Chapitre 5

Implémentation des routines de service du système d'exploitation

Le noyau du système d'exploitation MS-DOS se trouve sur le fichier `MSDOS.COM`, dont le fichier source, MSDOS.ASM, est écrit en langage d'assemblage. N'a été rendu public, que très tardivement, en 2014, le fichier source de la version 1.25 de MS-DOS, correspondant à la version 1.10 de PC-DOS.

5.1 Appel des fonctions de l'interruption 21h

Les interruptions du DOS n'ont qu'une seule fonction, sauf l'interruption principale `21h`.

5.1.1 Liste des adresses des routines de service des fonctions de l'interruption 21h

Le tableau DISPATCH du fichier MSDOS.ASM de la version 1.25 liste les décalages des routines de service des 46 fonctions de l'interruption `21h` :

```
348 : ; Standard Functions
349 : DISPATCH DW      ABORT           ;0
350 :          DW      CONIN
351 :          DW      CONOUT
352 :          DW      READER
353 :          DW      PUNCH
354 :          DW      LIST            ;5
355 :          DW      RAWIO
356 :          DW      RAWINP
357 :          DW      IN
358 :          DW      PRTBUF
359 :          DW      BUFIN           ;10
360 :          DW      CONSTAT
361 :          DW      FLUSHKB
362 :          DW      DSKRESET
363 :          DW      SELDSK
364 :          DW      OPEN            ;15
365 :          DW      CLOSE
366 :          DW      SRCHFRST
367 :          DW      SRCHNXT
368 :          DW      DELETE
369 :          DW      SEQRD           ;20
370 :          DW      SEQWRT
371 :          DW      CREATE
372 :          DW      RENAME
373 :          DW      INUSE
374 :          DW      GETDRV          ;25
375 :          DW      SETDMA
376 :          DW      GETFATPT
377 :          DW      GETFATPTDL
378 :          DW      GETRDONLY
379 :          DW      SETATTRIB       ;30
380 :          DW      GETDSKPT
381 :          DW      USERCODE
382 :          DW      RNDRD
383 :          DW      RNDWRT
384 :          DW      FILESIZE        ;35
385 :          DW      SETRNDREC
386 : ; Extended Functions
387 :          DW      SETVECT
388 :          DW      NEWBASE
389 :          DW      BLKRD
390 :          DW      BLKWRT          ;40
391 :          DW      MAKEFCB
392 :          DW      GETDATE
393 :          DW      SETDATE
394 :          DW      GETTIME
395 :          DW      SETTIME         ;45
396 :          DW      VERIFY
```

Les routines de service sont, quant à elles, définies plus loin dans le code, leurs décalages étant définis grâce aux étiquettes listées ici.

Il y a donc 46 fonctions, dont les 36 premières étaient présentes sur CP/M :

```
63 : MAXCALL EQU     36
```

et peuvent être appelées de deux façons.

5.1.2 Emplacement des routines de service de ces fonctions

Les codes des routines de service des fonctions de l'interruption `21h` étant éparpillés dans le fichier MSDOS.ASM, donnons la ligne à laquelle chacune d'elles commence dans le fichier source :

— ligne 1356 pour la fonction 0 (repérée par ABORT) ;
— ligne 3130 pour la fonction 1 (CONIN) ;
— ligne 3015 pour la fonction 2 (CONOUT) ;
— ligne 432 pour la fonction 3 (READER) ;
— ligne 438 pour la fonction 4 (PUNCH) ;
— ligne 3163 pour la fonction 5 (LIST) ;
— ligne 3143 pour la fonction 6 (RAWIO) ;
— ligne 3156 pour la fonction 7 (RAWINP) ;
— ligne 3138 pour la fonction 8 (IN) ;
— ligne 3172 pour la fonction 9 (PRTBUF) ;
— ligne 2705 pour la fonction 10 (BUFIN) ;
— ligne 3122 pour la fonction 11 (CONSTAT) ;
— ligne 412 pour la fonction 12 (FLUSHKB) ;
— ligne 2656 pour la fonction 13 (DSKRESET) ;
— ligne 2698 pour la fonction 14 (SELDSK) ;
— ligne 865 pour la fonction 15 (OPEN) ;
— ligne 1013 pour la fonction 16 (CLOSE) ;
— ligne 2494 pour la fonction 17 (SRCHFRST) ;
— ligne 2554 pour la fonction 18 (SRCHNXT) ;
— ligne 704 pour la fonction 19 (DELETE) ;
— ligne 1396 pour la fonction 20 (SEQRD) ;
— ligne 1401 pour la fonction 21 (SEQWRT) ;
— ligne 1123 pour la fonction 22 (CREATE) ;
— ligne 744 pour la fonction 23 (RENAME) ;
— ligne 398 pour la fonction 24 (INUSE) ;
— ligne 2683 pour la fonction 25 (GETDRV) ;
— ligne 2611 pour la fonction 26 (SETDMA) ;
— ligne 2620 pour la fonction 27 (GETFATPT) ;
— ligne 2623 pour la fonction 28 (GETFATPTDL) ;
— ligne 401 pour la fonction 29 (GETRONLY) ;
— ligne 402 pour la fonction 30 (SETATTRIB) ;
— ligne 2644 pour la fonction 31 (GETDSKPT) ;
— ligne 403 pour la fonction 32 (USERCODE) ;
— ligne 1410 pour la fonction 33 (RNDRD) ;
— ligne 1415 pour la fonction 34 (RNDWRT) ;
— ligne 2573 pour la fonction 35 (FILESIZE) ;
— ligne 2688 pour la fonction 36 (SETRNDREC) ;
— ligne 3342 pour la fonction 37 (SETVECT) ;

— ligne 3353 pour la fonction 38 (NEWBASE) ;
— ligne 1420 pour la fonction 39 (BLKRD) ;
— ligne 1425 pour la fonction 40 (BLKWRT) ;
— ligne 3189 pour la fonction 41 (MAKEFCB) ;
— ligne 3515 pour la fonction 42 (GETDATE) ;
— ligne 3528 pour la fonction 43 (SETDATE) ;
— ligne 3592 pour la fonction 44 (GETTIME) ;
— ligne 3602 pour la fonction 45 (SETTIME) ;
— ligne 407 pour la fonction 46 (VERIFY).

5.1.3 Appel des fonctions de l'interruption 21h

Le traitement de l'appel d'une fonction de l'interruption `21h` constitue le début de la routine de service de l'interruption `21h`, repérée par l'étiquette COMMAND :

```
270 : COMMAND: ;Interrupt call entry point
271 :          CMP     AH,MAXCOM
272 :          JBE     SAVREGS
273 : BADCALL:
274 :          MOV     AL,0
275 : IRET:    IRET
276 :
277 : ENTRY:   ;System call entry point and dispatcher
278 :          POP     AX              ;IP from the long call at 5
279 :          POP     AX              ;Segment from the long call at 5
280 :          POP     CS:[TEMP]       ;IP from the CALL 5
281 :          PUSHF                   ;Start re-ordering the stack
282 :          CLI
283 :          PUSH    AX              ;Save segment
284 :          PUSH    CS:[TEMP]       ;Stack now ordered as if INT had been used
285 :          CMP     CL,MAXCALL      ;This entry point doesn't get as many calls
286 :          JA      BADCALL
287 :          MOV     AH,CL
288 : SAVREGS:
289 :          PUSH    ES
290 :          PUSH    DS
291 :          PUSH    BP
292 :          PUSH    DI
293 :          PUSH    SI
294 :          PUSH    DX
295 :          PUSH    CX
296 :          PUSH    BX
297 :          PUSH    AX
298 :
299 :          IF      DSKTEST
300 :          MOV     AX,CS:[SPSAVE]
301 :          MOV     CS:[NSP],AX
302 :          MOV     AX,CS:[SSSAVE]
303 :          MOV     CS:[NSS],AX
304 :          POP     AX
305 :          PUSH    AX
306 :          ENDIF
307 :
308 :          MOV     CS:[SPSAVE],SP
309 :          MOV     CS:[SSSAVE],SS
310 :          MOV     SP,CS
311 :          MOV     SS,SP
312 : REDISP:
313 :          MOV     SP,OFFSET DOSGROUP:IOSTACK
314 :          STI                     ;Stack OK now
315 :          MOV     BL,AH
316 :          MOV     BH,0
317 :          SHL     BX,1
318 :          CLD
319 :          CMP     AH,12
320 :          JLE     SAMSTK
321 :          MOV     SP,OFFSET DOSGROUP:DSKSTACK
322 : SAMSTK:
323 :          CALL    CS:[BX+DISPATCH]
324 : LEAVE:
325 :          CLI
326 :          MOV     SP,CS:[SPSAVE]
327 :          MOV     SS,CS:[SSSAVE]
```

```
328 :          MOV     BP,SP
329 :          MOV     BYTE PTR [BP.AXSAVE],AL
330 :
331 :          IF      DSKTEST
332 :          MOV     AX,CS:[NSP]
333 :          MOV     CS:[SPSAVE],AX
334 :          MOV     AX,CS:[NSS]
335 :          MOV     CS:[SSSAVE],AX
336 :          ENDIF
337 :
338 :          POP     AX
339 :          POP     BX
340 :          POP     CX
341 :          POP     DX
342 :          POP     SI
343 :          POP     DI
344 :          POP     BP
345 :          POP     DS
346 :          POP     ES
347 :          IRET
```

Commentaires.- 1^o) Lors de l'appel de l'interruption `21h`, le numéro de la fonction demandée doit se trouver dans le registre `AH`. Si son contenu est strictement plus grand que 36, on place le code d'erreur `00h` dans le registre `AL`, pour indiquer qu'une telle fonction ne peut pas être appelée de cette façon, et la routine est terminée (lignes 271–275).

2^o) On sauvegarde les contenus des registres sur la pile (lignes 288–297).

L'étiquette TEMP repère un mot :

```
3691 : TEMP    LABEL   WORD
```

3^o) Puisqu'on n'est pas en train de tester une disquette, on peut ignorer les lignes 299–306.

4^o) On sauvegarde le contenu des registres de pile, pointeur de pile SP et segment de pile SS, aux emplacements adéquats de la zone de données du DOS, et on confond le segment de pile avec le segment de code (lignes 308–311).

Les variables SPSAVE et SSSAVE servent à sauvegarder les valeurs du pointeur de pile et du segment de pile :

```
3692 : SPSAVE  DW      ?
3693 : SSSAVE  DW      ?
```

Le pointeur de pile est initialisé (ligne 313) à la fin de la pile utilisée par MS-DOS :

```
3718 :          DB     80H DUP (?)     ;Stack space
3719 : IOSTACK  LABEL  BYTE
3720 :          DB     80H DUP (?)
3721 : DSKSTACK LABEL  BYTE
```

On inhibe les interruptions masquables (ligne 314).

5^o) On place le numéro de fonction dans le registre `BX` (lignes 315 et 316), on le multiplie par 2, puisqu'un décalage occupe deux octets (ligne 317). Les opérations sur les chaînes de caractères décrémenteront les registres `SI` et `DI` (ligne 318).

6^o) Si la fonction est strictement supérieure à 12, c'est-à-dire concerne les disquettes, on place le pointeur de pile à la fin de la pile utilisée par MS-DOS (lignes 319–322).

7^o) On appelle la routine de service de la fonction en utilisant le tableau DISPATCH (ligne 323).

8°) On termine (lignes 324–347) classiquement la routine en se mettant à nouveau à l'écoute des interruptions masquables, en restaurant les valeurs des registres de pile, en sauvegardant la valeur de `AL` et en restaurant les valeurs des registres généraux.

5.1.4 Recours aux pré-routines définies dans IO.COM

Nous avons vu que la partie des routines de service des interruptions de MS-DOS dépendant du matétiel sont traités par des pré-routines définies dans IO.COM. Ces pré-routines sont accessibles par des sauts définis au début du code d'initialisation de IO.ASM :

```
109 :  ORG 0
110 :  PUT 100H
111 :
112 :  JMP INIT
113 :  JMP STATUS
114 :  JMP INP
115 :  JMP OUTP
116 :  JMP PRINT
117 :  JMP AUXIN
118 :  JMP AUXOUT
119 :  JMP READ
120 :  JMP WRITE
121 :  JMP DSKCHG
122 :  JMP SETDATE
123 :  JMP SETTIME
124 :  JMP GETTIME
125 :  JMP FLUSH
126 :  JMP MAPDEV
```

Ces sauts sont repérés par des étiquettes différentes dans le fichier MSDOS.ASM :

```
166 : ; BOIS entry point definitions
167 :
168 :          IF      IBM
169 : BIOSSEG EQU     60H
170 :          ENDIF
171 :          IF      NOT IBM
172 : BIOSSEG EQU     40H
173 :          ENDIF
174 :
175 : SEGBIOS          SEGMENT AT BIOSSEG
176 :                  ORG     0
177 :                  DB      3 DUP (?)       ;Reserve room for jump to init code
178 : BIOSSTAT         DB      3 DUP (?)       ;Console input status check
179 : BIOSIN           DB      3 DUP (?)       ;Get console character
180 : BIOSOUT          DB      3 DUP (?)       ;Output console character
181 : BIOSPRINT        DB      3 DUP (?)       ;Output to printer
182 : BIOSAUXIN        DB      3 DUP (?)       ;Get byte from auxilliary
183 : BIOSAUXOUT       DB      3 DUP (?)       ;Output byte to auxilliary
184 : BIOSREAD         DB      3 DUP (?)       ;Disk read
185 : BIOSWRITE        DB      3 DUP (?)       ;Disk write
186 : BIOSDSKCHG       DB      3 DUP (?)       ;Dsik-change status
187 : BIOSSETDATE      DB      3 DUP (?)       ;Set date
188 : BIOSSETTIME      DB      3 DUP (?)       ;Set time
189 : BIOSGETTIME      DB      3 DUP (?)       ;Get time and date
190 : BIOSFLUSH        DB      3 DUP (?)       ;Clear console input buffer
191 : BIOSMAPDEV       DB      3 DUP (?)       ;Dynamic disk table mapper
192 :
193 : SEGBIOS ENDS
```

(la faute d'orthographe BOIS au lieu de BIOS de la ligne 166 se trouve bien dans le source).

5.1.5 Routines des fonctions annexes (1Dh, 20h et 25h) de l'interruption 21h

Les routines de service de certaines fonctions de l'interruption `21h`, correspondant à des fonctions annexes, ont un code très simple.

5.1.5.1 Routine de service de la fonction 30 (1Dh)

La fonction 30 n'étant pas utilisée par MS-DOS, sa routine de service, repérée par l'étiquette SETATTRIB, est réduite au strict minimum :

```
402 : SETATTRIB:
403 : USERCODE:
404 :         MOV     AL,0
405 :         RET
```

Elle se contente de renvoyer 00h *via* le registre `AL` pour indiquer que tout s'est déroulé correctement.

5.1.5.2 Routine de service de la fonction 32 (20h)

La fonction 32 (`20h`) n'étant pas utilisée par MS-DOS, sa routine de service, repérée par l'étiquette USERCODE, se contente, comme nous venons de le voir, de renvoyer `00h` *via* le registre `AL` pour indiquer que tout s'est déroulé correctement.

5.1.5.3 Routine de service de la fonction 37 (25h) d'initialisation d'un vecteur d'interruption

La fonction 37 (`25h`) permet de placer l'adresse contenue dans `DS:DX` comme vecteur de l'interruption dont le numéro est contenu dans le registre `AL`.

Sa routine de service est repérée par l'étiquette SETVECT :

```
3342 : SETVECT: ; Interrupt call 37
3343 :         XOR     BX,BX
3344 :         MOV     ES,BX
3345 :         MOV     BL,AL
3346 :         SHL     BX,1
3347 :         SHL     BX,1
3348 :         MOV     ES:[BX],DX
3349 :         MOV     ES:[BX+2],DS
3350 :         RET
```

On se place, à partir du début de la mémoire (lignes 3343–3344), à quatre fois la valeur du contenu de `AL` (lignes 3345–3347), c'est-à-dire à l'emplacement du vecteur d'interruption concerné, et on y place le vecteur d'interruption (lignes 3348–3349).

5.2 Routines de service des entrées-sorties standard

5.2.1 Routine de service de la fonction 08h de saisie d'un caractère sans écho à l'écran

5.2.1.1 Routine principale

La routine de service de la fonction `08h` de l'interruption `21h` de saisie d'un caractère sans écho à l'écran est repérée, comme nous l'avons vu, par l'étiquette IN :

```
IN:     ;System call 8
        CALL    INCHK
        JZ      IN
RET29:  RET
```

Elle fait appel à la sous-routine INCHK jusqu'à ce qu'on récupère, dans le tampon des caractères, un caractère différent de l'un des trois caractères demandant une action immédiate. À ce moment ce caractère a été transféré dans le registre `AL`, on peut donc terminer la routine.

5.2.1.2 Sous-routine INCHK de filtrage de l'entrée

La sous-routine de filtrage de l'entrée est repérée par l'étiquette INCHK :

```
3031 : RET18:  RET
[...]
3039 : INCHK:
3040 :         CALL    FAR PTR BIOSIN
3041 :         CMP     AL,'P'-'@'
3042 :         JZ      PRINTON
3043 :         IF      NOT TOGLPRN
3044 :         CMP     AL,'N'-'@'
3045 :         JZ      PRINTOFF
3046 :         ENDIF
3047 :         CMP     AL,'C'-'@'
3048 :         JNZ     RET18
3049 : ; Ctrl-C handler.
3050 : ; "^C" and CR/LF is printed. Then the user registers are restored and the
3051 : ; user CTRL-C handler is executed. At this point the top of the stack has
3052 : ; 1) the interrupt return address should the user CTRL-C handler wish to
3053 : ; allow processing to continue; 2) the original interrupt return address
3054 : ; to the code that performed the function call in the first place. If the
3055 : ; user CTRL-C handler wishes to continue, it must leave all registers
3056 : ; unchanged and IRET. The function that was interrupted will simply be
3057 : ; repeated.
3058 :         MOV     AL,3            ;Display "^C"
3059 :         CALL    BUFOUT
3060 :         CALL    CRLF
3061 :         CLI                     ;Prepare to play with stack
3062 :         MOV     SS,CS:[SSSAVE]
3063 :         MOV     SP,CS:[SPSAVE]  ;User stack now restored
3064 :         POP     AX
3065 :         POP     BX
3066 :         POP     CX
3067 :         POP     DX
3068 :         POP     SI
3069 :         POP     DI
3070 :         POP     BP
3071 :         POP     DS
3072 :         POP     ES              ;User registers now restored
3073 :         INT     CONTC           ;Execute user Ctrl-C handler
3074 :         JMP     COMMAND         ;Repeat command otherwise
```

Commentaire.- 1^o) La sous-routine appelle (ligne 3040) la pré-routine de IO.COM de la fonction `01h` de l'interruption `21h` de lecture d'un caractère dans le tampon du clavier, en en attendant un si besoin est, placé alors dans le registre `AL`.

Remarquons que l'appel à une pré-routine de IO.COM s'effectue par un appel lointain.

2^o) MS-DOS traite tous les caractères de la même façon, sauf trois d'entre eux, qui exigent un traitement immédiat, à savoir CTRL-N, CTRL-P et CTRL-C.

— Le caractère 'N' a pour code ASCII 78 et '@' pour code 64, donc 'N' - '@' est le code ASCII 14, soit '*Shift out*', obtenu par la combinaison de touches CTRL-N.

Sur les téléscripteurs puis sur les claviers d'ordinateurs, une pression sur la touche de contrôle (CTRL), de façon concomitante avec une autre touche, annule les 2 bits de l'extrémité gauche des 7 bits du code du caractère ASCII (bits numéros 5 et 6 avec une numérotation commençant à 0). Ceci permet à l'opérateur de produire les 32 premiers caractères de la table ASCII, non affichables, parfois appelés *caractères de contrôle*.

La signification originelle des caractères CTRL-N et CTRL-O était, sur une machine à écrire électromécanique et sur certains téléscripteurs, par exemple le Teletype Model 38, possédant un ruban bicolore, en général rouge et noir, de passer d'une couleur à l'autre. Le noir était la couleur par défaut et donc '*Shift in*' et '*Shift out*' permettait de passer et de sortir de l'autre couleur.

Sous MS-DOS 1.0, CTRL-N signifie : supprimer l'écho de sortie sur l'imprimante.

— De même 'P' - '@', le code ASCII 16, signifiait '*Data Link Escape*' à l'origine, mais CTRL-P (avec P comme *Print*) signifie pour MS-DOS 1.0 : démarrer l'écho de sortie sur l'imprimante, action inverse de l'action précédente.

