT
```

— On fait pointer `SI` sur la ligne de commande après l'identificateur de la commande et on appelle la routine SCANOFF pour placer dans `AL` le premier caractère qui n'est pas un délimiteur (lignes 1808–1809).

Affichage de la date

— S'il s'agit d'un retour à la ligne, l'utilisateur veut voir afficher la date et éventuellement la modifier, on va à PRMTDAT (pour *PRiMiTive DATe*, lignes 1810–1811).

— On affiche '`Current date is `' (lignes 1816–1819).

Le message est CURDAT :

```
174 : CURDAT  DB      "Current date is $"
```

— On appelle la fonction de l'interruption `21h` de récupération de la date (lignes 1820–1821).

Le numéro de la fonction de l'interruption `21h` de récupération de la date est donné par GETDATE :

```
60 : GETDATE EQU     2AH
```

Cette fonction 42 de l'interruption `21h` renvoie la date dans les registres `CX`, `DX` et `AL`, avec `CX` contenant l'année (en binaire dans l'amplitude 1980–2099), `DH` le mois (1 pour janvier, 2 pour février, etc.), `DL` le jour du mois et `AL` le jour de la semaine (0 pour dimanche, 1 pour lundi, etc.).

— On place le jour de la semaine dans `AX`, puis dans `SI`, que l'on multiplie par 3 (car la table WEEKTAB contient trois caractères par nom du jour de la semaine), on fait pointer `SI` dans la table WEEKTAB à la position du jour donné, on sauvegarde l'année dans `BX`, on place 3 dans `CX`, nombre de lettres retenues pour identifier chaque nom du jour de la semaine, et on appelle la sous-routine OUTCNT pour afficher ces trois lettres (lignes 1822–1829).

La table WEEKTAB donne les trois premières lettres de chaque nom du jour de la semaine :

```
172 : WEEKTAB DB      "SunMonTueWedThuFriSat"
```

— On affiche un espace (lignes 1830–1831).

— On place l'année dans `AX`, on sauvegarde le mois et le jour du mois dans `CX`, on divise l'année par 100, dont on place le quotient dans `AL` et le reste dans `AH`, on échange les contenus de `AL` et de `AH`, puis les contenus de `AX` et de `DX`, on place le code ASCII du

caractère '-' dans `BL` et on appelle la sous-routine SHOW, étudiée ci-dessous, pour afficher la date (lignes 1832–1839).

— On fait pointer `DX` sur un message de demande de la nouvelle date, on place '/-' dans `BX` et on appelle la sous-routine GETBUF, étudiée ci-dessous, pour récupérer la date donnée par l'utilisateur (lignes 1840–1843).

Le message est NEWDAT :

```
175 : NEWDAT  DB      13,10,"Enter new date: $"
```

— Si l'utilisateur répond par un passage à la ligne, signifiant qu'il ne veut pas changer la date, on termine la routine (ligne 1844).

Modification de la date

— Si l'utilisateur entre une date erronée, on affiche un message d'erreur et on revient à GETDAT pour demander une nouvelle date à l'utilisateur (lignes 1845 et 1873–1877).

Le message est BADDAT :

```
173 : BADDAT  DB      13,10,"Invalid date$"
```

Demande d'affichage de la date

— S'il ne s'agit pas d'un retour à la ligne, on place '/-' dans `BX` et on appelle la sous-routine INLINE, partie de GETBUF, pour récupérer la date donnée par l'utilisateur et aller en COMDAT (lignes 1810–1814).

6.3.8.2 Routine d'affichage de la date ou de l'heure

La routine SHOW affiche de la date, ou l'heure si BL contient le caractère ':' :