— De même, 'C' - '@', de code ASCII 3, signifiait '*End of Text* à l'origine', mais CTRL-C signifie pour MS-DOS 1.0 : terminer la commande en cours.

3^o) On traite le cas de CTRL-P :

— Si la combinaison de touches CTRL-P est pressée, on doit démarrer l'écho de sortie sur l'imprimante. Pour cela, on va (lignes 3041 et 3042) au code étiqueté par PRINTON :

```
3076 : PRINTON:
3077 :          IF      TOGLPRN
3078 :          NOT     CS:BYTE PTR [PFLAG]
3079 :          RET
3080 :          ENDIF
3081 :          IF      NOT TOGLPRN
3082 :          MOV     CS:BYTE PTR [PFLAG],1
3083 :          RET
```

La variable booléenne TOGLPRN indique qu'il faut un écho de sortie sur l'imprimante lorsqu'elle a la valeur vraie :

```
47 :          IF      IBM
[...]
51 : TOGLPRN EQU     TRUE            ;One key toggles printer echo
[...]
54 :          ELSE
[...]
58 : TOGLPRN EQU     FALSE           ;Separate keys for printer echo on and off
[...]
61 :          ENDIF
```

La variable PFLAG (le drapeau d'impression) est déclarée lors de la définition de la zone mémoire du DOS, en étant initialisé à faux.

```
3648 : PFLAG   DB      0
```

— Si TOGLPRN avait déjà la valeur vraie, on inverse la valeur de PFLAG et on termine la routine (lignes 3077–3079).

— Sinon on inverse la valeur de TOGLPRN, on met PFLAG à vrai et on termine la routine (lignes 3081–3083).

4^o) Si la valeur de vérité de TOGLPRN est faux et que la combinaison de touches CTRL-N est pressée, on doit arrêter l'écho de sortie sur l'imprimante. Pour cela, on fait appel (lignes 3043 et 3045) à la sous-routine PRINTOFF :

```
3085 : PRINTOFF:
3086 :          MOV     CS:BYTE PTR [PFLAG],0
3087 :          RET
```

qui positionne à 0 le drapeau PFLAG et termine la routine.

5^o) Sinon, lorsqu'on n'a pas appuyé sur la combinaison de touches CTRL-C, le caractère étant placé dans le registre `AL`, on termine la routine (lignes 3047 et 3048).

6^o) Il ne reste donc plus qu'un dernier cas à traiter : celui où on a appuyé sur la combinaison de touches CTRL-C, signifiant qu'on veut terminer la commande en cours.

— Dans ce cas, on commence par afficher « ^C » (lignes 3058 et 3059).

La routine d'affichage est BUFOUT :

```
2992 : BUFOUT:
2993 :          CMP     AL," "
2994 :          JAE     OUT
2995 :          CMP     AL,9
2996 :          JZ      OUT
2997 :          PUSH    AX
2998 :          MOV     AL,"^"
2999 :          CALL    OUT
3000 :          POP     AX
3001 :          OR      AL,40H
3002 :          JMP     SHORT OUT
```

— On passe à la ligne (ligne 3060).

La routine de passage à la ligne est CRLF :

```
2793 : CRLF:
2794 :          MOV     AL,13
2795 :          CALL    OUT
2796 :          MOV     AL,10
2797 :          JMP     OUT
```

— On inhibe les interruptions masquables (ligne 3061).

— On restaure les valeurs des registres de pile et des registres généraux (lignes 3062–3072).

— On appelle l'interruption `23h` (gestion de *Control-Break*).

La constante CONTC donne l'adresse du vecteur de l'interruption `23h` :

```
69 : CONTC   EQU     INTTAB+3
```

Nous étudierons la routine de service de l'interruption `23h` avec l'interpréteur de commande.

— On revient au début de la routine de service de l'interruption `21h` (ligne 3074).

5.2.2 Routine de service de la fonction 02h d'affichage d'un caractère

5.2.2.1 Partie principale de la routine

La routine de service de la fonction `02h` de l'interruption `21h` d'affichage d'un caractère est repérée, comme nous l'avons vu, par l'étiquette CONOUT :

```
3015 : CONOUT: ;System call 2
3016 :         MOV     AL,DL
3017 : OUT:
3018 :         CMP     AL,20H
3019 :         JB      CTRLOUT
3020 :         CMP     AL,7FH
3021 :         JZ      OUTCH
3022 :         INC     CS:BYTE PTR [CARPOS]
3023 : OUTCH:
3024 :         PUSH    AX
3025 :         CALL    STATCHK
3026 :         POP     AX
3027 :         CALL    FAR PTR BIOSOUT
3028 :         TEST    CS:BYTE PTR [PFLAG],-1
3029 :         JZ      RET18
3030 :         CALL    FAR PTR BIOSPRINT
3031 : RET18:  RET
```

Commentaires.- 1^o) Le code ASCII du caractère à afficher a été passé en paramètre *via* le registre `AL` lors de l'appel à l'interruption `21h`. On le sauvegarde dans le registre `DL` (ligne 3016), car on va détruire le contenu du registre `AX`.

2^o) On traite d'une façon différente des autres les trois caractère de contrôle suivants : le retour chariot, le retour en arrière et la tabulation horizontale. Plus généralement, pour tous les caractères de contrôle, c'est-à-dire de code ASCII strictement inférieur à 32, on passe (lignes 3018 et 3019) à la partie de code étiquetée par CTRLOUT, étudiée ci-dessous.

3^o) On incrémente la position du caractère en mémoire graphique, sauf s'il s'agit du caractère DEL, de code ASCII `7Fh` (lignes 3020–3022).

La position du caractère est sauvegardée dans la variable CARPOS, se trouvant dans la zone de communication du DOS :

```
3646 : CARPOS  DB      0
```

4^o) On sauvegarde sur la pile la valeur du registre `AX`, car elle va être détruite par la sous-routine appelée tout de suite après, on appelle la sous-routine STATCHK et on restaure la valeur du registre `AX` (lignes 3024–3026).

La routine STATCHK vérifie la suspension de l'affichage :

```
3031 : RET18:  RET
3032 :
3033 : STATCHK:
3034 :         CALL    FAR PTR BIOSSTAT
3035 :         JZ      RET18
3036 :         CMP     AL,'S'-'@'
3037 :         JNZ     NOSTOP
3038 :         CALL    FAR PTR BIOSIN          ;Eat Cntrl-S
3039 : INCHK:
```

— On appelle la pré-routine de IO.COM de la fonction `0Bh` de l'interruption `21h` permettant de vérifier si un caractère est présent dans le tampon du clavier (ligne 3034), interruption non bloquante.

Rappelons que s'il y en a un, l'indicateur de zéro est mis à zéro et le code ASCII du caractère est placé dans `AL` (sans être retiré du tampon) et que, sinon, l'indicateur de zéro est levé.

— Si l'indicateur de zéro vaut un, on termine la sous-routine (ligne 3035).

— S'il y a un caractère dans le tampon du clavier et qu'il s'agit de CTRL-S (ligne 3036), on le récupère du tampon du clavier (puisqu'on va en traiter l'action immédiate).

Sous MS-DOS, CTRL-S (avec S pour *Suspend*) suspend l'affichage des résultats à l'écran, par exemple pour avoir le temps de les lire lors d'un défilement rapide. On appuie sur n'importe quelle touche pour reprendre le défilement.

— Sinon on va à NOSTOP :

```
3004 : NOSTOP:
3005 :          CMP     AL,"P"-"@"
3006 :          JZ      INCHK
3007 :          IF      NOT TOGLPRN
3008 :          CMP     AL,"N"-"@"
3009 :          JZ      INCHK
3010 :          ENDIF
3011 :          CMP     AL,"C"-"@"
3012 :          JZ      INCHK
3013 :          RET
```

dont nous avons déjà commenté la partie INCHK à propos de la lecture d'un caractère.

5^o) Après avoir fait appel à la sous-routine STATCHK, on appelle (ligne 3027) la pré-routine de IO.COM de la fonction `02h` de l'interruption `21h` d'affichage d'un caractère à l'écran.

6^o) Nous avons vu qu'il peut y avoir, sous MS-DOS, écho sur l'imprimante. Si celui-ci est activé, on imprime également le caractère (lignes 3028–3030). Dans tous les cas on termine la routine.

5.2.2.2 Cas des trois caractères de contrôle spécifiques

Nous avons vu que nous sommes renvoyés à la partie de code repérée par l'étiquette CTRLOUT pour traiter trois des caractères de contrôle :

```
3090 : CTRLOUT:
3091 :          CMP     AL,13
3092 :          JZ      ZERPOS
3093 :          CMP     AL,8
3094 :          JZ      BACKPOS
3095 :          CMP     AL,9
3096 :          JNZ     OUTCHJ
3097 :          MOV     AL,CS:[CARPOS]
3098 :          OR      AL,0F8H
3099 :          NEG     AL
3100 : TAB:
3101 :          PUSH    CX
3102 :          MOV     CL,AL
3103 :          MOV     CH,0
3104 :          JCXZ    POPTAB
3105 : TABLP:
3106 :          MOV     AL," "
3107 :          CALL    OUT
3108 :          LOOP    TABLP
3109 : POPTAB:
3110 :          POP     CX
3111 : RET19:   RET
3112 :
3113 : ZERPOS:
3114 :          MOV     CS:BYTE PTR [CARPOS],0
3115 : OUTCHJ:  JMP     OUTCH
3116 :
3117 : BACKPOS:
3118 :          DEC     CS:BYTE PTR [CARPOS]
3119 :          JMP     OUTCH
```

- 1^o) S'il s'agit du caractère de code ASCII 13, c'est-à-dire un retour chariot, on revient en début de ligne (lignes 3091–3092 et 3113–3114).

- 2^o) S'il s'agit du caractère de code ASCII 8, c'est-à-dire un retour en arrière, on décrémente la position (lignes 3093–3094 et 3117–3119).

- 3^o) S'il s'agit du caractère de code ASCII 9, c'est-à-dire d'une tabulation horizontale, (lignes 3095–3111), on se place à la tabulation horizontale suivante.

- 4^o) Les autres cas sont traités de la façon habituelle (lignes 3095–3096).

5.2.3 Routines de service des fonctions auxiliaires 01h, 03h, 04h, 05h, 06h, 07h, 09h, 0Ah, 0Bh et Ch

5.2.3.1 Routine de service de la fonction 01h de saisie d'un caractère avec écho à l'écran

La routine de service de la fonction `01h` de l'interruption `21h`, de saisie d'un caractère avec écho à l'écran, est repérée par l'étiquette CONIN :

```
3130 : CONIN:  ;System call 1
3131 :         CALL    IN
3132 :         PUSH    AX
3133 :         CALL    OUT
3134 :         POP     AX
3135 :         RET
```

Elle récupère un caractère et l'affiche.

Puisque le contenu du registre `AX` est détruit par la routine OUT, celui-ci est sauvegardé avant l'appel à cette routine, puis restauré ensuite.

5.2.3.2 Routine de service de la fonction 03h de lecture d'un caractère sur l'interface auxiliaire

La routine de service de la fonction `03h` de l'interruption `21h`, de lecture d'un caractère sur l'interface auxiliaire, est repérée par l'étiquette READER :

```
432 : READER:
433 : AUXIN:
434 :          CALL    STATCHK
435 :          CALL    FAR PTR BIOSAUXIN
436 :          RET
```

On appelle la routine STATCHK de vérification de suspension de l'affichage, déjà étudiée à propos de l'affichage d'un caractère, puis on appelle la pré-routine de la fonction `03h` de IO.COM.

5.2.3.3 Routine de service de la fonction 04h d'envoi d'un caractère sur l'interface auxiliaire

La routine de service de la fonction `04h` de l'interruption `21h`, d'envoi d'un caractère sur l'interface auxiliaire, est repérée par l'étiquette PUNCH :

```
438 : PUNCH:
439 :          MOV     AL,DL
440 : AUXOUT:
441 :          PUSH    AX
442 :          CALL    STATCHK
443 :          POP     AX
444 :          CALL    FAR PTR BIOSAUXOUT
445 :          RET
```

On place le caractère à transférer, se trouvant dans le registre `DL` lors de l'appel de l'interruption, est placé dans le registre `AL`, on sauvegarde sur la pile le contenu du registre `AX`, on appelle la routine STATCHK de vérification de suspension de l'affichage, on restaure le contenu de `AX` et on appelle la pré-routine de IO.COM de la fonction `04h`.

5.2.3.4 Routine de service de la fonction 05h d'impression d'un caractère

La routine de service de la fonction `05h` de l'interruption `21h`, d'impression d'un caractère, est repérée par l'étiquette LIST :

```
3163 : LIST:   ;System call 5
3164 :         MOV     AL,DL
3165 : LISTOUT:
3166 :         PUSH    AX
3167 :         CALL    STATCHK
3168 :         POP     AX
3169 :         CALL    FAR PTR BIOSPRINT
3170 : RET20:  RET
```

On place le code ASCII du caractère à imprimer, se trouvant dans le registre `DL` lors de l'appel de l'interruption, dans le registre `AL` (ligne 3164). On le sauvegarde sur la pile, on appelle la routine de vérification de suspension de l'affichage puis on restaure la valeur de `AX` (lignes 3166–3168). On appelle la pré-routine de IO.COM de la fonction `05h` (ligne 3169).

5.2.3.5 Routine de service de la fonction 06h de sortie directe sur la console

La routine de service de la fonction `06h` de l'interruption `21h`, de sortie directe sur la console, est repérée par l'étiquette RAWIO :

```
3143 : RAWIO:  ;System call 6
3144 :         MOV     AL,DL
3145 :         CMP     AL,-1
3146 :         JNZ     RAWOUT
3147 :         LDS     SI,DWORD PTR CS:[SPSAVE]        ;Get pointer to register save area
3148 :         CALL    FAR PTR BIOSSTAT
3149 :         JNZ     RESFLG
3150 :         OR      BYTE PTR [SI.FSAVE],40H ;Set user's zero flag
3151 :         XOR     AL,AL
3152 :         RET
3153 :
3154 : RESFLG:
3155 :         AND     BYTE PTR [SI.FSAVE],0FFH-40H    ;Reset user's zero flag
3156 : RAWINP: ;System call 7
3157 :         CALL    FAR PTR BIOSIN
3158 :         RET
3159 : RAWOUT:
3160 :         CALL    FAR PTR BIOSOUT
3161 :         RET
```

— On place l'octet de commande, contenu dand le registre `DL` lors de l'appel de l'interruption, dans le registre `AL` (ligne 3144).

— Si l'octet de commande n'est pas `FFh` (lignes 3145 et 3146), c'est qu'il s'agit d'une saisie. On fait donc pointer le registre `SI` sur la pile du DOS (ligne 3147) et on appelle la pré-routine de IO.COM testant si le tampon du clavier est non vide (ligne 3148).

S'il est vide (lignes 3148 et 3149), on lève le drapeau de zéro (ligne 3150), on place zéro dans le registre `AL` (ligne 3151) et on termine la routine (ligne 3152).

S'il n'est pas vide (lignes 3148 et 3149), on baisse le drapeau de zéro (ligne 3155), on appelle la pré-routine de IO.COM de saisie d'un caractère (ligne 3157) et on termine la routine (ligne 3158).

— Si l'octet de commande est `FFh`, on doit afficher le caractère. On va donc à l'étiquette RAWOUT (lignes 3145 et 3146), où l'on se contente de faire appel à la pré-routine de IO.COM d'affichage d'un caractère.

5.2.3.6 Routine de service de la fonction 07h d'entrée directe sur la console sans écho

La routine de service de la fonction `07h` de l'interruption `21h`, d'entrée directe sur la console sans écho, est repérée par l'étiquette RAWINP :

```
3156 : RAWINP: ;System call 7
3157 :         CALL    FAR PTR BIOSIN
3158 :         RET
```

Elle se contente d'appeler la pré-routine de IO.COM de saisie d'un caractère.

5.2.3.7 Routine de service de la fonction 09h d'affichage d'une chaîne de caractères

La routine de service de la fonction `09h` de l'interruption `21h`, d'affichage de la chaîne de caractères d'adresse `DS:DX` et se terminant par le caractère '$' (de code ASCII `24h`), est repérée par l'étiquette PRTBUF :

```
3172 : PRTBUF: ;System call 9
3173 :         MOV     SI,DX
3174 : OUTSTR:
3175 :         LODSB
3176 :         CMP     AL,"$"
3177 :         JZ      RET20
3178 :         CALL    OUT
3179 :         JMP     SHORT OUTSTR
```

5.2.3.8 Routine de service de la fonction 0Ah de saisie de caractères à placer dans un tampon

La fonction `0Ah` de l'interruption `21h` permet de saisir des caractères au clavier et de les placer dans un tampon. Le paramètre `DS:DX` pointe sur un tampon d'entrée. Le premier octet de celui-ci, non nul, spécifie le nombre de caractères que peut contenir le tampon. On lit les caractères à partir du clavier et on les place dans le tampon en commençant par les placer au troisième octet du tampon. La lecture au clavier et le remplissage du tampon sont effectués jusqu'à ce qu'on appuie sur la touche 'Enter'. Lorsqu'on en arrive à l'avant dernier caractère que le tampon peut contenir, chaque caractère additionnel est ignoré et cause l'émission d'un bip, jusqu'à ce qu'on appuie sur la touche 'Enter'. Le second octet du tampon contient le nombre de caractères reçus, sans compter le caractère retour chariot (de code ASCII `0Dh`), qui est toujours le dernier caractère.

La routine de service de la fonction `0Ah` de l'interruption `21h`, de saisir de caractères au clavier à placer dans un tampon, est repérée par l'étiquette BUFIN :

```
2703 : RET17:  RET
2704 :
2705 : BUFIN:  ;System call 10
2706 :         MOV     AX,CS
2707 :         MOV     ES,AX
2708 :         MOV     SI,DX
2709 :         MOV     CH,0
2710 :         LODSW
2711 :         OR      AL,AL
2712 :         JZ      RET17
2713 :         MOV     BL,AH
2714 :         MOV     BH,CH
2715 :         CMP     AL,BL
```

```
2716 :          JBE     NOEDIT
2717 :          CMP     BYTE PTR [BX+SI],0DH
2718 :          JZ      EDITON
2719 : NOEDIT:
2720 :          MOV     BL,CH
2721 : EDITON:
2722 :          MOV     DL,AL
2723 :          DEC     DX
2724 : NEWLIN:
2725 :          MOV     AL,CS:[CARPOS]
2726 :          MOV     CS:[STARTPOS],AL
2727 :          PUSH    SI
2728 :          MOV     DI,OFFSET DOSGROUP:INBUF
2729 :          MOV     AH,CH
2730 :          MOV     BH,CH
2731 :          MOV     DH,CH
2732 : GETCH:
```

— On confond le segment supplémentaire avec le segment de code (lignes 2706 et 2707) et on fait pointer `SI` sur le décalage du tampon, spécifié par l'adresse `DS:DX` lors de l'appel de l'interruption (ligne 2708).

— On charge le contenu des deux premiers octets du tampon dans le registre `AX` (ligne 2710). Si le premier octet de celui-ci, spécifiant le nombre de caractères que peut contenir ce tampon, est nul, on termine la routine (lignes 2711 et 2712).

— On place dans le registre `BL` le nombre de caractères maximum du tampon (ligne 2713) et 0 dans le registre `BH`, nombre de caractères déjà présents dans le tampon (lignes 2709 et 2714).

— Si le nombre de caractères maximum est atteint et qu'on ne pointe pas sur un retour chariot (lignes 2715–2718), le nombre de caractères maximum est mis à zéro (ligne 2720).

— On place le contenu de `AL` dans `DX` et on décrémente (ligne 2723).

5.2.3.9 Routine de service de la fonction 0Bh de vérification du statut du clavier

La routine de service de la fonction `0Bh` de l'interruption `21h`, de vérification du statut du clavier, est repérée par l'étiquette CONSTAT :

```
3111 : RET19:  RET
[...]
3122 : CONSTAT: ;System call 11
3123 :         CALL    STATCHK
3124 :         MOV     AL,0
3125 :         JZ      RET19
3126 :         OR      AL,-1
3127 :         RET
```

— On appelle la pré-routine de IO.COM de cette même fonction (ligne 3123).
Cette pré-routine baisse, entre autres, le drapeau de zéro ZF si un caractère se trouve dans le tampon du clavier et le lève sinon.
— On place 0 dans le registre `AL` (ligne 3124).
— Si le drapeau ZF est levé, il n'y a pas de caractère dans le tampon. On termine la routine, avec le drapeau ZF levé (ligne 3125).
— Sinon on place `FFh` dans `AL` et on termine la routine (lignes 3126–3127).