```
1990 : SHOW:
1991 :          MOV     AL,CH
1992 :          MOV     BH,"0"-" "      ;Enable leading zero suppression
1993 :          CALL    OUT2
1994 :          MOV     AL,BL
1995 :          CALL    OUT
1996 :          MOV     AL,CL
1997 :          CALL    OUT2
1998 :          MOV     AL,BL
1999 :          CALL    OUT
2000 :          MOV     AL,DH
2001 :          CALL    OUT2
2002 :          CMP     BL,":"          ;Are we outputting time?
2003 :          JNZ     SKIPIT
2004 :          MOV     AL,"."
2005 :          CALL    OUT
2006 : SKIPIT: MOV      AL,DL
2007 : OUT2:    ;Output binary number as two ASCII digits
```

Avant d'appeler cette sous-routine, DH contient l'année divisée par 100, DL le reste dans la division euclidienne par 100, CL le mois (1 pour janvier, 2 pour février, etc.), CH le jour du mois et BL le code ASCII du caractère '-'. On a affiché le jour de la semaine sous la forme de trois caractères.

— On place le jour du mois dans AL, la différence entre le code ASCII pour le chiffre '0' et celui pour l'espace dans BH, pour remplacer le premier '0' d'un nombre (non nul) de deux chiffres décimaux par un espace, et on appelle la sous-routine OUT2, étudiée ci-dessous, pour afficher ce jour sous la forme de deux chiffres décimaux, en remplaçant le cas échéant le premier zéro par un espace (lignes 1991–1993).

— On place le contenu de BL, à savoir le code ASCII du caractère '-' dans AL, et on appelle la sous-routine OUT pour afficher ce caractère sans affecter les registres (lignes 1994–1995).

— On place le mois dans AL et on appelle la sous-routine OUT2 pour l'afficher sous la forme de deux chiffres décimaux, en remplaçant le cas échéant le premier zéro par un espace (lignes 1996–1997).

— On affiche le caractère '-' (lignes 1998-1999).

— On place l'année divisée par 100 dans AL et on appelle la sous-routine OUT2 pour l'afficher sous la forme de deux chiffres décimaux (lignes 2000–2001).

— Puisque BL ne contient pas le code ASCII du caractère ' :', on place le reste dans la division euclidienne de l'année par 100 dans AL et on passe à OUT2 pour l'afficher sous la forme de deux chiffres décimaux et terminer la routine (lignes 2002–2007).

La sous-routine d'affichage du contenu de `AL` sous la forme de deux chiffres décimaux, en remplaçant le cas échéant le premier zéro par un espace est OUT2 :

```
2007 : OUT2:   ;Output binary number as two ASCII digits
2008 :         AAM                     ;Convert binary to unpacked BCD
2009 :         XCHG    AL,AH
2010 :         OR      AX,3030H        ;Add "0" bias to both digits
2011 :         CMP     AL,"0"          ;Is MSD zero?
2012 :         JNZ     NOSUP
2013 :         SUB     AL,BH           ;Suppress leading zero if enabled
2014 : NOSUP:
2015 :         MOV     BH,0            ;Disable zero suppression
2016 :         CALL    OUT
2017 :         MOV     AL,AH
2018 : OUT:
```

— On convertit le contenu de `AL` en deux chiffres décimaux non compactés dans `AX` (ligne 2008).

— On échange les contenus de `AL` et de `AH` pour que ces deux chiffres soient dans l'ordre naturel (ligne 2009).

— On ajoute le code ASCII de '0' à chacun de ces deux chiffres pour qu'on ait les codes ASCII de ces deux chiffres (ligne 2010).

— Si le chiffre de gauche est '0', on lui ajoute le contenu de `BL` pour que son code ASCII soit remplacé par celui de l'espace (lignes 2011–2014).

— On place 0 dans `BH` pour que cette conversion ne s'effectue plus ensuite (ligne 2015).

— On affiche le chiffre (ou l'espace) de gauche (ligne 2016).

— On affiche le chiffre de droite (lignes 2017–2018).

6.3.8.3 Routine de récupération d'une nouvelle date

La routine de récupération d'une nouvelle date est repérée par l'étiquette GETBUF :

```
1938 : GETBUF:
1939 :          MOV     AH,PRINTBUF
1940 :          INT     33              ;Print "Enter new date: "
1941 :          MOV     AH,INBUF
1942 :          MOV     DX,OFFSET TRANGROUP:COMBUF
1943 :          INT     33              ;Get input line
1944 :          CALL    CRLF2
1945 :          MOV     SI,OFFSET TRANGROUP:COMBUF+2
1946 :          CMP     BYTE PTR[SI],13 ;Check if new date entered
1947 :          JZ      RET100
1948 : INLINE:
1949 :          CALL    GETNUM          ;Get one or two digit number
1950 :          JC      RET100
1951 :          MOV     DH,AH           ;Put in position
1952 :          LODSB
1953 :          CMP     AL,BL
1954 :          JZ      NEXT
1955 :          CMP     BL,":"          ;Is it a date seperator?
1956 :          JNZ     DATESEP
1957 :          DEC     SI
1958 :          MOV     DL,0
1959 : RET100: RET                      ;Time may have only an hour specified
1960 : DATESEP:
1961 :          CMP     AL,BH
1962 :          STC
1963 :          JNZ     RET100
1964 : NEXT:    CALL    GETNUM
1965 :          MOV     DL,AH           ;Put in position
1966 :          RET
```

`BL` contient le code ASCII de '/'.

Requête d'une nouvelle date

— On place dans `AH` le numéro de la fonction de l'interruption `21h` d'affichage d'une chaîne de caractères à l'écran et on appelle l'interruption pour afficher le message pointé par `DX`, à savoir '`Enter new date :`' (lignes 1939–1940).

— On place dans `AH` le numéro de la fonction de l'interruption `21h` de saisie dans un tampon, pointé par `DS:DX`, on fait pointer `DX` sur la variable chaîne de caractères COMBUF et on appelle l'interruption `21h` pour y placer la ligne tapée par l'utilisateur, qui doit être de la forme JJ/MM/AAAA (1941–1943).

Le numéro de la fonction de l'interruption `21h` de saisie dans le tampon pointé par `DS:DX` est donné par INBUF :