5.2.3.10 Routine de service de la fonction 12 (0Ch) de vidage du tampon du clavier

La routine de service de la fonction `Ch` de l'interruption `21h`, de vidage du tampon du clavier puis de l'exécution de la fonction dont le numéro est dans `AL` (seuls `01h`, `06h`, `07h`, `08h` et `Ah` étant permis), est repérée par l'étiquette FLUSHKB :

```
412 : FLUSHKB:
413 :         PUSH    AX
414 :         CALL    FAR PTR BIOSFLUSH
415 :         POP     AX
416 :         MOV     AH,AL
417 :         CMP     AL,1
418 :         JZ      REDISPJ
419 :         CMP     AL,6
420 :         JZ      REDISPJ
421 :         CMP     AL,7
422 :         JZ      REDISPJ
423 :         CMP     AL,8
424 :         JZ      REDISPJ
425 :         CMP     AL,10
426 :         JZ      REDISPJ
427 :         MOV     AL,0
428 :         RET
429 :
430 : REDISPJ:JMP     REDISP
```

— Le contenu du registre `AX` est sauvegardé, donc essentiellement le numéro de la fonction qu'il va falloir appeler, on appelle la pré-routine de IO.COM de vidage du tampon du clavier, dont on a vu qu'en fait elle ne fait rien, et on restaure la valeur de `AX` (lignes 413–415).
— La valeur du registre `AL` est transférée dans le registre `AH`, puisque c'est dans celui-ci qu'on doit placer le numéro de fonction avant l'appel de l'interruption `21h` (ligne 416).
— Si cette valeur est `01h`, `06h`, `07h`, `08h` ou `0Ah`, on exécute la fonctions de l'interruption `21h` correspondante. Sinon on termine la routine en plaçant 0 dans `AL` (lignes 416–430).
L'étiquette REDISP repère une partie de la routine de service de l'interruption `21h`.

5.3 Routines de service de gestion de la date et de l'heure

5.3.1 Routine de service de la fonction 42 (2Ah) de récupération de la date

5.3.1.1 Routine principale

La routine de service de la fonction 42 de l'interruption `21h`, de récupération de la date dans `CX`, `DX` et `AL`, avec `CX` contenant l'année (1980–2099 en binaire), `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.), est repérée par l'étiquette GETDATE :

```
3515:GETDATE: ;Function call 42
3516:         PUSH    CS
3517:         POP     DS
3518:         CALL    READTIME                ;Check for rollover to next day
3519:         MOV     AX,[YEAR]
3520:         MOV     BX,WORD PTR [DAY]
3521:         LDS     SI,DWORD PTR [SPSAVE]   ;Get pointer to user registers
3522:         MOV     [SI.DXSAVE],BX          ;DH=month, DL=day
3523:         ADD     AX,1980                 ;Put bias back
3524:         MOV     [SI.CXSAVE],AX          ;CX=year
3525:         MOV     AL,CS:[WEEKDAY]
3526:RET24:   RET
```

— On confond le segment des données avec le segment de code et on appelle la routine READTIME, étudiée ci-dessous, pour placer dans la variable DAYCNT le nombre de jours écoulés depuis le 1er janvier 1980, dans la variable YEAR l'année, dans la variable MONTH le mois, dans la variable DAY le quantième (jour dans le mois) et dans la variable WEEKDAY le jour dans la semaine.

Les variables DAY, MONTH, YEAR, DAYCNT et WEEKDAY contiennent respectivement le quantième (jour dans le mois), le mois, l'année, le nombre de jours écoulés depuis le 1er janvier 1980 et le jour dans la semaine :

```
3664:DAY     DB      0
3665:MONTH   DB      0
3666:YEAR    DW      0
3667:DAYCNT  DW      -1
3668:WEEKDAY DB      0
```

— On place l'année dans `AX`, le jour du mois et le mois dans `BX`, on fait pointer `SI` sur le contenu de la variable SPSAVE, on place le mois sur la sauvegarde de `DH`, le jour du mois sur la sauvegarde de `DL`, on ajoute 1980 au contenu `AX` pour obtenir l'année, que l'on place dans la sauvegarde de `CX` puis le jour de la semaine dans la sauvegarde de `AL` et on termine la routine (lignes 3519–3526).

5.3.1.2 Lecture de la date

La routine READTIME commence par récupérer le nombre de jours écoulés depuis le 1er janvier 1980 dans le registre `AX`, en utilisant la pré-routine de IO.COM. Si la date a changé, elle est ajustée, en prenant en compte le nombre de jours de chaque mois et des années bissextiles pour placer dans la variable DAYCNT le nombre de jours écoulé depuis le 1er janvier 1980, dans YEAR l'année, dans MONTH le mois, dans DAY le quantième et dans WEEKDAY le jour de la semaine :

```
3431 : RET22:  RET
[...]
3435 : READTIME:
3436 : ;Gets time in CX:DX. Figures new date if it has changed.
3437 : ;Uses AX, CX, DX.
3438 :         CALL    FAR PTR BIOSGETTIME
3439 :         CMP     AX,[DAYCNT]                     ;See if day count is the same
3440 :         JZ      RET22
3441 :         CMP     AX,FOURYEARS*30                 ;Number of days in 120 years
3442 :         JAE     RET22                           ;Ignore if too large
3443 :         MOV     [DAYCNT],AX
3444 :         PUSH    SI
3445 :         PUSH    CX
3446 :         PUSH    DX                              ;Save time
3447 :         XOR     DX,DX
3448 :         MOV     CX,FOURYEARS                    ;Number of days in 4 years
3449 :         DIV     CX                              ;Compute number of 4-year units
3450 :         SHL     AX,1
3451 :         SHL     AX,1
3452 :         SHL     AX,1                            ;Multiply by 8 (no. of half-years)
3453 :         MOV     CX,AX                           ;<240 implies AH=0
3454 :         MOV     SI,OFFSET DOSGROUP:YRTAB        ;Table of days in each year
3455 :         CALL    DSLIDE                          ;Find out which of four years we're in
3456 :         SHR     CX,1                            ;Convert half-years to whole years
3457 :         JNC     SK                              ;Extra half-year?
3458 :         ADD     DX,200
3459 : SK:
3460 :         CALL    SETYEAR
3461 :         MOV     CL,1                            ;At least at first month in year
3462 :         MOV     SI,OFFSET DOSGROUP:MONTAB       ;Table of days in each month
3463 :         CALL    DSLIDE          ;Find out which month we're in
3464 :         MOV     [MONTH],CL
3465 :         INC     DX              ;Remainder is day of month (start with one)
3466 :         MOV     [DAY],DL
3467 :         CALL    WKDAY           ;Set day of week
3468 :         POP     DX
3469 :         POP     CX
3470 :         POP     SI
3471 : RET23:  RET
```

— On appelle la pré-routine BIOSGETTIME, définie dans IO.ASM sous le nom de GETTIME, pour placer le nombre de jours écoulés depuis le 1er janvier 1980 dans le registre `AX` (ligne 3448).

Elle place également les secondes et les centièmes de seconde dans `DX`, l'heure et la minute dans `CX`, mais cela ne nous intéresse pas pour l'instant.

— Si ce nombre est le même que celui actuellement contenu dans la variable DAYCNT, il n'y a rien à faire. On termine donc la routine (lignes 3439–3440).

— Si l'année est postérieure à 2099, on termine la routine. Sinon on place le nouveau nombre de jours écoulé depuis le 1er janvier 1980 dans la variable DAYCNT (lignes 3441–3443).

— On sauvegarde sur la pile les contenus de `SI`, `CX` et `DX`, c'est-à-dire ce qui concerne l'heure qui ne nous intéresse pas ici mais dont nous aurons besoin pour la routine de service de la fonction 44.

— On place 0 dans `DX` car on va effectué une division sur `DX:AX`, on divise le nouveau nombre de jours écoulé depuis le 1er janvier par le nombre de jours pour une durée de quatre ans, comprenant donc une année bissextile, ce qui change les contenus de `AX` et de `DX`, que l'on multiplie par huit, pour obtenir le nombre de demi-années écoulées, que l'on place dans `CX` (lignes 3444–3453).

La constante FOURYEARS contient le nombre de jours sur une période de quatre ans, comprenant donc une année bissextile :

```
3433 : FOURYEARS       EQU     3*365+366
```

— On fait pointer `SI` sur la table YRTAB des nombres de jours par an contenus sur une période de quatre ans, commençant par une année bissextile, et on appelle la routine DSLIDE, étudiée ci-dessous, pour déterminer l'année dans laquelle on se trouve (lignes 3454–3455).

La table YRTAB contient le nombre de jours par an contenus sur une période de quatre ans, en commençant par une année bissextile :

```
3495 : ;Days in year
3496 : YRTAB   DB      200,166         ;Leap year
3497 :         DB      200,165
3498 :         DB      200,165
3499 :         DB      200,165
```

— On revient au nombre d'années entières (lignes 3456–3459).

— On appelle la routine SETYEAR, étudiée ci-dessous, pour placer le numéro de l'année dans la variable YEAR et pour ajuster le nombre de jours du mois de février dans le cas d'une année bissextile (ligne 3460).

— On détermine le mois dans lequel on se trouve et on le place dans la variable MONTH (lignes 3461–3464).

La table MONTAB contient le nombre de jours de chaque mois :

```
3501 : ;Days of each month
3502 : MONTAB  DB      31              ;January
3503 :         DB      28              ;February--reset each time year changes
3504 :         DB      31              ;March
3505 :         DB      30              ;April
3506 :         DB      31              ;May
3507 :         DB      30              ;June
3508 :         DB      31              ;July
3509 :         DB      31              ;August
3510 :         DB      30              ;September
3511 :         DB      31              ;October
3512 :         DB      30              ;November
3513 :         DB      31              ;December
```

— On détermine le quantième (jour dans le mois) et on le place dans la variable DAY (lignes 3465–3466).

— On appelle la routine WKDAY, étudiée ci-dessous, pour déterminer le jour de la semaine et le placer dans la variable WEEKDAY, on restaure les contenus `DX`, `CX` et `SI`, dont nous aurons besoin pour la routine de service de la fonction 44, et on termine la routine (lignes 3467–3471).

5.3.1.3 Routine de détermination de l'année

La routine de détermination de l'année est DSLIDE :

```
3473:DSLIDE:
3474:        MOV     AH,0
3475:DSLIDE1:
3476:        LODSB                   ;Get count of days
3477:        CMP     DX,AX           ;See if it will fit
3478:        JB      RET23           ;If not, done
3479:        SUB     DX,AX
3480:        INC     CX              ;Count one more month/year
3481:        JMP     SHORT DSLIDE1
```

On place 0 dans `AH`, on lit la valeur pour la première année et on termine si le nombre de jours correspond à la première année. Sinon on enlève le nombre de jours de la première année de `AX` et on incrémente `CX` pour ajouter une année et ainsi de suite jusqu'au plus la quatrième année (lignes 3473–3481).

5.3.1.4 Routine d'ajustement du nombre de jours en février

La routine de détermination de l'année et d'ajustement du nombre de jours en février est SETYEAR :

```
3483 : SETYEAR:
3484 : ;Set year with value in CX. Adjust length of February for this year.
3485 :         MOV     BYTE PTR [YEAR],CL
3486 : CHKYR:
3487 :         TEST    CL,3            ;Check for leap year
3488 :         MOV     AL,28
3489 :         JNZ     SAVFEB          ;28 days if no leap year
3490 :         INC     AL              ;Add leap day
3491 : SAVFEB:
3492 :         MOV     [MONTAB+1],AL   ;Store for February
3493 :         RET
```

— On place le numéro d'année obtenue dans la variable YEAR (ligne 3485).
— S'il s'agit d'une année bissextile, on ajuste le nombre de jours du mois de février pour cette année (lignes 3487–3493).

5.3.1.5 Routine de détermination du jour de la semaine

Étant donné un jour, repéré dans la variable DAYCNT par le nombre de jours écoulés depuis le 1er janvier 1980, la routine WKDAY détermine le jour de la semaine (dans l'amplitude 0 à 6 avec 0 pour dimanche) et de le place dans la variable WEEKDAY :

```
3572:WKDAY:
3573:         MOV     AX,[DAYCNT]
3574:         XOR     DX,DX
3575:         MOV     CX,7
3576:         INC     AX
3577:         INC     AX              ;First day was Tuesday
3578:         DIV     CX              ;Compute day of week
3579:         MOV     [WEEKDAY],DL
3580:         XOR     AL,AL           ;Flag OK
3581:RET25:   RET
```

On place ce nombre de jours écoulés dans `AX`, que l'on incrémente deux fois puisque le 1er janvier 1980 était un mardi. Le jour de la semaine est donné par le reste dans la division euclidienne de ce nombre par 7, que l'on place dans la variable WEEKDAY. On renvoie 0 dans `AL`.

5.3.2 Routine de service de la fonction 43 (2Bh) d'initialisation de la date

5.3.2.1 Routine principale

La routine de service de la fonction 43 de l'interruption `21h`, d'initialisation de la date, étant donnée la date dans `CX:DX`, avec `CX` contenant l'année (1980–2099 en binaire), `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.), est repérée par l'étiquette SETDATE :

```
3528 : SETDATE: ;Function call 43
3529 :          MOV     AL,-1                   ;Be ready to flag error
3530 :          SUB     CX,1980                 ;Fix bias in year
3531 :          JC      RET24                   ;Error if not big enough
3532 :          CMP     CX,119                  ;Year must be less than 2100
3533 :          JA      RET24
3534 :          OR      DH,DH
3535 :          JZ      RET24
3536 :          OR      DL,DL
3537 :          JZ      RET24                   ;Error if either month or day is 0
3538 :          CMP     DH,12                   ;Check against max. month
3539 :          JA      RET24
3540 :          PUSH    CS
3541 :          POP     DS
3542 :          CALL    CHKYR                   ;Set Feb. up for new year
3543 :          MOV     AL,DH
3544 :          MOV     BX,OFFSET DOSGROUP:MONTAB-1
3545 :          XLAT                            ;Look up days in month
3546 :          CMP     AL,DL
3547 :          MOV     AL,-1                   ;Restore error flag, just in case
3548 :          JB      RET24                   ;Error if too many days
3549 :          CALL    SETYEAR
3550 :          MOV     WORD PTR [DAY],DX       ;Set both day and month
3551 :          SHR     CX,1
3552 :          SHR     CX,1
3553 :          MOV     AX,FOURYEARS
3554 :          MOV     BX,DX
3555 :          MUL     CX
3556 :          MOV     CL,BYTE PTR [YEAR]
3557 :          AND     CL,3
3558 :          MOV     SI,OFFSET DOSGROUP:YRTAB
3559 :          MOV     DX,AX
3560 :          SHL     CX,1                    ;Two entries per year, so double count
3561 :          CALL    DSUM                    ;Add up the days in each year
3562 :          MOV     CL,BH                   ;Month of year
3563 :          MOV     SI,OFFSET DOSGROUP:MONTAB
3564 :          DEC     CX                      ;Account for months starting with one
3565 :          CALL    DSUM                    ;Add up days in each month
3566 :          MOV     CL,BL                   ;Day of month
3567 :          DEC     CX                      ;Account for days starting with one
3568 :          ADD     DX,CX                   ;Add in to day total
3569 :          XCHG    AX,DX                   ;Get day count in AX
3570 :          MOV     [DAYCNT],AX
3571 :          CALL    FAR PTR BIOSSETDATE
3572 : WKDAY:
```

— On place le code d'erreur `FFh`, renvoyé dans le cas où la date n'est pas valide, dans `AL`. On soustrait 1980 à `CX` pour obtenir la différence de l'année depuis 1980. Si on obtient un nombre négatif ou si cette différence est supérieure à 119, la date n'est pas dans l'amplitude voulue ; on termine donc la routine avec le code d'erreur `FFh` (lignes 3529–3533).

— Si le numéro du mois est nul ou strictement supérieur à 12, on n'est pas dans l'amplitude voulue pour le mois ; on termine donc la routine avec le code d'erreur `FFh` (lignes 3534–3535 et 3538–3539).

— Si le numéro du jour est nul, on termine la routine avec le code d'erreur `FFh` (lignes 3536–3537).

Pour l'instant on ne peut rien dire sur la limite supérieure du numéro du jour, puisque celle-ci dépend du mois.

— On confond le segment des données et le segment de code et on appelle la routine CHKYR, partie de SETYEAR, pour ajuster le nombre de jours du mois de février suivant qu'il s'agit d'une année bissextile ou non (lignes 3540–3542).

— Si le jour est strictement supérieur au nombre de jours dans le mois, on termine la routine avec le code d'erreur `FFh` (lignes 3543–3548).

— On appelle la routine SETYEAR pour placer l'année dans la variable YEAR (ligne 3549).

— On place le jour du mois et le mois dans les variables DAY et MONTH (ligne 3550).

— On détermine le nombre de jours écoulés depuis le 1er janvier 1980 et on le place dans la variable DAYCNT (lignes 3551–3570).

— On appelle la pré-routine BIOSSETDATE de IO.COM d'initialisation de la date pour mettre à jour la date dans les registres du minuteur (ligne 3571).

— On détermine le numéro du jour dans la semaine et on le place dans la variable WEEKDAY (lignes 3572–3579).

— On place le code d'erreur `00h` dans `AL` et on termine la routine (lignes 3580–3581).

5.3.2.2 Routine de sommation du nombre de jours

La routine de sommation du nombre de jours est DSUM :

```
3581 : RET25:  RET
3582 :
3583 : DSUM:
3584 :         MOV     AH,0
3585 :         JCXZ    RET25
3586 : DSUM1:
3587 :         LODSB
3588 :         ADD     DX,AX
3589 :         LOOP    DSUM1
3590 :         RET
```

5.3.3 Routine de la fonction 44 (2Ch) de récupération de l'heure

La routine de service de la fonction 44 de l'interruption `21h`, renvoyant l'heure dans `CX:DX`, avec `CH` contenant l'heure (0–23), `CL` les minutes (0–59), `DH` les secondes (0–59), `DL` les 1/100 ièmes de secondes (0–99), est repérée par l'étiquette GETTIME :

```
3592 : GETTIME: ;Function call 44
3593 :          PUSH    CS
3594 :          POP     DS
3595 :          CALL    READTIME
3596 :          LDS     SI,DWORD PTR [SPSAVE]   ;Get pointer to user registers
3597 :          MOV     [SI.DXSAVE],DX
3598 :          MOV     [SI.CXSAVE],CX
3599 :          XOR     AL,AL
3600 : RET26:   RET
```

— On confond le segment des données et celui du code (ligne 3593).

— On appelle la routine READTIME pour placer les secondes et les centièmes de seconde dans `DX`, l'heure et la minute dans `CX`.

— On donne à `SI` la valeur sauvegardée du pointeur de pile, on sauvegarde les contenus de `DX` et de `CX`, on place 0 dans `AL` et on termine la routine (lignes 3596–3600).

La structure STKPTRS contient l'emplacement des registres généraux :

```
194 : ; Location of user registers relative user stack pointer
195 :
196 : STKPTRS STRUC
197 : AXSAVE  DW      ?
198 : BXSAVE  DW      ?
199 : CXSAVE  DW      ?
200 : DXSAVE  DW      ?
201 : SISAVE  DW      ?
202 : DISAVE  DW      ?
203 : BPSAVE  DW      ?
204 : DSSAVE  DW      ?
205 : ESSAVE  DW      ?
206 : IPSAVE  DW      ?
207 : CSSAVE  DW      ?
208 : FSAVE   DW      ?
209 : STKPTRS ENDS
```

5.3.4 Routine de service de la fonction 45 (2Dh) d'initialisation de l'heure

La routine de service de la fonction 45 de l'interruption `21h`, d'initialisation de l'heure (`CH` contenant l'heure (0–23), `CL` la minute (0–59), `DH` la seconde (0–59) et `DL` la 1/100 ièmee de seconde (0–99)), est repérée par l'étiquette SETTIME :

```
3602 : SETTIME: ;Function call 45
3603 : ;Time is in CX:DX in hours, minutes, seconds, 1/100 sec.
3604 :          MOV     AL,-1           ;Flag in case of error
3605 :          CMP     CH,24           ;Check hours
3606 :          JAE     RET26
3607 :          CMP     CL,60           ;Check minutes
3608 :          JAE     RET26
3609 :          CMP     DH,60           ;Check seconds
3610 :          JAE     RET26
3611 :          CMP     DL,100          ;Check 1/100's
3612 :          JAE     RET26
3613 :          CALL    FAR PTR BIOSSETTIME
3614 :          XOR     AL,AL
3615 :          RET
```

— On place le code d'erreur `FFh` dans `AL` pour le renvoyer en cas de format non valide (ligne 3604). Si l'heure est supérieure à 24, la minute à 60, la seconde à 60 ou le 1/100 ième de seconde à 100, on termine la routine avec le code d'erreur `FFh` (lignes 3604–3612).

— Sinon on appelle la pré-routine BIOSSETTIME de IO.COM pour mettre à jour l'heure dans les registres du minuteur, on place le code d'erreur `00h` dans `AL` et on termine la routine (lignes 3613–3615).

5.4 Routines de service des appels système généraux du système de fichiers

5.4.1 Routines de service des opérations sur le lecteur de disquette par défaut (18h, 1Ch, 0Eh, 19h)

5.4.1.1 Routine de service de la fonction 24 (18h) de demande d'informations sur le lecteur de disquette en cours

Cette fonction n'est pas implémentée pour MS-DOS 1.25. Sa routine de service, repérée par l'étiquette INUSE :

```
398 : INUSE:
399 : GETIO:
400 : SETIO:
401 : GETRDONLY:
402 : SETATTRIB:
403 : USERCODE:
404 :         MOV     AL,0
405 :         RET
```

se contente de renvoyer `00h` *via* le registre `AL` pour indiquer que tout s'est déroulé correctement.

Remarquons que les fonctions GETIO, SETIO avaient visiblement été prévues, mais qu'elles n'ont pas été retenues dans la version finale.

5.4.1.2 Routine de service de la fonction 29 (1Ch) de demande d'informations sur un lecteur de disquette

Comme pour la fonction précédente, sa routine de service, repérée par l'étiquette GETRDONLY, se contente de renvoyer `00h` *via* le registre `AL` pour indiquer que tout s'est déroulé correctement.

5.4.1.3 Routine de service de la fonction 14 (0Eh) de sélection du lecteur de disquette par défaut

La routine de service de la fonction 14 de l'interruption `21h` est repérée par l'étiquette SELDSK :

```
2698 : SELDSK: ;System call 14
2699 :         MOV     AL,CS:[NUMDRV]
2700 :         CMP     DL,AL
2701 :         JNB     RET17
2702 :         MOV     CS:[CURDRV],DL
2703 : RET17:  RET
```

On compare avec le numéro de lecteur de disquette en cours. S'il s'agit du même, on a terminé. Sinon on le spécifie.

La variable CURDRV spécifie le numéro du lecteur de disquette par défaut. Elle est initialisée au lecteur de disquette A :

```
3669 : CURDRV  DB      0               ;Default to drive A
```

5.4.1.4 Routine de service de la fonction 25 (19h) de détermination du lecteur de disquette par défaut

La routine de service de la fonction 25 de l'interruption `21h`, de détermination du lecteur de disquette par défaut, repérée par l'étiquette GETDRV :

```
2683 : GETDRV: ;System call 25
2684 :         MOV     AL,CS:[CURDRV]
2685 : RET15:  RET
```

renvoie dans `AL` le numéro du lecteur de disquette par défaut.

5.4.2 Routine de service de la fonction 26 (1Ah) de sélection de l'adresse de transfert (DTA)

La routine de service de la fonction 26 de l'interruption `21h`, de sélection de l'adresse de transfert (DTA), est repérée par l'étiquette SETDMA (pour *SET Disk Memory Address*) :

```
2611 : SETDMA: ;System call 26
2612 :         MOV     CS:[DMAADD],DX
2613 :         MOV     CS:[DMAADD+2],DS
2614 :         RET
```

Le décalage de la variable DMAADD est initialisée à `80h`, l'adresse de segment restant à préciser :

```
3654 : DMAADD  DW      80H             ;User's disk transfer address (disp/seg)
3655 :         DW      ?
```

5.4.3 Routine de service de la fonction 46 (2Eh) d'enclenchement de la vérification lors de l'écriture sur disque

La routine de service de la fonction `2Eh` de l'interruption `21h`, d'enclenchement de la vérification lors de l'écriture sur la disquette, est repérée par l'étiquette VERIFY :

```
407 : VERIFY:
408 :         AND     AL,1
409 :         MOV     CS:VERFLG,AL
410 :         RET
```

La variable VERFLG est initialisée à non vérification :

```
3652 : VERFLG  DB      0       ;Initialize with verify off
```

5.4.4 Routine de service de la fonction 13 (0Dh) de réinitialisation d'une disquette

5.4.4.1 Routine principale

La routine de service de la fonction 13 de l'interruption `21h`, de réinitialisation des paramètres d'une disquette en mémoire centrale, est repérée par l'étiquette DSKRESET :

```
2656 : DSKRESET: ;System call 13
2657 :         PUSH    CS
2658 :         POP     DS
2659 : WRTFATS:
2660 : ; DS=CS. Writes back all dirty FATs. All registers destroyed.
2661 :         XOR     AL,AL
2662 :         XCHG    AL,[DIRTYBUF]
2663 :         OR      AL,AL
2664 :         JZ      NOBUF
2665 :         MOV     BP,[BUFDRVBP]
2666 :         MOV     DX,[BUFSECNO]
2667 :         MOV     BX,[BUFFER]
2668 :         MOV     CX,1
2669 :         CALL    DWRITE
2670 : NOBUF:
2671 :         MOV     CL,[NUMIO]
2672 :         MOV     CH,0
2673 :         MOV     BP,[DRVTAB]
2674 : WRTFAT:
2675 :         PUSH    CX
2676 :         CALL    CHKFATWRT
2677 :         POP     CX
2678 :         ADD     BP,DPBSIZ
2679 :         LOOP    WRTFAT
2680 :         RET
```

— On confond le segment des données avec le segment de code (lignes 2657–2658).

— On commence par placer `00h` dans `AL`, que l'on échange avec le contenu de la variable DIRTYBUF : la variable DIRTYBUF indiquera désormais qu'il n'y a pas eu de modification sur la copie en mémoire centrale du secteur de la disqette depuis sa dernière sauvegarde et on trouve dans `AL` l'ancien contenu de cette variable (lignes 2659–2662).

La variable DIRTYBUF spécifie si le tampon a été modifié (`FFh` ou non (`00h` depuis sa dernière sauvegarde sur la disquette :

```
3661 : DIRTYBUF DB     0
```

— Si la sauvegarde du secteur se trouvant dans le tampon de secteur en mémoire centrale n'a pas été effectuée depuis la dernière modification de celui-ci, on place les contenus des variables BUFDRVBP, BUFSECNO et BUFFER dans les registres respectifs `BP`, `DX` et `BX`, 1 dans `CX`, pour indiquer qu'on ne veut écrire qu'un seul secteur, et on appelle la routine DWRITE pour l'enregistrer sur la disquette (lignes 2663-2670).

La variable BUFDRVBP contient le décalage de l'emplacement du tampon du bloc des paramètres des lecteurs de disquette :

```
3662 : BUFDRVBP DW     ?
```

— On place le nombre de lecteurs de disquette dans `CL`, 0 dans `CH`, on fait pointer `BP` sur la table des lecteurs de disquette, on sauvegarde le contenu de `CX` sur la pile et on appelle la routine CHKFATWRT, étudiée ci-dessous, pour sauvegarder sur la disquette la copie de la FAT se trouvant en mémoire centrale si elle ne l'a pas été depuis la dernière modification intervenue, et ceci pour chaque instance de la FAT sur la disquette, en fait toujours 2 pour MS-DOS (lignes 2671–2679).

5.4.4.2 Enregistrement conditionnel de la copie de la FAT en mémoire centrale

La routine CHKFATWRT enregistre sur la disquette la copie de la FAT se trouvant en mémoire centrale si elle ne l'a pas été depuis la dernière modification intervenue sur celle-ci :

```
1055 : CHKFATWRT:
1056 : ; Do FATWRT only if FAT is dirty and uses same I/O driver
1057 :          MOV     SI,[BP.FAT]
1058 :          MOV     AL,[BP.DEVNUM]
1059 :          MOV     AH,1
1060 :          CMP     [SI-2],AX       ;See if FAT dirty and uses same driver
1061 :          JNZ     OKRET
1062 :
1063 : FATWRT:
```

— Avant d'appeler la routine, `BP` doit pointer sur le bloc des paramètres du lecteur de disquette en cours. On fait pointer `SI` sur la copie de la FAT se trouvant en mémoire centrale de la disquette se trouvant dans ce lecteur de disquette (ligne 1057).

— On place le numéro de ce lecteur de disquette dans `AL` et 1 dans `AH` (lignes 1058–1059).

— Si on n'a pas sauvegardé la copie de la FAT en mémoire centrale depuis sa dernière modification, on passe à FATWRT pour la sauvegarder sur la disquette. Sinon on termine la routine, OKRET étant une partie de la routine FATWRT (lignes 1060–1063).

5.5 Lecture et écriture de secteurs logiques de la disquette

5.5.1 Implémentation de la structure d'un bloc des paramètres d'une disquette

La structure du bloc des paramètres d'une disquette est reprise par DPBLOCK :

```
126 : ; Field definition for Drive Parameter Block
127 :
128 : DPBLOCK STRUC
129 : DEVNUM  DB      ?       ;I/O driver number
130 : DRVNUM  DB      ?       ;Physical Unit number
131 : SECSIZ  DW      ?       ;Size of physical sector in bytes
132 : CLUSMSK DB      ?       ;Sectors/cluster - 1
133 : CLUSSHFT DB     ?       ;Log2 of sectors/cluster
134 : FIRFAT  DW      ?       ;Starting record of FATs
135 : FATCNT  DB      ?       ;Number of FATs for this drive
136 : MAXENT  DW      ?       ;Number of directory entries
137 : FIRREC  DW      ?       ;First sector of first cluster
138 : MAXCLUS DW      ?       ;Number of clusters on drive + 1
139 : FATSIZ  DB      ?       ;Number of records occupied by FAT
140 : FIRDIR  DW      ?       ;Starting record of directory
141 : FAT     DW      ?       ;Pointer to start of FAT
142 : DPBLOCK ENDS
143 :
144 : DPBSIZ  EQU     20      ;Size of the structure in bytes
145 : DIRSEC  =       FIRREC  ;Number of dir. sectors (init temporary)
146 : DSKSIZ  =       MAXCLUS ;Size of disk (temp used during init only)
```

— Le champ DEVNUM spécifie le nombre de lecteurs de disquette.
— Le champ DRVNUM spécifie le numéro du lecteur de disquette par défaut : 1 pour A et 2 pour B.
— Le champ SECSIZ spécifie la taille, en octets, d'un secteur.
— Le champ CLUSMSK spécifie le nombre de secteurs moins un d'une unité d'allocation.
— Le champ CLUSSHFT spécifie le nombre de secteurs d'une unité d'allocation.
— Le champ FIRFAT spécifie le numéro du premier secteur de la FAT.
— Le champ FATCNT spécifie le nombre de copies de la FAT sur la disquette. En fait il y en a toujours 2.
— Le champ FIRREC spécifie le numéro du premier secteur de la première unité d'allocation.
Il y a dû y avoir hésitation sur le nom, car on a REC pour *record* (*enregistrement*) et non CLUS pour *cluster*.
— Le champ MAXCLUS spécifie le nombre d'unités d'allocation du disque, plus un.
— Le champ FATSIZ spécifie le nombre de secteurs occupés par la FAT. Il s'agit toujours de 1 pour MS-DOS 1.25.
— Le champ FIRDIR spécifie le numéro du premier secteur du répertoire.
— Le champ FAT pointe sur le début de la copie de la FAT en mémoire centrale.

5.5.2 Lecture de plusieurs secteurs consécutifs de la disquette

La routine DSKREAD lit plusieurs secteurs (consécutifs) de la disquette. Avant d'appeler cette routine, `BP` doit pointer sur le bloc de paramètres de la disquette, `DS:BX` contenir l'adresse en mémoire centrale à laquelle transférer ce qui est lu, `CX` le nombre de secteurs à lire et `DX` le numéro du premier secteur à lire :

```
1290 : DSKREAD:
1291 :          MOV     AL,[BP.DEVNUM]
1292 :          PUSH    BP
1293 :          PUSH    BX
1294 :          PUSH    CX
1295 :          PUSH    DX
1296 :          CALL    FAR PTR BIOSREAD
1297 :          POP     DX
1298 :          POP     DI
1299 :          POP     BX
1300 :          POP     BP
1301 : RET9:    RET
```

On place le numéro de lecteur de disquette dans le registre `AL`, on sauvegarde sur la pile le contenu des registres que l'on va utiliser, on fait appel à la pré-routine de IO.COM de lecture de plusieurs secteurs (consécutifs) de la disquette et on restaure les valeurs des registres sauvegardées.

Après son exécution, d'après ce que nous avons vu sur la routine READ de IO.COM, donc BIOSREAD de MSDOS.COM, le drapeau CF est baissé si le transfert s'est effectué avec succès. Sinon il est levé, `CX` contient le nombre de secteurs qui n'ont pas été transférés et `AL` contient un code d'erreur :

— `02h` si la disquette n'est pas prête ;
— `04h` si le CRC n'est pas correct ;
— `06h` si la recherche du secteur a été infructueuse ;
— `08h` si le secteur n'a pas été trouvé :
— 12 pour une erreur autre que celles ci-dessus.

5.5.3 Traitement des erreurs de lecture-écriture sur une disquette

5.5.3.1 Routine principale

La routine HARDERR gère les erreurs des lecteurs de disquette. Avant de l'appeler, `DS:BX` doit pointer sur la zone d'échange entre la disquette et la mémoire centrale, `DX` contenir le numéro du premier secteur qui devait être lu, `CX` le nombre de secteurs qu'on a lu avec succès, `AX` le code d'erreur matérielle, `DI` le nombre de secteurs transférés, `BP` ointer sur le bloc de paramètres de la disquette et la variable READOP contenir la nature de l'opération : 0 pour une lecture, 1 pour une écriture.

La variable READOP, de capacité un octet, vaut 0 pour une lecture, 1 pour une écriture.

```
3699 : READOP  DB      ?
```

La routine de traitement des erreurs de lecture-écriture sur une disquette est repérée par l'étiquette HARDERR :

```
1239 : HARDERR:
1240 :
1241 : ;Hard disk error handler. Entry conditions:
1242 : ;       DS:BX = Original disk transfer address
1243 : ;       DX = Original logical sector number
1244 : ;       CX = Number of sectors to go (first one gave the error)
1245 : ;       AX = Hardware error code
1246 : ;       DI = Original sector transfer count
1247 : ;       BP = Base of drive parameters
1248 : ;       [READOP] = 0 for read, 1 for write
1249 :
1250 :         XCHG    AX,DI           ;Error code in DI, count in AX
1251 :         SUB     AX,CX           ;Number of sectors successfully transferred
1252 :         ADD     DX,AX           ;First sector number to retry
1253 :         PUSH    DX
1254 :         MUL     [BP.SECSIZ]     ;Number of bytes transferred
1255 :         POP     DX
1256 :         ADD     BX,AX           ;First address for retry
1257 :         MOV     AH,0            ;Flag disk section in error
1258 :         CMP     DX,[BP.FIRFAT]  ;In reserved area?
1259 :         JB      ERRINT
1260 :         INC     AH              ;Flag for FAT
1261 :         CMP     DX,[BP.FIRDIR]  ;In FAT?
1262 :         JB      ERRINT
1263 :         INC     AH
1264 :         CMP     DX,[BP.FIRREC]  ;In directory?
1265 :         JB      ERRINT
1266 :         INC     AH              ;Must be in data area
1267 : ERRINT:
1268 :         SHL     AH,1            ;Make room for read/write bit
1269 :         OR      AH,CS:[READOP]
1270 : FATAL:
1271 :         MOV     AL,[BP.DRVNUM]  ;Get drive number
1272 : FATAL1:
1273 :         PUSH    BP              ;The only thing we preserve
1274 :         MOV     CS:[CONTSTK],SP
1275 :         CLI                     ;Prepare to play with stack
1276 :         MOV     SS,CS:[SSSAVE]
1277 :         MOV     SP,CS:[SPSAVE]  ;User stack pointer restored
1278 :         INT     24H             ;Fatal error interrupt vector
1279 :         MOV     CS:[SPSAVE],SP
1280 :         MOV     CS:[SSSAVE],SS
1281 :         MOV     SP,CS
1282 :         MOV     SS,SP
```

```
1283 :          MOV     SP,CS:[CONTSTK]
1284 :          STI
1285 :          POP     BP
1286 :          CMP     AL,2
1287 :          JZ      ERROR
1288 :          RET
```

— Le code d'erreur, transmis dans `AX` par la pré-routine de IO.COM, est placé dans `DI` et le nombre de secteurs qu'on a lu avec succès dans le registre `AX` (lignes 1250 et 1251). On l'ajoute au numéro du premier secteur qui devait être lu, ce qui place le numéro de secteur à relire dans le registre `DX` (ligne 1252). On sauvegarde ce numéro sur la pile (ligne 1253). On le multiplie par la taille d'un secteur pour donner le nombre d'octets transférés avec succès, que l'on place dans le couple de registres `DX:AX` (ligne 1254). On restaure le contenu `DX` avec la valeur sauvegardée (ligne 1255). On place dans `BX` la première adresse à réessayer.

— On positionne le drapeau d'erreur de disque (ligne 1257). Si la partie qu'on n'a pas pu lire se trouve dans la partie réservée, on passe au traitement de l'erreur avec ce drapeau (lignes 1258 et 1259).

— On positionne le drapeau pour une erreur de FAT (ligne 1260). Si la partie qu'on n'a pas pu lire concerne la FAT, on passe au traitement d'erreur avec ce drapeau (lignes 1261 et 1262).

— On positionne le drapeau pour une erreur de lecture du répertoire (ligne 1263). Si la partie qu'on n'a pas pu lire concerne le répertoire, on passe au traitement d'erreur avec ce drapeau (lignes 1264 et 1265).

— Sinon on positionne le drapeau d'erreur de lecture dans les secteurs de données (ligne 1266).

— On traite l'erreur avec le drapeau adéquat. On indique qu'il s'agit d'une erreur de lecture (lignes 1268 et 1269). On place le numéro de lecteur de disquette dans le registre `AL` (lignes 1270 et 1271). On sauvegarde sur la pile le contenu du registre `BP`, pointant sur le bloc des paramètres du lecteur de disquette (ligne 1273). On sauvegarde la valeur du pointeur de pile dans la variable CONSTK (ligne 1274).

  La variable CONTSTK contient une sauvegarde de la valeur du pointeur de pile :

```
3694 : CONTSTK DW      ?
```

— On inhibe les interruptions masquables (ligne 1275). On restaure les registres de pile, pointeur `SP` et segment `SS`, de l'utilisateur (lignes 1276 et 1277). On appelle l'interruption `24h` de traitement d'une erreur critique. On sauvegarde les registres de pile, `SP` et `SS` (lignes 1279 et 1280). On confond le segment de pile avec le segment de code (lignes 1281 et 1282). Le pointeur de pile est restauré (ligne 1283). On se remet à l'écoute des interruptions masquables (ligne 1284). On restaure la valeur de `BP` (ligne 1285). Si le contenu de `AL` est `02h`, disquette non prête, on va à la partie ERROR, étudiée ci-dessous, et sinon on termine la routine (lignes 1286–1288).

  La routine de service de l'interruption `24h` de traitement d'une erreur critique ne sera implémentée qu'avec l'interpréteur de commandes.

5.5.3.2 La partie ERROR

La partie ERROR est :

```
1369 : ERROR:
1370 :          MOV     AX,CS
1371 :          MOV     DS,AX
1372 :          MOV     ES,AX
1373 :          CALL    WRTFATS
1374 :          XOR     AX,AX
1375 :          CLI
1376 :          MOV     SS,[SSSAVE]
1377 :          MOV     SP,[SPSAVE]
1378 :          MOV     DS,AX
1379 :          MOV     SI,EXIT
1380 :          MOV     DI,OFFSET DOSGROUP:EXITHOLD
1381 :          MOVSW
1382 :          MOVSW
1383 :          POP     AX
1384 :          POP     BX
1385 :          POP     CX
1386 :          POP     DX
1387 :          POP     SI
1388 :          POP     DI
1389 :          POP     BP
1390 :          POP     DS
1391 :          POP     ES
1392 :          STI             ;Stack OK now
1393 :          JMP     CS:DWORD PTR [EXITHOLD]
```

— On confond les segment des données et supplémentaire avec le segment de code (lignes 1370–1372).