```
59 : INBUF   EQU     10
```

— On appelle la routine CRLF2 pour passer à la ligne sur l'écran (ligne 1944.

L'utilisateur n'entre pas de date

— On fait pointer `SI` sur le premier caractère de la ligne tapée par l'utilisateur (rappelons que les deux premiers octets donnent le nombre maximum de caractères de la ligne et le nombre effectif). S'il s'agit d'un passage à la ligne, l'utilisateur n'a rien entré. On termine donc la routine (lignes 1945–1947 et 1959).

Récupération du jour du mois

— On appelle la sous-routine GETNUM pour récupérer le ou les deux chiffres du jour de la semaine (ligne 1949).

— Si la date n'est pas valide, on termine la routine (lignes 1950 et 1959).

— Sinon on place le premier chiffre du jour du mois dans `DH` et on place le caractère suivant dans `AL` (lignes 1951–1952).

— S'il ne s'agit pas du code ASCII du caractère '/', on appelle à nouveau GETNUM et on place le résultat dans `DL` (lignes 1953–1954 et 1964–1966).

La sous-routine INDIG détermine si la valeur pointée par `SI` est le code ASCII d'un chiffre, en place la valeur dans `AL` si c'est le cas, lève le drapeau de retenue CF sinon :

```
1980 : INDIG:
1981 :          MOV     AL,BYTE PTR[SI]
1982 :          SUB     AL,"0"
1983 :          JC      RET110
1984 :          CMP     AL,10
1985 :          CMC
1986 :          JC      RET110
1987 :          INC     SI
1988 :          RET
[...]
1978 : RET110: RET
```

— On place le contenu de `SI`, qui contient le code ASCII d'un chiffre dans `AL`, on lui soustrait le code ASCII du chiffre '0' pour obtenir sa valeur numérique (lignes 1981–1982).

— Si cette valeur n'est pas comprise entre 0 et 9, on lève le drapeau de retenue et on termine la routine sans incrémenter `SI`, de façon à pouvoir lire à nouveau le contenu de cet emplacement (lignes 1983–1986).

— Sinon on incrémente `SI`, puisqu'il n'on a trouvé le code ASCII d'un chiffre, dont la valeur est placée dans `AL` et on termine la routine, drapeau de retenue baissée (lignes 1987–1988).

La sous-routine de récupération d'un nombre de un ou deux chiffres est GETNUM :

```
1968 : GETNUM:
1969 :          CALL    INDIG
1970 :          JC      RET100
1971 :          MOV     AH,AL           ;Save first digit
1972 :          CALL    INDIG           ;Another digit?
1973 :          JC      OKRET
1974 :          AAD                     ;Convert unpacked BCD to decimal
1975 :          MOV     AH,AL
1976 : OKRET:
1977 :          OR      AL,1
1978 : RET110: RET
```

— On appelle la routine pour placer la valeur du premier chiffre, pointé par `SI`, dans `AL`. S'il ne s'agit pas d'un chiffre, on termine la routine, drapeau de retenue CF levé (lignes 1969–1970 et 1978).

— On sauvegarde la valeur du premier chiffre dans `AH` et on appelle à nouveau la sous-routine INDIG pour voir s'il y a un second chiffre. Si ce n'est pas le cas on ne garde que le bit de poids faible dans `AL` et on termine la routine (lignes 1971–1973 et 1977–1978).

— Sinon on code les deux chiffres, contenus dans `AX`, en BCD compacté, on remplace le premier chiffre par le second, on ne garde que le bit de poids faible dans `AL` et on termine la routine (lignes 1974–1978).

6.3.9 Routine de service de la commande TIME de modification ou de lecture de l'heure

Comme nous l'avons vu, la routine de service de la commande TIME de modification ou de lecture de l'heure est repérée par l'étiquette TIME :