— On appelle la routine WRTFATS, partie de la routine DSKRESET, pour enregistrer sur la disquette la copie de la FAT se trouvant en mémoire centrale si on ne l'avait pas fait depuis la dernière modification de celle-ci et pour enregistrer la copie du secteur se trouvant dans le tampon de secteur en mémoire centrale (ligne 1373).

— On inhibe les interruptions masquables, on revient à la pile antérieure, on fait commencer le segment des données au tout début de la mémoire centrale, on fait pointer `SI` sur le vecteur d'interruption de l'interruption `22h`, on place dans `DI` l'adresse de la variable EXITHOLD, dans laquelle on place les 4 octets du vecteur d'interruption de l'interruption `22h` (lignes 1374–1382).

  La variable EXITHOLD contient l'adresse de sortie :

  ```
  3681 : EXITHOLD DB     4 DUP (?)
  ```

— On restaure les valeurs des registres, on se remet à l'écoute des interruptions masquables et on appelle l'interruption `22h` pour terminer le programme en cours (lignes 1383–1393).

  La routine de service de l'interruption `22h` de terminaison d'un programme ne sera implémentée qu'avec l'interpréteur de commandes.

5.5.4 Lecture de plusieurs secteurs consécutifs de la disquette

La lecture d'un certain nombre de secteurs sur la disquette est effectuée par la routine DREAD. Avant de l'appeler, on fait pointer `DS:BX` sur la zone de transfert de ces secteurs en mémoire centrale, placer le nombre de secteurs à transférer dans le registre `CX`, le numéro du premier secteur à transférer dans le registre `DX` et faire pointer `BP` sur le bloc de paramètres de la disquette :

```
1215 : RET8:   RET
1216 :
1217 :
1218 : DREAD:
1219 :
1220 : ; Inputs:
1221 : ;       BX,DS = Transfer address
1222 : ;       CX = Number of sectors
1223 : ;       DX = Absolute record number
1224 : ;       BP = Base of drive parameters
1225 : ; Function:
1226 : ;       Calls BIOS to perform disk read. If BIOS reports
1227 : ;       errors, will call HARDERR for further action.
1228 : ; BP preserved. All other registers destroyed.
1229 :
1230 :         CALL    DSKREAD
1231 :         JNC     RET8
1232 :         MOV     CS:BYTE PTR [READOP],0
1233 :         CALL    HARDERR
1234 :         CMP     AL,1            ;Check for retry
1235 :         JZ      DREAD
1236 :         RET                     ;Ignore otherwise
```

— On appelle la pré-routine de IO.COM de lecture de secteurs (ligne 1230).

— Si aucune erreur ne s'est produite lors de cette lecture, on termine la routine (ligne 1231).

— Si, par contre, une erreur s'est produite, on indique qu'elle s'est produite lors d'une lecture de la disquette et on appelle la sous-routine HARDERR (lignes 1232 et 1233).

— Si le code d'erreur est `01h`, on essaie de lire à nouveau (lignes 1234 et 1235). Sinon on termine la routine.

5.5.5 Écriture de plusieurs secteurs consécutifs sur la disquette

La routine DWRITE écrit plusieurs secteurs (consécutifs) sur la disquette. Avant de l'appeler, on fait pointer `DS:BX` sur la zone depuis laquelle transférer le contenu de ces secteurs, on place le nombre de secteurs à transférer dans le registre `CX`, le numéro du premier secteur sur lequel transférer dans le registre `DX` et on fait pointer `BP` sur le bloc des paramètres du lecteur de disquette :

```
1301 : RET9:   RET
[...]
1325 : DWRITE:
1326 :
1327 : ; Inputs:
1328 : ;       BX,DS = Transfer address
1329 : ;       CX = Number of sectors
1330 : ;       DX = Absolute record number
1331 : ;       BP = Base of drive parameters
1332 : ; Function:
1333 : ;       Calls BIOS to perform disk write. If BIOS reports
1334 : ;       errors, will call HARDERR for further action.
1335 : ; BP preserved. All other registers destroyed.
1336 :
1337 :         MOV     AL,[BP.DEVNUM]
1338 :         MOV     AH,CS:VERFLG
1339 :         PUSH    BP
1340 :         PUSH    BX
1341 :         PUSH    CX
1342 :         PUSH    DX
1343 :         CALL    FAR PTR BIOSWRITE
1344 :         POP     DX
1345 :         POP     DI
1346 :         POP     BX
1347 :         POP     BP
1348 :         JNC     RET9
1349 :         MOV     CS:BYTE PTR [READOP],1
1350 :         CALL    HARDERR
1351 :         CMP     AL,1            ;Check for retry
1352 :         JZ      DWRITE
1353 :         RET
```

— On place le numéro de lecteur de disquette dans le registre `AL` et la valeur du drapeau de vérification dans le registre `AH` (lignes 1337 et 1338).

— On sauvegarde sur la pile les contenus des registres `BP`, `BX`, `CX` et `DX`, on appelle la pré-routine de IO.COM d'écriture de secteurs et on restaure les valeurs des registres (lignes 1339–1347).

— S'il n'y a pas eu d'erreur lors de l'écriture, on termine la routine (ligne 1348). Sinon on positionne le drapeau READOP à écriture et on affiche le type d'erreur en appelant la sous-routine HARDERR (lignes 1349 et 1350).

— Si le code d'erreur est `01h`, on essaie d'écrire à nouveau. Sinon on termine la routine (lignes 1351–1353).

5.6 Routines de service concernant le répertoire

5.6.1 Rappel de la structure d'une entrée de répertoire

La structure d'une entrée de répertoire est rappelée au début du fichier MSDOS.COM :

```
99  : ; Description of 32-byte directory entry (same as returned by SEARCH FIRST
100 : ; and SEARCH NEXT, functions 17 and 18).
101 : ;
102 : ; Location      bytes   Description
103 : ;
104 : ;    0          11      File name and extension ( 0E5H if empty)
105 : ;   11           1      Attributes. Bits 1 or 2 make file hidden
106 : ;   12          10      Zero field (for expansion)
107 : ;   22           2      Time. Bits 0-4=seconds/2, bits 5-10=minute, 11-15=hour
108 : ;   24           2      Date. Bits 0-4=day, bits 5-8=month, bits 9-15=year-1980
109 : ;   26           2      First allocation unit ( < 4080 )
110 : ;   28           4      File size, in bytes (LSB first, 30 bits max.)
111 : ;
```

5.6.2 Préparation des registres pour une lecture-écriture d'un secteur du répertoire

Avant de lire ou d'écrire un secteur du répertoire, le registre `BP` pointant sur le bloc de paramètres du lecteur de disquette concerné et `AL` contenant le numéro (relatif au répertoire) du secteur du répertoire à lire ou sur lequel écrire, la routine DIRCOMP place le numéro absolu du secteur dans le registre `DX`, le décalage de DIRBUF, tampon à lire ou à écrire, dans `BX` et 1 dans `CX` (pour indiquer qu'il faut lire ou écrire un seul secteur) :

```
1113 : DIRCOMP:
1114 : ; Prepare registers for directory read or write
1115 :         CBW
1116 :         ADD     AX,[BP.FIRDIR]
1117 :         MOV     DX,AX
1118 :         MOV     BX,OFFSET DOSGROUP:DIRBUF
1119 :         MOV     CX,1
1120 :         RET
```

— On étend le numéro (relatif) de secteur du répertoire à lire ou sur lequel écrire, contenu dans le registre `AL`, en un mot, contenu dans le registre `AX`, auquel on ajoute le numéro du premier secteur du répertoire, pour obtenir le numéro absolu du secteur à lire ou sur lequel écrire, que l'on place dans `DX` (lignes 1115–1117).

— On place dans `BX` le décalage de l'étiquette DIRBUF, qui repère le tampon (ligne 1118).

L'étiquettee DIRBUF (pour adresse du 'tampon d'un secteur du répertoire') repère le tampon en mémoire centrale d'un secteur du répertoire :

```
3728 : DIRBUF LABEL    WORD
```

— On place 1 dans `CX` pour spécifier qu'il n'y a qu'un seul secteur à lire ou sur lequel écrire (ligne 1119).

5.6.3 Écriture d'un secteur du répertoire

Le répertoire occupe plusieurs secteurs, dont le nombre et les numéros logiques dépendent du type de disquette (simple face ou double face pour MS-DOS 1.25). La donnée du numéro (relatif) du secteur du répertoire sur lequel écrire et du bloc de paramètres du lecteur de disquette

correspondant sont les paramètres suffisants pour spécifier le secteur qu'il faut écrire sur la disquette.

La routine DIRWRITE permet d'écrire un secteur du répertoire. Avant son appel, le segment des données doit être confondu avec le segment de code, le registre `AL` doit contenir le numéro (relatif) du secteur du répertoire sur lequel écrire et `BP` doit pointer sur le bloc de paramètres de la disquette sur laquelle se trouve le répertoire. La routine lit le secteur concerné depuis l'emplacement en mémoire centrale de décalage DIRBUF :

```
1308 : DIRWRITE:
1309 :
1310 : ; Inputs:
1311 : ;       DS = CS
1312 : ;       AL = Directory block number
1313 : ;       BP = Base of drive parameters
1314 : ; Function:
1315 : ;       Write the directory block into DIRBUF.
1316 : ; Outputs:
1317 : ;       BP unchanged
1318 : ; All other registers destroyed.
1319 :
1320 :         MOV     BYTE PTR [DIRTYDIR],0
1321 :         MOV     AL,BYTE PTR [DIRBUFID]
1322 :         CALL    DIRCOMP
1323 :
1324 :
1325 : DWRITE:
```

— On place zéro dans la variable DIRTYDIR pour indiquer qu'on a sauvegardé sur la disquette les modifications effectuées sur la copie en mémoire centrale du secteur du répertoire qui s'y trouve.

La variable DIRTYDIR est égale à `FFh` pour indiquer qu'on a sauvegardé sur la disquette les modifications effectuées sur la copie en mémoire centrale du secteur du répertoire qui s'y trouve :

```
3649 : DIRTYDIR DB     0               ;Dirty buffer flag
```

— On place le numéro de secteur du répertoire sur lequel écrire dans `AL` (ligne 1321)

La variable DIRBUFID contient le numéro de secteur du répertoire dont la copie en mémoire est actuellement contenue dans le tampon de secteur du répertoire :

```
3663 : DIRBUFID DW     -1
```

— On appelle la routine DIRCOMP pour préparer les registres avant d'écrire sur le répertoire (ligne 1322).

Le registre `BP` pointant sur le bloc de paramètre du lecteur de disquette concerné et `AL` contenant le numéro (relatif) du secteur du répertoire sur lequel écrire, la routine DIRCOMP place dans le registre `DX` le numéro absolu du secteur, dans `BX` le décalage de la variable DIRBUF et 1 dans `CX` (pour indiquer qu'il faut écrire un seul secteur).

— Puisque la routine DWRITE est suivie de la routine DWRITE, celle-ci est exécutée (ligne 1325).

Rappelons que l'adresse depuis laquelle transférer le contenu de ce secteur est `DS:BX`, que le nombre de secteurs à transférer est contenu dans le registre `CX`, qu'on en a donc bien un seul, que le numéro logique du secteur sur lequel il faut écrire est contenu dans le registre `DX` et que la structure des paramètres du lecteur de disquette est repérée par `BP`.

5.6.4 Lecture d'un secteur du répertoire

La routine DIRREAD lit un secteur du répertoire. Avant son appel, le segment des données doit être confondu avec le segment de code, le registre `AL` contient le numéro (relatif) du secteur du répertoire à lire et `BP` pointe sur le bloc de paramètres de la disquette sur laquelle se trouve le répertoire. La sous-routine lit le secteur concerné et le place en mémoire centrale à l'emplacement pointé par DIRBUF :

```
1194 : DIRREAD:
1195 :
1196 : ; Inputs:
1197 : ;       DS = CS
1198 : ;       AL = Directory block number
1199 : ;       BP = Base of drive parameters
1200 : ; Function:
1201 : ;       Read the directory block into DIRBUF.
1202 : ; Outputs:
1203 : ;       AX,BP unchanged
1204 : ; All other registers destroyed.
1205 :
1206 :         PUSH    AX
1207 :         CALL    CHKDIRWRITE
1208 :         POP     AX
1209 :         PUSH    AX
1210 :         MOV     AH,[BP.DEVNUM]
1211 :         MOV     [DIRBUFID],AX
1212 :         CALL    DIRCOMP
1213 :         CALL    DREAD
1214 :         POP     AX
1215 : RET8:   RET
```

— On sauvegarde le contenu du registre `AX` sur la pile, car il va être détruit par la sous-routine appelée ligne suivante. On appelle la sous-routine CHKDIRWRITE pour savoir si la copie en mémoire centrale du secteur du répertoire se trouvant actuellement dans le tampon d'un secteur du répertoire a été modifié depuis la dernière sauvegarde, en l'écrivant si besoin est, puis on restaure la valeur de `AX` (lignes 1206-1208).

La sous-routine CHKDIRWRITE est très simple :

```
1301 : RET9:   RET
1302 :
1303 :
1304 : CHKDIRWRITE:
1305 :         TEST    BYTE PTR [DIRTYDIR],-1
1306 :         JZ      RET9
1307 :
1308 : DIRWRITE:
```

— Si la variable DIRTYDIR est `FFh`, il n'y a pas eu de modifications depuis la dernière sauvegarde. On termine donc la sous-routine.

— Sinon il faut en sauvegarder la copie, ce que l'on fait grâce à DIRWRITE.

— On sauvegarde à nouveau le contenu de `AX` sur la pile, on place dans le registre `AH` le numéro de lecteur de la disquette dont on veut lire le répertoire et on sauvegarde dans la variable DIRBUFID le numéro de secteur du répertoire dont on va placer le contenu dans le tampon en mémoire centrale (lignes 1209–1211).

— On appelle la routine DIRCOMP pour initialiser les registres avant l'opération de lecture (ligne 1212).

— On appelle la routine DREAD pour lire le secteur, on restaure la valeur de `AX` et on termine la routine (lignes 1213–1215).

5.6.5 Recherche de l'entrée suivante du répertoire

5.6.5.1 Première façon

La variable LASTENT (pour adresse du numéro de la dernière entrée de répertoire trouvée) spécifie le numéro de la dernière entrée de répertoire lue :

```
3680 : LASTENT DW      ?
```

La routine GETENTRY donne l'entrée suivante du répertoire, le numéro de la dernière entrée lue étant spécifiée par la variable LASTENT.

Avant d'appeler cette routine, `BP` doit pointer sur le bloc de paramètres de la disquette sur laquelle se trouve le répertoire.

Après exécution, le drapeau CF est levé si aucune entrée nouvelle n'a été trouvée (autrement dit si on pointait sur la dernière entrée). Sinon `AL` contient le numéro (relatif) du secteur du répertoire sur lequel se trouve la nouvelle entrée, `BX` pointe sur la nouvelle entrée dans la zone de mémoire centrale repérée par DIRBUF, `DX` pointe sur le premier octet suivant DIRBUF et LASTENT donne le numéro de cette nouvelle entrée de répertoire :

```
632 : GETENTRY:
633 :
634 : ; Inputs:
635 : ;       [LASTENT] has previously searched directory entry
636 : ; Function:
637 : ;       Locates next sequential directory entry in preparation for search
638 : ; Outputs:
639 : ;       Carry set if none
640 : ;       ELSE
641 : ;       AL = Current directory block
642 : ;       BX = Pointer to next directory entry in [DIRBUF]
643 : ;       DX = Pointer to first byte after end of DIRBUF
644 : ;       [LASTENT] = New directory entry number
645 :
646 :         MOV     AX,[LASTENT]
647 :         INC     AX                      ;Start with next entry
648 :         CMP     AX,[BP.MAXENT]
649 :         JAE     NONE
650 : GETENT:
651 :         MOV     [LASTENT],AX
652 :         MOV     CL,4
653 :         SHL     AX,CL
654 :         XOR     DX,DX
655 :         SHL     AX,1
656 :         RCL     DX,1                    ;Account for overflow in last shift
657 :         MOV     BX,[BP.SECSIZ]
658 :         AND     BL,255-31               ;Must be multiple of 32
659 :         DIV     BX
660 :         MOV     BX,DX                   ;Position within sector
661 :         MOV     AH,[BP.DEVNUM]          ;AL=Directory sector no.
662 :         CMP     AX,[DIRBUFID]
663 :         JZ      HAVDIRBUF
664 :         PUSH    BX
665 :         CALL    DIRREAD
666 :         POP     BX
667 : HAVDIRBUF:
668 :         MOV     DX,OFFSET DOSGROUP:DIRBUF
669 :         ADD     BX,DX
670 :         ADD     DX,[BP.SECSIZ]
671 :         RET
```

— On incrémente le numéro d'entrée de répertoire fourni (lignes 646 et 647). Si on dépasse le nombre maximum d'entrées du répertoire de la disquette, on vérifie si la copie du secteur du répertoire en mémoire a été modifié, on lève le drapeau CF, pour indiquer qu'il n'existe pas d'entrée suivante, et on termine la routine (lignes 648–649 et 698–701) :

```
698 : NONE:
699 :          CALL    CHKDIRWRITE
700 :          STC
701 : RET4:    RET
```

— Sinon on place le nouveau numéro dans la variable LASTENT (ligne 651), on multiplie le contenu de `AX` par 32 (lignes 652 et 653), puiqu'une entrée occupe 32 octets, et on place la retenue éventuelle dans le registre `DX` (lignes 654–656). On divise `DX:AX` par la taille d'un secteur (lignes 657–659) pour obtenir le numéro de secteur du répertoire dans laquelle se trouve la nouvelle entrée, que l'on place dans le registre `BX` (ligne 660).

— On place dans le registre `AH` le numéro de lecteur de la disquette et si le contenu de `AX` est égal à `FFh`, on lit le secteur suivant du répertoire (lignes 662–666).

— On fait pointer `BX` sur la nouvelle entrée (lignes 668 et 669).

— On fait pointer `DX` sur le premier octet suivant DIRBUF (ligne 670).

5.6.5.2 Deuxième façon de recherche de l'entrée de répertoire suivante

La routine NEXTENTRY permet d'obtenir l'entrée de répertoire suivante.

Avant d'appeler la routine, un numéro d'entrée de répertoire est fourni dans LASTENT, `BP` pointe sur le bloc de paramètres de la disquette sur laquelle se trouve le répertoire, `AL` contient le numéro de secteur du répertoire sur lequel se trouve l'entrée, `BX` pointe sur l'entrée dans la copie en mémoire centrale du secteur, `DX` pointe sur le premier octet suivant DIRBUF.

Cette routine met à jour `AL`, `BX` et la zone pointée par LASTENT pour l'entrée suivante si elle existe, lève le drapeau CF sinon :

```
673 : NEXTENTRY:
674 :
675 : ; Inputs:
676 : ;       Same as outputs of GETENTRY, above
677 : ; Function:
678 : ;       Update AL, BX, and [LASTENT] for next directory entry.
679 : ;       Carry set if no more.
680 :
681 :         MOV     DI,[LASTENT]
682 :         INC     DI
683 :         CMP     DI,[BP.MAXENT]
684 :         JAE     NONE
685 :         MOV     [LASTENT],DI
686 :         ADD     BX,32
687 :         CMP     BX,DX
688 :         JB      HAVIT
689 :         INC     AL                      ;Next directory sector
690 :         PUSH    DX                      ;Save limit
691 :         CALL    DIRREAD
692 :         POP     DX
693 :         MOV     BX,OFFSET DOSGROUP:DIRBUF
694 : HAVIT:
695 :         CLC
696 :         RET
697 :
698 : NONE:
699 :         CALL    CHKDIRWRITE
700 :         STC
701 : RET4:   RET
```

— On place le numéro d'entrée contenu dans la variable LASTENT dans `DI` et on l'incrémente (lignes 681 et 682).

— Comme pour la routine prédédente, si on dépasse le nombre maximum d'entrées du répertoire de la disquette, on vérifie si la copie du secteur répertoire a été modifiée, on lève le drapeau CF et on termine la routine (lignes 683–684 et 698–701).

— Sinon on place ce nouveau numéro dans la variable LASTENT (ligne 685) et on ajoute 32 à `BX` (ligne 686), puiqu'une entrée occupe 32 octets.

— Si `BX` pointe alors sur le premier octet suivant la zone DIRBUF, autrement dit qu'on a épuisé le secteur de répertoire en cours, on incrémente `AL` pour indiquer qu'on passe au secteur suivant du répertoire, on sauvegarde le contenu de `DX`, c'est-à-dire le début de la zone suivant DIRBUF, sur la pile, on lit le secteur suivant du répertoire depuis la disquette, on restaure le contenu de `DX` et on fait pointer `BX` au début de ce nouveau secteur de répertoire, soit au début de DIRBUF mis à jour (lignes 687–693).

— On a alors mis à jour `AL`, `BX` et LASTENT. On baisse le drapeau CF et on termine la routine (lignes 695–696).

5.6.6 Routine de service de la fonction 17 (11h) de recherche de la première entrée du répertoire concordant avec un nom

La routine de service de la fonction 17 de l'interruption `21h`, de recherche de la première entrée du répertoire dont le champ 'nom de fichier' concorde avec le champ 'nom de fichier' d'un FCB fourni en paramètre, est repérée par SRCHFRST :

```
2494:SRCHFRST: ;System call 17
2495:        CALL    GETFILE
2496:SAVPLCE:
2497:; Search-for-next enters here to save place and report
2498:; findings.
2499:        JC      KILLSRCH
2500:        OR      BH,BH
2501:        JS      SRCHDEV
2502:        MOV     AX,[LASTENT]
2503:        MOV     ES:[DI.FILDIRENT],AX
2504:        MOV     ES:[DI.DRVBP],BP
2505:;Information in directory entry must be copied into the first
2506:; 33 bytes starting at the disk transfer address.
2507:        MOV     SI,BX
2508:        LES     DI,DWORD PTR [DMAADD]
2509:        MOV     AX,00FFH
2510:        CMP     AL,[EXTFCB]
2511:        JNZ     NORMFCB
2512:        STOSW
2513:        INC     AL
2514:        STOSW
2515:        STOSW
2516:        MOV     AL,[ATTRIB]
2517:        STOSB
2518:NORMFCB:
2519:        MOV     AL,[THISDRV]
2520:        INC     AL
2521:        STOSB   ;Set drive number
2522:        MOV     CX,16
2523:        REP     MOVSW           ;Copy remaining 10 characters of name
2524:        XOR     AL,AL
2525:        RET
2526:
2527:KILLSRCH:
2528:KILLSRCH1       EQU     KILLSRCH+1
2529:;The purpose of the KILLSRCH1 label is to provide a jump label to the following
2530:;   instruction which leaves out the segment override.
2531:        MOV     WORD PTR ES:[DI.FILDIRENT],-1
2532:        MOV     AL,-1
2533:        RET
2534:
2535:SRCHDEV:
2536:        MOV     ES:[DI.FILDIRENT],BX
2537:        LES     DI,DWORD PTR [DMAADD]
2538:        XOR     AX,AX
2539:        STOSB                   ;Zero drive byte
2540:        SUB     SI,4            ;Point to device name
2541:        MOVSW
2542:        MOVSW
2543:        MOV     AX,2020H
2544:        STOSB
2545:        STOSW
2546:        STOSW
2547:        STOSW                   ;Fill with 8 blanks
2548:        XOR     AX,AX
```

```
2549:         MOV     CX,10
2550:         REP     STOSW
2551:         STOSB
2552:RET14:   RET
```

— `DS:DX` pointe sur un FCB, contenant un nom de fichier pouvant comprendre des caractères d'ambiguïté ' ?'. On appelle la routine GETFILE pour rechercher une entrée dans le répertoire, le FCB étant passé en paramètre à l'adresse `DS:DX`, dont le champ 'nom de fichier' concorde (ligne 2494).

Le drapeau CF est levé s'il n'existe pas de fichier de nom cooncordant dans le répertoire. Sinon le drapeau de zéro est baissé, `BP` pointe sur le bloc des paramètres du lecteur de disquette concerné, les segments de données et supplémentaire sont confondus avec le segment de code, `SI` contient le numéro de la première unité d'allocation du fichier, DIRBUF pointe sur l'entrée de répertoire correspondante dans la copie en mémoire centrale, la variable chaîne de caractères NAME1 contient le nom du fichier, tout en majuscules, `ES:DI` pointe sur le FCB et `BX` sur le tampon du répertoire en mémoire centrale.

— Si on n'a trouvé aucune entrée de nom concordant, on place `FFh` dans le champ 'numéro d'entrée dans le répertoire' du FCB et comme code d'erreur dans `AL`, puis on termine la routine (lignes 2499 et 2527–2533).

Pour SEARCH FIRST et SEARCH NEXT le champ FILDIRENT de la structure FCBLOCK de FCB se substitue au champ FILSIZ du FCB :

```
97 : FILDIRENT       = FILSIZ                ;Used only by SEARCH FIRST and SEARCH NEXT
```

— Si le bit de poids fort du pointeur sur le tampon du répertoire est égal à 1, on le fait pointer sur le tampon du répertoire, contenu dans `BX`, comme champ 'taille du fichier' du FCB, on fait pointer `DI` au début de la zone de transfert, on place 0 comme octet de numéro de lecteur de disquette dans la zone de transfert, puis comme nom de périphérique (4 octets), 7 caractères espace comme nom de fichier, suivi de 21 caractères nuls et on termine la routine (lignes 2500-2501 et 2535–2552).

— Sinon on place le numéro de l'entrée trouvée dans le champ FILDIRENT du FCB, le pointeur sur le bloc des paramètres du lecteur de disquette dans son champ DRVBP, on copie les informations de l'entrée dans les 33 premiers octets de la zone de transfert et on termine la routine (lignes 2502–2525).

5.6.7 Routine de service de la fonction 18 (12h) de recherche de l'entrée suivante du répertoire concordant avec un nom

La routine de service de la fonction 18 de l'interruption `21h`, de recherche de l'entrée suivante du répertoire dont le champ 'nom de fichier' concorde avec le champ 'nom de fichier' d'un FCB fourni en paramètre, est repérée par SRCHNXT :

```
2554:SRCHNXT: ;System call 18
2555:         CALL    MOVNAME
2556:         MOV     DI,DX
2557:         JC      NEAR PTR KILLSRCH1
2558:         MOV     BP,[DI.DRVBP]
2559:         MOV     AX,[DI.FILDIRENT]
2560:         OR      AX,AX
2561:         JS      NEAR PTR KILLSRCH1
2562:         PUSH    DX
2563:         PUSH    DS
2564:         PUSH    CS
2565:         POP     DS
2566:         MOV     [LASTENT],AX
2567:         CALL    CONTSRCH
2568:         POP     ES
2569:         POP     DI
2570:         JMP     SAVPLCE
```

— `DS:DX` pointe sur un FCB, contenant un nom de fichier pouvant comprendre des caractères d'ambiguïté ' ?'. On appelle la routine MOVNAME pour que `DS:DX` pointe sur le FCB standard associé si le FCB passé en paramètre est un FCB étendu, confondre le segment supplémentaire avec le segment de code, placer dans la variable chaîne de caractères NAME1 le nom de fichier contenu dans le FCB, le tout en majuscules, et faire pointer `BP` sur le bloc des paramètres du lecteur de disquette spécifié par le FCB (ligne 2555).

— On fait pointer `DI` sur le FCB standard et, si on n'a trouvé aucune nouvelle entrée de nom concordant, on place `FFh` dans le champ 'numéro d'entrée dans le répertoire' du FCB et comme code d'erreur dans `AL`, puis on termine la routine (lignes 2556–2557 et 2527–2533).

— On fait pointer `BP` sur le bloc des paramètres du lecteur de disquette spécifié par le FCB et on place le dernier numéro trouvé d'entrée du répertoire dont le nom de fichier concorde avec celui passé en paramètre dans `AX` (lignes 2558–2559).

— Si le bit de poids fort de cette entrée est égal à 1, on place `FFh` dans le champ 'numéro d'entrée dans le répertoire' du FCB et comme code d'erreur dans `AL`, puis on termine la routine (lignes 2560-2561 et 2535–2552).

— Sinon on sauvegarde sur la pile l'adresse du FCB standard, on confond le segment des données avec le segment de code et on place dans la variable LASTENT le numéro d'entrée dans le répertoire que l'on vient de trouver (lignes 2562–2566).

— On appelle la routine CONTSRCH, partie de la routine GETNAME, pour faire pointer `BP` sur le bloc des paramètres du lecteur de disquette concerné, confondre les segments de données et supplémentaire avec le segment de code, placer dans `SI` le numéro de la première unité d'allocation du fichier, faire pointer DIRBUF sur l'entrée de répertoire dans sa copie en mémoire centrale et placer dansNAME1 le nom du fichier, tout en majuscules (ligne 2567).

— On fait pointer `ES:DI` sur le bloc des paramètres du lecteur de disquette et on va à SAVPLCE, au début de SRCHFRST (lignes 2568–2570).

5.7 Opérations sur la table d'allocation des fichiers (FAT)

5.7.1 Rappel de la structure d'une entrée de la FAT

La structure d'une entrée de la FAT est rappelée au début du fichier MSDOS.COM :

```
112 : ; The File Allocation Table uses a 12-bit entry for each allocation unit on
113 : ; the disk. These entries are packed, two for every three bytes. The contents
114 : ; of entry number N is found by 1) multiplying N by 1.5; 2) adding the result
115 : ; to the base address of the Allocation Table; 3) fetching the 16-bit word at
116 : ; this address; 4) If N was odd (so that N*1.5 was not an integer), shift the
117 : ; word right four bits; 5) mask to 12 bits (AND with 0FFF hex). Entry number
118 : ; zero is used as an end-of-file trap in the OS and as a flag for directory
119 : ; entry size (if SMALLDIR selected). Entry 1 is reserved for future use. The
120 : ; first available allocation unit is assigned entry number two, and even
121 : ; though it is the first, is called cluster 2. Entries greater than 0FF8H are
122 : ; end of file marks; entries of zero are unallocated. Otherwise, the contents
123 : ; of a FAT entry is the number of the next cluster in the file.
```

5.7.2 Préparation à l'échange entre la FAT sur disque et sa copie en mémoire

Le registre `BP` pointant sur le bloc des paramètres du lecteur de disquette souhaité, la routine FIGFAT initialise les registres nécessaires à l'échange entre la FAT, se trouvant sur la disquette (en fait sa première instance, puisqu'il y en a deux), et sa copie en mémoire centrale :

```
1102 : FIGFAT:
1103 : ; Loads registers with values needed to read or
1104 : ; write a FAT.
1105 :         MOV     AL,[BP.FATCNT]
1106 :         MOV     BX,[BP.FAT]
1107 :         MOV     CL,[BP.FATSIZ]  ;No. of records occupied by FAT
1108 :         MOV     CH,0
1109 :         MOV     DX,[BP.FIRFAT]  ;Record number of start of FATs
1110 :         RET
```

— On initialise le registre `AL` avec le nombre de copies de la FAT qui doit être présent sur sur la disquette.
C'est toujours 2 pour MS-DOS.

— On initialise le registre `BX` avec le décalage du début de la copie de la FAT en mémoire centrale.

— On initialise le registre `CL` avec le nombre de secteurs occupés par la FAT.
C'est toujours 1 pour MS-DOS 1.25.

— On initialise le registre `CH` avec 0 pour que ce soit le registre `CX` qui contienne ce nombre de secteurs.

— On initialise le registre `DX` avec le numéro du secteur du début de la FAT sur le disque.
C'est en fait le numéro du secteur de la première instance de la FAT sur le disque.

5.7.3 Sauvegarde sur la disquette de la copie de la FAT en mémoire centrale

Le segment des donnés étant confondu avec le segment de code et `BP` pointant sur le bloc des paramètres du lecteur de disquette désiré, la routine FATWRT enregistre la copie de la FAT située en mémoire centrale sur la disquette, indique qu'il n'y a pas eu de modification sur cette copie depuis sa dernière sauvegarde sur la disquette et renvoie 0 sur le registre `AL` :

```
1063 : FATWRT:
1064 :
1065 : ; Inputs:
1066 : ;       DS = CS
1067 : ;       BP = Base of drive parameter table
1068 : ; Function:
1069 : ;       Write the FAT back to disk and reset FAT
1070 : ;       dirty bit.
1071 : ; Outputs:
1072 : ;       AL = 0
1073 : ;       BP unchanged
1074 : ; All other registers destroyed
1075 :
1076 :         CALL    FIGFAT
1077 :         MOV     BYTE PTR [BX-1],0
1078 : EACHFAT:
1079 :         PUSH    DX
1080 :         PUSH    CX
1081 :         PUSH    BX
1082 :         PUSH    AX
1083 :         CALL    DWRITE
1084 :         POP     AX
1085 :         POP     BX
1086 :         POP     CX
1087 :         POP     DX
1088 :         ADD     DX,CX
1089 :         DEC     AL
1090 :         JNZ     EACHFAT
1091 : OKRET:
1092 :         MOV     AL,0
1093 :         RET
```

— On appelle la routine FIGFAT pour initialiser les registres nécessaires, tel que nous venons de le voir (ligne 1076).

— On indique que la copie de la FAT en mémoire centrale a été sauvegardée sur le disque en plaçant `00h` sur l'octet précédent la copie de la FAT en mémoire centrale (ligne 1077).

— On sauvegarde sur la pile les contenus des registres `DX`, `CX`, `BX` et `AX`, qui vont être détruits par l'appel à la routine d'écriture sur la disquette, on appelle la routine d'écriture sur la disquette puis on en restaure les contenus des registres qui ont été sauvegardés (lignes 1078–1087).

Rappelons que l'adresse du tampon depuis lequel écrire sur la disquette est spécifiée par `DS:BX`, que le nombre de secteurs à écrire est contenu dans le registre `CX` (toujours 1 pour MS-DOS 1.25) et que le numéro du premier secteur sur lequel il faut écrire est contenu dans le registre `DX`.

— On réinitialise le numéro du premier secteur sur lequel il faut écrire à la valeur antérieure plus le nombre de secteurs occupés par la FAT (mais il n'y en a qu'un pour MS-DOS 1.25) et on recommence pour sauvegarder la copie de la FAT sur la deuxième instance de la FAT sur la disquette, et plus généralement pour le nombre d'instances qu'il faut (lignes 1088–1090), mais il y en a toujours 2 pour MS-DOS.

— On renvoie 0 *via* le registre `AL` et on termine la routine (lignes 1091–1093).

5.7.4 Obtention d'une copie de la FAT en mémoire centrale

Le segment des donnés étant confondu avec le segment de code, la routine FATREAD transfère, si la disquette a été changée, une copie de la FAT située sur cette dernière en mémoire centrale et `BP` pointe sur le bloc des paramètres du lecteur de disquette correspondant. Le drapeau CF est levé si un lecteur non valide est renvoyé par la pré-routine MAPDEV de IO.COM :

```
926  : FATERR:
927  :         XCHG    AX,DI           ;Put error code in DI
928  :         MOV     AH,2            ;While trying to read FAT
929  :         MOV     AL,[THISDRV]    ;Tell which drive
930  :         CALL    FATAL1
931  :         JMP     SHORT FATREAD
932  : STARTSRCH:
933  :         MOV     AX,-1
934  :         MOV     [LASTENT],AX
935  :         MOV     [ENTFREE],AX
936  : FATREAD:
937  :
938  : ; Inputs:
939  : ;       DS = CS
940  : ; Function:
941  : ;       If disk may have been changed, FAT is read in and buffers are
942  : ;       flagged invalid. If not, no action is taken.
943  : ; Outputs:
944  : ;       BP = Base of drive parameters
945  : ;       Carry set if invalid drive returned by MAPDEV
946  : ; All other registers destroyed
947  :
948  :         MOV     AL,[THISDRV]
949  :         XOR     AH,AH                   ;Set default response to zero & clear carry
950  :         CALL    FAR PTR BIOSDSKCHG      ;See what BIOS has to say
951  :         JC      FATERR
952  :         CALL    GETBP
953  :         MOV     AL,[THISDRV]            ;Use physical unit number
954  :         MOV     SI,[BP.FAT]
955  :         OR      AH,[SI-1]               ;Dirty byte for FAT
956  :         JS      NEWDSK                  ;If either say new disk, then it's so
957  :         JNZ     MAPDRV
958  :         MOV     AH,1
959  :         CMP     AX,WORD PTR [BUFDRVNO]  ;Does buffer have dirty sector of this drive?
960  :         JZ      MAPDRV
961  : NEWDSK:
962  :         CMP     AL,[BUFDRVNO]           ;See if buffer is for this drive
963  :         JNZ     BUFOK                   ;If not, don't touch it
964  :         MOV     [BUFSECNO],0            ;Flag buffers invalid
965  :         MOV     WORD PTR [BUFDRVNO],00FFH
966  : BUFOK:
967  :         MOV     [DIRBUFID],-1
968  :         CALL    FIGFAT
969  : NEXTFAT:
970  :         PUSH    AX
971  :         CALL    DSKREAD
972  :         POP     AX
973  :         JC      BADFAT
974  :         SUB     AL,[BP.FATCNT]
975  :         JZ      NEWFAT
976  :         CALL    FATWRT
977  : NEWFAT:
```

```
978  :          MOV     SI,[BP.FAT]
979  :          MOV     AL,[BP.DEVNUM]
980  :          MOV     AH,[SI]                 ;Get first byte of FAT
981  :          OR      AH,0F8H                 ;Put in range
982  :          CALL    FAR PTR BIOSMAPDEV
983  :          MOV     AH,0
984  :          MOV     [SI-2],AX               ;Set device no. and reset dirty bit
985  : MAPDRV:
986  :          MOV     AL,[SI-2]               ;Get device number
987  : GETBP:
988  :          MOV     BP,[DRVTAB]             ;Just in case drive isn't valid
989  :          AND     AL,3FH                  ;Mask out dirty bit
990  :          CMP     AL,[NUMIO]
991  :          CMC
992  :          JC      RET7
993  :          PUSH    AX
994  :          MOV     AH,DPBSIZ
995  :          MUL     AH
996  :          ADD     BP,AX
997  :          POP     AX
998  : RET7:    RET
999  :
1000 : BADFAT:
1001 :          MOV     CX,DI
1002 :          ADD     DX,CX
1003 :          DEC     AL
1004 :          JNZ     NEXTFAT
1005 :          CALL    FIGFAT                    ;Reset registers
1006 :          CALL    DREAD                     ;Try first FAT once more
1007 :          JMP     SHORT NEWFAT
```

— On place le numéro de lecteur de disquette par défaut en cours dans le registre `AL` (ligne 948), on place 0 dans `AH`, on baisse le drapeau CF (ligne 949) et on appelle la pré-routine DSKCHG de IO.COM, donc BIOSDSKCHG de MSDOS.COM, pour savoir si la disquette a été changée (ligne 950).

— Si cette pré-routine renvoie une erreur de lecture sur le disque, on place le code d'erreur renvoyé dans `DI`, 2 dans `AH` et le numéro de lecteur de disquette dans `AL`, on appelle la routine FATAL1, partie de la routine HARDERR, et on essaie de lire à nouveau (lignes 951 et 926–931).

— On appelle la routine GETBP (ligne 952), partie du code commençant à la ligne 987, pour faire pointer `BP` sur le bloc des paramètres du lecteur de disquette concerné.

  — On fait pointer `BP` sur le début des blocs de paramètres des lecteurs de disquette (ligne 988).

  — On indique que la copie de la FAT en mémoire centrale a été modifiée depuis sa dernière sauvegarde sur la disquette (ligne 989).

  — On compare le numéro de lecteur de disquette par défaut et le nombre de tables de lecteurs de disquettes et on inverse l'indicateur de retenue. Si le numéro physique du lecteur de disquette est supérieur au nombre de tables de lecteurs de disquette, on termine la routine (lignes 990–992 et 998).

  — Sinon on sauvegarde sur la pile la valeur de `AX`, on place dans `AH` la taille d'un bloc de paramètres de lecteur de disquette, on multiplie le numéro de lecteur par cette taille, on l'ajoute à la valeur de `BP`, ce qui fait pointer `BP` sur le bloc de paramètres du lecteur de disquette désiré, on restaure la valeur de `AX` et on termine la routine (lignes 993–998).

— On place le numéro du lecteur de disquette dans `AL` et on fait pointer `SI` sur la copie de la FAT en mémoire centrale (lignes 953 et 954).