```
1879 : ; TIME gets and sets the time
1880 :
1881 : TIME:
1882 :          MOV     SI,81H          ;Accepting argument for time inline
1883 :          CALL    SCANOFF
1884 :          CMP     AL,13
1885 :          JZ      PRMTTIM
1886 :          MOV     BX,3A00H+":"
1887 :          CALL    INLINE
1888 :          JMP     COMTIM
1889 :
1890 : PRMTTIM:
1891 :          MOV     DX,OFFSET TRANGROUP:CURTIM
1892 :          MOV     AH,PRINTBUF
1893 :          INT     33              ;Print "Current time is "
1894 :          MOV     AH,GETTIME
1895 :          INT     33              ;Get time in CX:DX
1896 :          MOV     BL,":"
1897 :          CALL    SHOW
1898 : GETTIM:
1899 :          XOR     CX,CX           ;Initialize hours and minutes to zero
1900 :          MOV     DX,OFFSET TRANGROUP:NEWTIM
1901 :          MOV     BX,3A00H+":"
1902 :          CALL    GETBUF
1903 : COMTIM:  JZ      RET100          ;If no time present, don't change it
1904 :          JC      TIMERR
1905 :          MOV     CX,DX
1906 :          XOR     DX,DX
1907 :          LODSB
1908 :          CMP     AL,13
1909 :          JZ      SAVTIM
1910 :          CMP     AL,BL
1911 :          JNZ     TIMERR
1912 :          MOV     BL,"."
1913 :          CALL    GETNUM
1914 :          JC      TIMERR
1915 :          MOV     DH,AH           ;Position seconds
1916 :          LODSB
1917 :          CMP     AL,13
1918 :          JZ      SAVTIM
1919 :          CMP     AL,BL
1920 :          JNZ     TIMERR
1921 :          CALL    GETNUM
1922 :          JC      TIMERR
1923 :          MOV     DL,AH
1924 :          LODSB
1925 :          CMP     AL,13
1926 :          JNZ     TIMERR
1927 : SAVTIM:
1928 :          MOV     AH,SETTIME
1929 :          INT     33
1930 :          OR      AL,AL
1931 :          JZ      RET100          ;Error in time?
1932 : TIMERR:
1933 :          MOV     DX,OFFSET TRANGROUP:BADTIM
1934 :          MOV     AH,PRINTBUF
```

```
1935 :          INT     33              ;Print error message
1936 :          JMP     GETTIM          ;Try again
```

— On fait pointer `SI` sur le deuxième caractère du champ 'ligne de commande' du PSP, on appelle la sous-routine pour placer dans `AL` le premier caractère de cette ligne de commande qui n'est pas un délimiteur (lignes 1882–1883).

— S'il ne s'agit pas d'un retour à la ligne, l'utilisateur se propose de changer l'heure. On place donc '`::`' dans `BX` et on appelle la sous-routine INLINE de récupération de l'heure proposée par l'utilisateur (lignes 1884–1888).

— S'il s'agit d'un retour à la ligne, on affiche l'heure actuelle et on récupère l'heure proposée par l'utilisateur. Pour cela, on fait pointer `DX` sur le message indiquant l'heure de l'ordinateur, on appelle la fonction de l'interruption `21h` d'affichage d'une chaîne de caractères, on appelle la fonction de l'interruption `21h` de récupération de l'heure telle qu'indiquée par l'ordinateur, on place '`:`' dans `BL` et on appelle la sous-routine SHOW d'affichage de l'heure. On place 0 dans `CX` pour initialiser l'heure et la minute à zéro, on fait pointer `DX` sur le message NEWTIM, on place donc '`::`' dans `BX` et on appelle la sous-routine GETBUF de récupération de l'heure proposée par l'utilisateur (lignes 1884–1885 et 1890–1902).

Le message CURTIM est :

```
177 : CURTIM  DB      "Current time is $"
```

Le message NEWTIM est :

```
178 : NEWTIM  DB      13,10,"Enter new time: $"
```

— Dans les deux cas, si l'utilisateur n'entre pas de nouvelle heure, on termine la routine. On ne change donc pas l'heure (ligne 1903).

— Si l'heure entrée par l'utilisateur n'est pas valide, on affiche un message d'erreur et on lui demande à nouveau une nouvelle heure (lignes 1904 et 1932–1936).

Le message BADTIM est :

```
176 : BADTIM  DB      13,10,"Invalid time$"
```

— Sinon on place '.' dans `BL` et on appelle la sous-routine GETNUM pour récupérer la seconde (lignes 1912–1913).

— Si la seconde entrée par l'utilisateur n'est pas valide, on affiche un message d'erreur et on lui demande à nouveau une nouvelle heure (ligne 1914).

— Sinon on sauvegarde la seconde dans `DH`. Si le caractère suivant de la ligne de commande n'est pas un passage à la ligne, on affiche un message d'erreur et on demande à l'utilisateur à nouveau une nouvelle heure. Sinon on appelle la fonction de l'interruption `21h` d'initialisation de l'heure et, si tout s'est bien passé, on termine la routine (lignes 1923–1936).

Le numéro de la fonction de l'interruption `21h` d'initialisation de l'heure est donné par la constante SETTIME :

```
63 : SETTIME EQU     2DH
```