— Si la copie de la FAT en mémoire centrale n'a pas modifiée depuis sa dernière sauvegarde, et si ? ?, on regarde si le tampon de (lignes 955–960).

— Si la disquette a changé et que le tampon ne correspond pas à ce lecteur de disquette, on place un drapeau de tampon non valide dans la variable BUFSECNO et dans BUFDRVNO (lignes 961–965).

La variable BUFDRVNO contient le numéro de lecteur de disquette dont la copie d'un secteur se trouve dans le tampon en mémoire centrale. La variable BUFSECNO contient le numéro du secteur se trouvant dans le tampon en mémoire centrale :

```
3659 : BUFSECNO DW     0
3660 : BUFDRVNO DB     -1
```

— On place `FFh` dans DIRBUFID, pour indiquer que la copie en mémoire centrale a été modifiée depuis sa dernière sauvegarde sur la disquette, et on appelle la routine FIGFAT de préparation à l'échange entre la copie de la FAT en mémoire centrale et les instances de la FAT sur la disquette (lignes 966–968).

— On sauvegarde sur la pile le contenu de `AX`, on essaie de lire la première instance de la FAT sur la disquette et on restaure la valeur de `AX` (lignes 969–972).

— Si on n'y parvient pas, on essaie de lire la seconde instance de la FAT sur la disquette (lignes 973 et 1000–1007).

— Sinon on soustrait le nombre d'instances de la FAT sur la disquette (toujours 2 pour MS-DOS) au numéro en cours et on transfère sur la disquette l'ancienne copie de la FAT se trouvant en mémoire centrale et on passe à la nouvelle FAT (lignes 974–977).

— On fait pointer `SI` au début de la copie en mémoire centrale de la (nouvelle) FAT, on place dans `AL` le numéro de lecteur de disquette concerné, on place dans `AH` le premier octet de la copie de la FAT en mémoire centrale, que l'on ajuste, on appelle la pré-routine MAPDEV de IO.COM, donc BIOSMAPDEV de MSDOS.ASM, dont nous avons vu qu'elle ne fait rien, on place 0 dans l'octet précédant la copie de la FAT en mémoire centrale pour indiquer qu'elle n'a pas été modifiée depuis sa dernière sauvegarde, le numéro de lecteur de disquette comme premier octet de cette copie, on fait pointer `BP` sur le bloc de paramètres du lecteur de disquette et on termine la routine (lignes 978–987)

5.7.5 Écriture d'une entrée sur la copie de la FAT en mémoire centrale

La routine PACK permet, les segments des données et de code étant confondus, le numéro d'unité d'allocation pour lequel il faut écrire l'entrée dans la FAT étant spécifié par `BX`, le contenu de la nouvelle entrée étant spécifié par `DX` et `SI` pointant sur le début de la copie de la FAT en mémoire centrale, d'écrire cette nouvelle entrée sur cette copie :

```
484 : PACK:
485 :
486 : ; Inputs:
487 : ;       DS = CS
488 : ;       BX = Cluster number
489 : ;       DX = Data
490 : ;       SI = Pointer to drive FAT
491 : ; Outputs:
492 : ;       The data is stored in the FAT at the given cluster.
493 : ;       BX,DX,DI all destroyed
494 : ;       No other registers affected
495 :
```

```
496 :          MOV     DI,BX
497 :          SHR     BX,1
498 :          ADD     BX,SI
499 :          ADD     BX,DI
500 :          SHR     DI,1
501 :          MOV     DI,[BX]
502 :          JNC     ALIGNED
503 :          SHL     DX,1
504 :          SHL     DX,1
505 :          SHL     DX,1
506 :          SHL     DX,1
507 :          AND     DI,0FH
508 :          JMP     SHORT PACKIN
509 : ALIGNED:
510 :          AND     DI,0F000H
511 : PACKIN:
512 :          OR      DI,DX
513 :          MOV     [BX],DI
514 :          RET
```

— Remarquons que, puisqu'il n'existe pas de registres de 12 bits, on utilise des registres d'une contenance de deux octets pour spécifier le numéro d'unité d'allocation et la donnée.

— La donnée pour le numéro d'unité d'allocation 0 (même si en fait il n'est pas concerné) occupe les 12 premiers bits du premier mot de la FAT, celle pour le numéro d'unité d'allocation 1 occupe les 4 derniers bits de ce mot et les 8 premiers bits du mot suivant. De façon générale, pour le numéro d'unité d'allocation 2*n, la donnée correspondante occupe les 12 premiers bits du mot ? ? ; pour celle de numéro 2*n+1, elle occupe les 4 derniers bits du mot ? ? et les 8 premiers bits du mot suivant. Autrement dit, pour localiser l'emplacement voulu :

  1. Multiplier le numéro d'unité d'allocation par 1,5.
  2. La partie entière de ce produit est un décalage dans la FAT, dont le mot pointé contient l'entrée voulue.
  3. Utiliser une instruction MOV pour placer ce mot dans un registre.
  4. Pour une unité d'allocation de numéro pair, la donnée est constituée des 12 bits de poids faible de ce registre ; sinon, il s'agit des 12 bits de poids fort.

  On multiplie donc par 3 et on divise par 2 le numéro d'unité d'allocation passé en paramètre (lignes 496–497 et 499) et on l'ajoute au décalage du début de la FAT (ligne 498) pour obtenir le décalage du mot à changer.

— On place dans le registre `DI` le mot ayant ce décalage dans la copie de la FAT (ligne 501).

— Si le numéro d'unité d'allocation est impair, on effectue quatre fois une rotation à gauche (lignes 500 et 503–508) sur la donnée pour que la partie utile se retrouve sur les 12 bits de poids fort, et on ne garde que les 4 bits de poids faible de `DI` (ligne 507), correspondant à la partie de la FAT qu'il ne faut pas changer.

— Si, au contraire, le numéro d'unité d'allocation est pair, on ne garde que les 4 bits de poids fort de `DI` (lignes 500, 502 et 510).

— L'opération OR entre `DI` et `DX` permet de placer dans `DI` la nouvelle valeur sur les 12 bits concernés et de ne pas modifier les 4 bits non concernés. On remplace alors le mot de la FAT se trouvant à l'emplacement adéquat et on termine la routine (lignes 511-514).

5.7.6 Lecture d'une entrée sur la copie de la FAT en mémoire centrale

La routine UNPACK permet, les segments des données et de code étant confondus, le numéro d'unité d'allocation dont on veut lire l'entrée dans la FAT étant spécifié par `BX`, le registre `BP` pointant sur le bloc des paramètres du lecteur de disquette concerné et `SI` pointant sur le début de la copie de la FAT en mémoire centrale, de lire l'entrée correspondante de la copie de la FAT et de la placer dans `DI`. De plus le drapeau ZF est levé lorsqu'il s'agit d'une unité d'allocation libre :

```
448 : UNPACK:
449 :
450 : ; Inputs:
451 : ;       DS = CS
452 : ;       BX = Cluster number
453 : ;       BP = Base of drive parameters
454 : ;       SI = Pointer to drive FAT
455 : ; Outputs:
456 : ;       DI = Contents of FAT for given cluster
457 : ;       Zero set means DI=0 (free cluster)
458 : ; No other registers affected. Fatal error if cluster too big.
459 :
460 :         CMP     BX,[BP.MAXCLUS]
461 :         JA      HURTFAT
462 :         LEA     DI,[SI+BX]
463 :         SHR     BX,1
464 :         MOV     DI,[DI+BX]
465 :         JNC     HAVCLUS
466 :         SHR     DI,1
467 :         SHR     DI,1
468 :         SHR     DI,1
469 :         SHR     DI,1
470 :         STC
471 : HAVCLUS:
472 :         RCL     BX,1
473 :         AND     DI,0FFFH
474 :         RET
475 : HURTFAT:
476 :         PUSH    AX
477 :         MOV     AH,80H          ;Signal Bad FAT to INT 24H handler
478 :         MOV     DI,0FFFH        ;In case INT 24H returns (it shouldn't)
479 :         CALL    FATAL
480 :         POP     AX              ;Try to ignore bad FAT
481 :         RET
```

— Si le numéro d'unité d'allocation est strictement supérieur au numéro maximum pour la disquette, on sauvegarde le contenu du registre `AX` sur la pile, on place `80h` comme code d'erreur (pour 'Mauvaise FAT'), on place `FFFh` dans `DI` en cas de retour de l'appel à l'interruption, on fait appel à l'interruption de traitement d'une erreur critique, on restaure la valeur de `AX` en cas de retour, et on termine la routine (lignes 460–461 et 475–481).

— Sinon on place dans le registre `DI` le contenu du mot de la FAT contenant l'entrée correspondant à l'unité d'allocation spécifiée (lignes 462–464).

— Si le numéro d'unité d'allocation est impair, on effectue quatre rotations à gauche (lignes 463 et 465–469) sur ce mot pour que l'entrée à lire se retrouve sur les 12 bits de poids faible et on lève le drapeau de retenue CF (ligne 470).

— On lève le drapeau ZF lorsqu'il s'agit d'une unité d'allocation libre (ligne 472).

— On met à zéro les quatre bits de poids fort de `DI` pour ne conserver que l'entrée voulue et on termine la routine (lignes 473 et 474).

5.7.7 Routine de service de la fonction 27 (1Bh) d'allocation de l'adresse de la FAT

5.7.7.1 Routine principale

La routine de service de la fonction 27 de l'interruption `21h`, d'allocation de l'adresse de la FAT en mémoire centrale, est repérée par l'étiquette GETFATPT :

```
2616 : NOSUCHDRV:
2617 :          MOV     AL,-1
2618 :          RET
2619 :
2620 : GETFATPT: ;System call 27
2621 :          MOV     DL,0                    ;Use default drive
2622 :
2623 : GETFATPTDL:      ;System call 28
2624 :          PUSH    CS
2625 :          POP     DS
2626 :          MOV     AL,DL
2627 :          CALL    GETTHISDRV
2628 :          JC      NOSUCHDRV
2629 :          CALL    FATREAD
2630 :          MOV     BX,[BP.FAT]
2631 :          MOV     AL,[BP.CLUSMSK]
2632 :          INC     AL
2633 :          MOV     DX,[BP.MAXCLUS]
2634 :          DEC     DX
2635 :          MOV     CX,[BP.SECSIZ]
2636 :          LDS     SI,DWORD PTR [SPSAVE]
2637 :          MOV     [SI.BXSAVE],BX
2638 :          MOV     [SI.DXSAVE],DX
2639 :          MOV     [SI.CXSAVE],CX
2640 :          MOV     [SI.DSSAVE],CS
2641 :          RET
```

— On confond le segment des données avec le segment de code et on place 0 dans `AL` pour spécifier le lecteur de disquette par défaut (lignes 2621–2626).

— On appelle la routine GETTHISDRV, étudiée ci-dessous, pour désambiguïser ce numéro de lecteur de disquette (1 ou 2 au lieu de 0) et le placer dans la variable THISDRV (ligne 2627).

La variable THISDRV contient le numéro du lecteur de disquette en cours :

```
3700 : THISDRV DB       ?
```

— Si le numéro de lecteur de disquette est supérieur au nombre de lecteurs de disquette, on place `FFh` dans `AL` et on termine la routine (lignes 2628 et 2616–2618).

— Sinon on appelle la routine FATREAD pour placer une copie de la FAT en mémoire centrale et faire pointer `BP` sur le bloc des paramètres du lecteur de disquette (ligne 2629).

— On place le décalage de cette copie dans `BX`, le nombre de secteurs par unité d'allocation dans `AL`, le nombre d'unités d'allocation de la disquette dans `DX`, la taille d'un secteur dans `CX`, on les sauvegarde sur la pile et on termine la routine (lignes 2630–2641).

5.7.7.2 Désambiguïsation du numéro de lecteur de disquette

La routine GETTHISDRV, un numéro de lecteur de disquette étant passé en paramètre *via* `AL`, le désambiguïse s'il s'agit du numéro de lecteur de disquette par défaut (0) et place le numéro de lecteur, désambiguïsé le cas échéant, dans la variable THISDRV :

```
852 : RET6:   RET
853 :
854 : GETTHISDRV:
855 :         CMP     CS:[NUMDRV],AL
856 :         JC      RET6
857 :         DEC     AL
858 :         JNS     PHYDRV
859 :         MOV     AL,CS:[CURDRV]
860 : PHYDRV:
861 :         MOV     CS:[THISDRV],AL
862 :         RET
```

— On compare le numéro de lecteur de disquette passé en paramètre avec le nombre de lecteurs de disquette (ligne 855).

— Si l'indicateur de retenue est levé, on termine la routine (lignes 856 et 852).

— Sinon on décrémente le numéro de lecteur de disquette passé en paramètre (ligne 857). Si le drapeau de signe vaut 1, il s'agit du lecteur de disquette de numéro 0, donc du lecteur de disquette par défaut. On attribue donc à `AL` le numéro de lecteur de disquette par défaut (lignes 858–859).

— On place le numéro de lecteur de disquette, désambiguïsé le cas échéant, dans la variable THISDRV et on termine la routine (lignes 861–862).

5.8 Routines de service concernant les fichiers

5.8.1 Structure du bloc de contrôle d'un fichier

La structure d'un bloc de contrôle de fichier (FCB) est définie grâce à la structure FCBLOCK :

```
76 : ; Field definition for FCBs
77 :
78 : FCBLOCK STRUC
79 :         DB      12 DUP (?)              ;Drive code and name
80 : EXTENT  DW      ?
81 : RECSIZ  DW      ?       ;Size of record (user settable)
82 : FILSIZ  DW      ?       ;Size of file in bytes
83 : DRVBP   DW      ?       ;BP for SEARCH FIRST and SEARCH NEXT
84 : FDATE   DW      ?       ;Date of last writing
85 : FTIME   DW      ?       ;Time of last writing
86 : DEVID   DB      ?       ;Device ID number, bits 0-5
87 :                         ;bit 7=0 for file, bit 7=1 for I/O device
88 :                         ;If file, bit 6=0 if dirty
89 :                         ;If I/O device, bit 6=0 if EOF (input)
90 : FIRCLUS DW      ?       ;First cluster of file
91 : LSTCLUS DW      ?       ;Last cluster accessed
92 : CLUSPOS DW      ?       ;Position of last cluster accessed
93 :         DB      ?       ;Forces NR to offset 32
94 : NR      DB      ?       ;Next record
95 : RR      DB      3 DUP (?)               ;Random record
96 : FCBLOCK ENDS
```

5.8.2 Routines auxiliaires

5.8.2.1 Mise en majuscules d'une lettre

La routine GETLET transforme une lettre minuscule contenue à l'emplacement pointé par le registre `SI` en la majuscule correspondante, lève ZF s'il s'agit d'un délimiteur et CF s'il s'agit d'un caractère de contrôle autre que TAB :

```
3296 : GETLET:
3297 : ;Get a byte from [SI], convert it to upper case, and compare for delimiter.
3298 : ;ZF set if a delimiter, CY set if a control character (other than TAB).
3299 :          LODSB
3300 :          AND     AL,7FH
3301 :          CMP     AL,"a"
3302 :          JB      CHK
3303 :          CMP     AL,"z"
3304 :          JA      CHK
3305 :          SUB     AL,20H          ;Convert to upper case
3306 : CHK:
3307 :          CMP     AL,"."
3308 :          JZ      RET21
3309 :          CMP     AL,'"'
3310 :          JZ      RET21
3311 :          CMP     AL,"/"
3312 :          JZ      RET21
3313 :          CMP     AL,"["
3314 :          JZ      RET21
3315 :          CMP     AL,"]"
3316 :          JZ      RET21
3317 :
3318 :          IF      IBM
3319 : DELIM:
3320 :          ENDIF
3321 :          CMP     AL,":"          ;Allow ":" as separator in IBM version
3322 :          JZ      RET21
3323 :          IF      NOT IBM
3324 : DELIM:
3325 :          ENDIF
3326 :
3327 :          CMP     AL,"+"
3328 :          JZ      RET101
3329 :          CMP     AL,"="
3330 :          JZ      RET101
3331 :          CMP     AL,";"
3332 :          JZ      RET101
3333 :          CMP     AL,","
3334 :          JZ      RET101
3335 : SPCHK:
3336 :          CMP     AL,9            ;Filter out tabs too
3337 :          JZ      RET101
3338 : ;WARNING! " " MUST be the last compare
3339 :          CMP     AL," "
3340 : RET101: RET
```

ce qui n'exige pas de commentaire particulier.

5.8.2.2 Récupération du champ 'nom de fichier' d'un FCB

La routine MOVNAME, `DS:DX` pointant sur un FCB ou un FCB étendu, fait pointer `DS:DX` au début du FCB standard associé s'il s'agit d'un FCB étendu, confond le segment supplémentaire avec le segment de code, place dans la chaîne de caractères NAME1 le nom de fichier du champ adéquat du FCB, tout en majuscules, et fait pointer `BP` sur le bloc des paramètres du lecteur de disquette spécifié par le FCB :

```
801 : RET5:   RET
802 :
803 :
804 : MOVNAME:
805 :
806 : ; Inputs:
807 : ;       DS, DX point to FCB or extended FCB
808 : ; Outputs:
809 : ;       DS:DX point to normal FCB
810 : ;       ES = CS
811 : ;       If file name OK:
812 : ;       BP has base of driver parameters
813 : ;       [NAME1] has name in upper case
814 : ; All registers except DX destroyed
815 : ; Carry set if bad file name or drive
816 :
817 :         MOV     CS:WORD PTR [CREATING],0E500H   ;Not creating, not DEL *.*
818 :         MOV     AX,CS
819 :         MOV     ES,AX
820 :         MOV     DI,OFFSET DOSGROUP:NAME1
821 :         MOV     SI,DX
822 :         LODSB
823 :         MOV     CS:[EXTFCB],AL                  ;Set flag if extended FCB in use
824 :         MOV     AH,0                            ;Set default attributes
825 :         CMP     AL,-1                           ;Is it an extended FCB?
826 :         JNZ     HAVATTRB
827 :         ADD     DX,7                            ;Adjust to point to normal FCB
828 :         ADD     SI,6                            ;Point to drive select byte
829 :         MOV     AH,[SI-1]                       ;Get attribute byte
830 :         LODSB                                   ;Get drive select byte
831 : HAVATTRB:
832 :         MOV     CS:[ATTRIB],AH                  ;Save attributes
833 :         CALL    GETTHISDRV
834 : LODNAME:
835 : ; This entry point copies a file name from DS,SI
836 : ; to ES,DI converting to upper case.
837 :         CMP     BYTE PTR [SI]," "               ;Don't allow blank as first letter
838 :         STC                                     ;In case of error
839 :         JZ      RET5
840 :         MOV     CX,11
841 : MOVCHK:
842 :         CALL    GETLET
843 :         JB      RET5
844 :         JNZ     STOLET                          ;Is it a delimiter?
845 :         CMP     AL," "                          ;This is the only delimiter allowed
846 :         STC                                     ;In case of error
847 :         JNZ     RET5
848 : STOLET:
849 :         STOSB
850 :         LOOP    MOVCHK
851 :         CLC                                     ;Got through whole name - no error
852 : RET6:   RET
```

— On place `E5h` dans la variable CREATING pour indiquer que le fichier n'est pas créé et `00h` dans la variable DELALL pour indiquer qu'on ne veut pas tout (ligne 817).

La variable CREATING, de capacité un mot, est constituée de deux octets pouvant être appelés séparément : CREATING et DELALL.

```
3688 : ;WARNING - the following two items are accessed as a word
3689 : CREATING DB     ?
3690 : DELALL   DB     ?
```

— On confond le segment supplémentaire avec le segment de code (lignes 818 et 819), on fait pointer `DI` sur la variable de chaîne de caractères NAME1 (ligne 820) et `SI` sur le FCB (ligne 821).

La variable NAME1 est un tampon de chaîne de 11 caractères, pouvant contenir un nom de fichier :

```
3683 : NAME1   DB      11 DUP (?)              ;File name buffer
```

— On charge le premier octet du FCB (ligne 822) et on le place dans la variable EXTFCB (ligne 823) pour voir s'il s'agit d'un FCB étendu, c'est-à-dire s'il commence par `FFh` (ligne 825). Si c'est le cas, on ajoute 7 à `DX` pour que ce registre pointe au début du FCB standard associé (lignes 826 et 827), on ajoute 6 à `SI` pour la même raison (ligne 828), on place l'octet d'attribut dans `AH` et le numéro de lecteur de disquette spécifié par le FCB étendu dans `AL` (ligne 830).

La variable EXTFCB contient le premier octet d'un FCB, celui qui permet de déterminer s'il s'agit d'un FCB étendu ou non :

```
3687 : EXTFCB  DB      ?
```

— On place l'attribut adéquat dans `AH` : `00h` s'il ne s'agit pas d'un FCB étendu (ligne 824) et celui spécifié par le FCB étendu sinon (ligne 829). On le sauvegarde dans la variable ATTRIB (ligne 832).

La variable ATTRIB contient l'attribut d'un FCB :

```
3684 : ATTRIB  DB      ?
```

— Si le numéro de lecteur de disquette spécifié par le FCB n'est pas nul, on termine la routine (lignes 833 et 854–856). Sinon on lui attribue le numéro de lecteur de disquette par défaut (lignes 857–859). Dans les deux cas, on reporte ce numéro dans la variable THISDRV (ligne 861).

— On copie le nom de fichier spécifiié par le FCB, y compris son extension, dans la chaîne de caractères NAME1, tout en le convertissant en majuscules (lignes 835–852).

5.8.2.3 Recherche des noms de fichiers spéciaux

Les **fichiers spéciaux** sont ceux désignant les périphériques d'entrées-sorties, comme 'CON' pour la console.

La routine DEVNAME vérifie si le nom de fichier placé dans NAME1 est un nom de fichier spécial et lève CF si c'est le cas :

```
516 : DEVNAME:
517 :         MOV     SI,OFFSET DOSGROUP:IONAME  ;List of I/O devices with file names
518 :         MOV     BH,NUMDEV                  ;BH = number of device names
519 : LOOKIO:
520 :         MOV     DI,OFFSET DOSGROUP:NAME1
521 :         MOV     CX,4                       ;All devices are 4 letters
522 :         REPE    CMPSB                      ;Check for name in list
523 :         JZ      IOCHK                      ;If first 3 letters OK, check for the rest
524 :         ADD     SI,CX                      ;Point to next device name
525 :         DEC     BH
526 :         JNZ     LOOKIO
527 : CRET:
528 :         STC                                ;Not found
529 :         RET
530 :
531 : IOCHK:
532 :         IF      IBM
533 :         CMP     BH,NUMDEV                  ;Is it the first device?
534 :         JNZ     NOTCOM1
535 :         MOV     BH,2                       ;Make it the same as AUX
536 : NOTCOM1:
537 :         ENDIF
538 :         NEG     BH
539 :         MOV     CX,2                       ;Check rest of name but not extension
540 :         MOV     AX,2020H
541 :         REPE    SCASW                      ;Make sure rest of name is blanks
542 :         JNZ     CRET
543 : RET1:   RET                                ;Zero set so CREATE works
```

— On pointe `SI` au début de la liste des noms de fichiers spéciaux.
Le tableau IONAME contient la liste des noms de fichiers spéciaux :

```
3638 : IONAME:
3639 :         IF      NOT IBM
3640 :         DB      "PRN ","LST ","NUL ","AUX ","CON "
3641 :         ENDIF
3642 :         IF      IBM
3643 :         DB      "COM1","PRN ","LPT1","NUL ","AUX ","CON "
3644 :         ENDIF
```

— On place le nombre de fichiers spéciaux dans `BH`.
La constante NUMDEV contient le nombre de fichiers spéciaux :

```
47 :         IF      IBM
[...]
52 : NUMDEV  EQU     6              ;Include "COM1" as I/O device name
[...]
54 :         ELSE
[...]
59 : NUMDEV  EQU     5              ;Number of I/O device names
[...]
61 :         ENDIF
```

— On place 4 dans `CX` car tous les noms de fichiers spéciaux ont été choisis avec 4 caractères, comme on peut le vérifier sur la liste ci-dessus.

— On vérifie si le nom de fichier est un nom de fichier spécial (lignes 519–526 et 531–542). On lève CF si c'est le cas (ligne 528).

5.8.2.4 Recherche dans le répertoire d'une entrée spécifiée par un FCB

Il semblerait logique que la première opération soit de placer une entrée dans le répertoire. Cependant, pour éviter les doublons, lorsqu'on veut placer une nouvelle entrée dans le répertoire, on doit regarder si le nom s'y trouve déjà. La recherche est donc plus primitive que la création d'une entrée.

La routine GETNAME cherche une entrée dans le répertoire, le nom du fichier étant spécifié par le champ adéquat d'un FCB, situé à l'adresse `DS:DX`. On peut utiliser ' ?' dans le nom, représentant n'importe quel caractère permis pour un nom de fichier.

Le drapeau CF est levé s'il n'existe pas de nom de fichier concordant dans le répertoire. Sinon le drapeau de zéro ZF est levé, `BP` pointe sur le bloc des paramètres du lecteur de disquette concerné, les segments de données et supplémentaire sont confondus avec le segment de code, `SI` pointe sur la première unité d'allocation du fichier, DIRBUF pointe sur l'entrée de répertoire correspondante et la variable chaîne de caractères NAME1 contient le nom du fichier, tout en majuscules :

```
554 : RET2:   RET
555 :
556 :
557 : GETNAME:
558 :
559 : ; Inputs:
560 : ;       DS,DX point to FCB
561 : ; Function:
562 : ;       Find file name in disk directory. First byte is
563 : ;       drive number (0=current disk). "?" matches any
564 : ;       character.
565 : ; Outputs:
566 : ;       Carry set if file not found
567 : ;       ELSE
568 : ;       Zero set if attributes match (always except when creating)
569 : ;       BP = Base of drive parameters
570 : ;       DS = CS
571 : ;       ES = CS
572 : ;       BX = Pointer into directory buffer
573 : ;       SI = Pointer to First Cluster field in directory entry
574 : ;       [DIRBUF] has directory record with match
575 : ;       [NAME1] has file name
576 : ; All other registers destroyed.
577 :
578 :         CALL    MOVNAME
579 :         JC      RET2                    ;Bad file name?
580 : FINDNAME:
581 :         MOV     AX,CS
582 :         MOV     DS,AX
583 :         CALL    DEVNAME
584 :         JNC     RET2
585 :         CALL    STARTSRCH
586 : CONTSRCH:
587 :         CALL    GETENTRY
588 :         JC      RET2
589 : SRCH:
590 :         MOV     AH,BYTE PTR [BX]
591 :         OR      AH,AH                    ;End of directory?
592 :         JZ      FREE
593 :         CMP     AH,[DELALL]              ;Free entry?
```

```
594 :          JZ      FREE
595 :          MOV     SI,BX
596 :          MOV     DI,OFFSET DOSGROUP:NAME1
597 :          MOV     CX,11
598 : WILDCRD:
599 :          REPE    CMPSB
600 :          JZ      FOUND
601 :          CMP     BYTE PTR [DI-1],"?"
602 :          JZ      WILDCRD
603 : NEXTENT:
604 :          CALL    NEXTENTRY
605 :          JNC     SRCH
606 : RET3:    RET
607 :
608 : FREE:
609 :          CMP     [ENTFREE],-1            ;Found a free entry before?
610 :          JNZ     TSTALL                  ;If so, ignore this one
611 :          MOV     CX,[LASTENT]
612 :          MOV     [ENTFREE],CX
613 : TSTALL:
614 :          CMP     AH,[DELALL]             ;At end of directory?
615 :          JZ      NEXTENT                 ;No - continue search
616 :          STC                             ;Report not found
617 :          RET
```

— On appelle la routine MOVNAME pour récupérer le nom de fichier du champ ‘nom de fichier’ du FCB (ligne 578).

Plus précisément, pour pointer au début du FCB standard associé s'il s'agit d'un FCB étendu, pour confondre le segment supplémentaire avec le segment de code, pour que la variable chaîne de caractères NAME1 contienne le nom du fichier spécifié par le FCB, tout en majuscules, et pour faire pointer `BP` sur le bloc des paramètres du lecteur de disquette spécifié par le FCB.

— Si le nom de fichier transmis n'est pas valide, on termine la routine, le drapeau CF étant levé (ligne 579).

— On confond le segment des données avec le segment de code (lignes 581 et 582).

— S'il s'agit du nom d'un fichier spécial, on termine la routine, le drapeau CF étant levé (lignes 583 et 584).

— On appelle la routine STARTSRCH pour placer une copie de la FAT en mémoire centrale si la disquette a été changée (ligne 585).

La routine STARTSRCH :

```
932 : STARTSRCH:
933 :          MOV     AX,-1
934 :          MOV     [LASTENT],AX
935 :          MOV     [ENTFREE],AX
936 : FATREAD:
```

place `FFh` dans la variable LASTENT, pour indiquer qu'on n'a pas encore lu d'entrée, et dans la variable ENTFREE, puis on transfère une copie de la FAT depuis la disquette vers la mémoire centrale si nécessaire.

La variable ENTFREE donne un numéro d'entrée libre :

```
3716 : ENTFREE DW      ?
```

— On appelle la routine GETENTRY pour obtenir l'entrée (ligne 587). S'il n'en existe pas, on termine la routine en levant le drapeau CF (ligne 588).

Le registre `AL` contient alors le numéro (relatif) du secteur du répertoire contenant l'entrée, `BX` pointe sur l'entrée dans la copie en mémoire du secteur du répertoire, repérée

par DIRBUF, `DX` pointe sur le premier octet suivant DIRBUF et LASTENT spécifie le numéro de l'entrée dans le répertoire.

— On place dans `AH` le premier octet de l'entrée (ligne 590). Si on est arrivé à la fin du répertoire (lignes 591 et 592), on regarde si on a trouvé une entrée libre auparavant (lignes 609). Si c'est le cas, on prend celle-ci comme nouvelle entrée (lignes 610–612). Si on n'est pas arrivé à la fin du répertoire, on continue la recherche (lignes 614 et 615). Sinon on termine la routine en levant le drapeau CF (ligne 616).

— On fait de même s'il y a une entrée libre (lignes 593 et 594).

— On fait pointer `SI` sur l'entrée dans la copie du secteur du répertoire, repérée par DIRBUF, (ligne 595) et `DI` sur le nom de fichier, tout en majuscules (ligne 596). On compare les 11 caractères des deux noms de fichiers (lignes 597–599). Si deux caractères de même position ne sont pas égaux, c'est-à-dire si le drapeau ZF est baissé, mais que le caractère du nom de fichier en majuscule est ' ?', on considère qu'ils sont égaux (lignes 601 et 602).

— Si les deux chaînes de caractères ne concordent pas, on passe à l'entrée de répertoire suivante (lignes 604 et 605), s'il en existe une. Sinon on termine la routine, le drapeau CF étant levé (lignes 605 et 606).

— Si les deux chaînes de caractères concordent, modulo les caractères ' ?', mais que l'attribut interdit de remonter l'information (car l'attribut du fichier spécifie qu'il est soit caché, soit système), on termine la routine, le drapeau CF étant levé (lignes 600 et 621–626).

— S'il s'agit d'une demande de création de fichier, on continue la recherche (lignes 627 et 628).

— Sinon on a trouvé (ligne 629). Alors le drapeau de zéro est levé, `BP` pointe sur le bloc des paramètres du lecteur de disquette concerné, les segments de données et supplémentaire sont confondus avec le segment de code, `SI` pointe sur la première unité d'allocation du fichier, DIRBUF pointe sur l'entrée de répertoire correspondante et la variable chaîne de caractères NAME1 contient le nom du fichier, tout en majuscules.

5.8.2.5 Deuxième façon de rechercher une entrée spécifiée par un FCB dans le répertoire

La routine GETFILE cherche une entrée dans le répertoire, un FCB étant fourni, d'adresse `DS:DX`. On peut utiliser le caractère ambigu ' ?' dans le nom, représentant n'importe quel caractère permis pour un nom de fichier.

Le drapeau CF est levé s'il n'existe pas de fichier de nom concordant dans le répertoire. Sinon le drapeau de zéro est baissé, `BP` pointe sur le bloc des paramètres du lecteur de disquette concerné, les segments de données et supplémentaire sont confondus avec le segment de code, `SI` pointe sur la première unité d'allocation du fichier, DIRBUF pointe sur l'entrée de répertoire correspondante, la chaîne de caractères NAME1 contient le nom du fichier, tout en majuscules, et `ES:DI` pointe sur le FCB :

```
543 : RET1:   RET                     ;Zero set so CREATE works
544 :
545 : GETFILE:
546 : ; Same as GETNAME except ES:DI points to FCB on successful return
547 :         CALL    MOVNAME
548 :         JC      RET1
549 :         PUSH    DX
550 :         PUSH    DS
551 :         CALL    FINDNAME
552 :         POP     ES
553 :         POP     DI
554 : RET2:   RET
```

— On appelle la routine MOVNAME (ligne 547) pour récupérer le nom du fichier à partir du champ 'nom de fichier' du FCB, plus précisément pour pointer au début du FCB standard associé s'il s'agit d'un FCB étendu, pour confondre le segment supplémentaire avec le segment de code, pour que la chaîne de caractères NAME1 contienne le nom de fichier spécifié par le FCB, tout en majuscules, et pour faire pointer `BP` sur le bloc des paramètres du lecteur de disquette spécifié par le FCB.

— S'il ne s'agit pas d'un nom de fichier valide ou d'un lecteur de disquette valide, autrement dit si le drapeau CF est levé, on termine la routine (ligne 548).

— Sinon on sauvegarde les registres `DX` et `DS` sur la pile, on fait appel à l'essentiel de la routine précédente de recherche d'une entrée dans le répertoire spécifiée par un FCB et on restaure les contenus sauvegardés sur la pile de `DX` et `DS` mais dans `DI` et `ES` respectivement (lignes 549–554).

5.8.3 Routine de service de la fonction 15 (0Fh) d'ouverture d'un fichier spécifié par un FCB

5.8.3.1 Routine principale

La routine de service de la fonction 15 de l'interruption `21h`, d'ouverture d'un fichier spécifié par un FCB, est repérée par l'étiquette OPEN :

```
865 : OPEN:   ;System call 15
866 :         CALL    GETFILE
867 : DOOPEN:
868 : ; Enter here to perform OPEN on file already found
869 : ; in directory. DS=CS, BX points to directory
870 : ; entry in DIRBUF, SI points to First Cluster field, and
871 : ; ES:DI point to the FCB to be opened. This entry point
872 : ; is used by CREATE.
873 :         JC      ERRET
874 :         OR      BH,BH           ;Check if file is I/O device
875 :         JS      OPENDEV         ;Special handler if so
876 :         MOV     AL,[THISDRV]
877 :         INC     AX
878 :         STOSB
879 :         XOR     AX,AX
880 :         IF      ZEROEXT
881 :         ADD     DI,11
882 :         STOSW                   ;Zero low byte of extent field if IBM only
883 :         ENDIF
884 :         IF      NOT ZEROEXT
885 :         ADD     DI,12           ;Point to high half of CURRENT BLOCK field
886 :         STOSB                   ;Set it to zero (CP/M programs set low byte)
887 :         ENDIF
888 :         MOV     AL,128          ;Default record size
889 :         STOSW                   ;Set record size
890 :         LODSW                   ;Get starting cluster
891 :         MOV     DX,AX           ;Save it for the moment
892 :         MOVSW                   ;Transfer size to FCB
893 :         MOVSW
894 :         MOV     AX,[SI-8]       ;Get date
895 :         STOSW                   ;Save date in FCB
896 :         MOV     AX,[SI-10]      ;Get time
897 :         STOSW                   ;Save it in FCB
898 :         MOV     AL,[BP.DEVNUM]
899 :         OR      AL,40H
900 :         STOSB
901 :         MOV     AX,DX           ;Restore starting cluster
902 :         STOSW                   ; first cluster
903 :         STOSW                   ; last cluster accessed
904 :         XOR     AX,AX
905 :         STOSW                   ; position of last cluster
906 :         RET
907 :
908 :
909 : OPENDEV:
910 :         ADD     DI,13           ;point to 2nd half of extent field
911 :         XOR     AX,AX
912 :         STOSB                   ;Set it to zero
913 :         MOV     AL,128
914 :         STOSW                   ;Set record size to 128
915 :         XOR     AX,AX
916 :         STOSW
917 :         STOSW                   ;Set current size to zero
918 :         CALL    DATE16
```

```
919 :         STOSW                   ;Date is todays
920 :         XCHG    AX,DX
921 :         STOSW                   ;Use current time
922 :         MOV     AL,BH           ;Get device number
923 :         STOSB
924 :         XOR     AL,AL           ;No error
925 :         RET
```

— `DS:DX` pointant sur un FCB non ouvert, on appelle la routine GETFILE pour rechercher une entrée concordante dans le répertoire (ligne 866).

Le drapeau CF est levé s'il n'existe pas d'entrée concordante dans le répertoire. Sinon le drapeau de zéro est levé, `BP` pointe sur le bloc des paramètres du lecteur de disquette concerné, les segments de données et supplémentaire sont confondus avec le segment de code, `SI` pointe sur la première unité d'allocation du fichier, DIRBUF pointe sur l'entrée de répertoire correspondante, la chaîne de caractères NAME1 contient le nom du fichier, tout en majuscules, et `ES:DI` pointe sur le FCB.

— S'il n'existe pas d'entrée correspondante, on place `FFh` dans `AL` et on termine la routine (lignes 873 et 799–801) :

```
799 : ERRET:
800 :         MOV     AL,-1
801 : RET5:   RET
```

— `BX` pointe sur l'entrée de la copie du répertoire se en mémoire centrale, en DIRBUF. Si le premier bit de `BH` est égal à 1, c'est qu'il s'agit d'un fichier de périphérique. On appelle alors la sous-routine OPENDEV : on fait pointer `DI` sur le champ 'taille d'un enregistrement' du FCB, on y place `00h` comme premier octet et 128 comme second octet, pour initialiser la taille d'un enregistrement à 128 octets ; on place ensuite deux mots de valeur nulle, pour indiquer qu'il s'agit d'un fichier de taille nulle ; on appelle la routine DATE16, étudiée ci-dessous, pour récupérer la date et l'heure en cours, que l'on place dans les deux champs suivants du FCB, puis on renseigne le numéro de périphérique ; on spécifie qu'il n'y a pas d'erreur et on termine la routine (lignes 874–875 et 909–925).

— S'il ne s'agit pas d'un fichier de périphérique, on place le numéro de fichier par défaut dans `AL` (0 pour A et 1 pour B), on l'incrémente (pour obtenir 1 pour A, 2 pour B) et on le place comme premier octet, et premier champ, du FCB (lignes 874–878).

— On renseigne le champ 'numéro de bloc en cours' du FCB : on place zéro comme mot dans le cas de PC-DOS et comme octet dans le cas de MS-DOS (lignes 879–887).

— On renseigne le champ 'taille d'un enregistrement' du FCB avec la valeur par défaut, à savoir 128 (soit `80h`) : il suffit de placer 128 comme mot suivant (ligne 888).

— On renseigne le champ 'taille du fichier' grâce aux informations obtenues à partir de l'entrée de répertoire (lignes 890–893).

— On renseigne le champ 'date de dernier accès au fichier' grâce aux informations obtenues à partir de l'entrée de répertoire (lignes 894–895).

— On renseigne le champ 'heure de dernier accès au fichier' grâce aux informations obtenues à partir de l'entrée de répertoire (lignes 896–897).

— On renseigne le champ 'numéro de lecteur de disquette' grâce aux informations obtenues à partir du bloc des paramètres du lecteur de disquette (lignes 898–900).

— On renseigne le champ 'numéro d'unité d'allocation' grâce à la valeur sauvegardée dans `DX` (lignes 901–903).

— On place 0 dans le FCB comme position dans la dernière unité d'allocation et on termine la routine (lignes 904–906).

5.8.3.2 Récupération de la date

On récupère la date grâce à la sous-routine DATE16 :

```
3412 : DATE16:
3413 :          PUSH    CX
3414 :          CALL    READTIME
3415 :          SHL     CL,1                    ;Minutes to left part of byte
3416 :          SHL     CL,1
3417 :          SHL     CX,1                    ;Push hours and minutes to left end
3418 :          SHL     CX,1
3419 :          SHL     CX,1
3420 :          SHR     DH,1                    ;Count every two seconds
3421 :          OR      CL,DH                   ;Combine seconds with hours and minutes
3422 :          MOV     DX,CX
3423 :          POP     CX
3424 :          MOV     AX,WORD PTR [MONTH]     ;Fetch month and year
3425 :          SHL     AL,1                    ;Push month to left to make room for day
3426 :          SHL     AL,1
3427 :          SHL     AL,1
3428 :          SHL     AL,1
3429 :          SHL     AX,1
3430 :          OR      AL,[DAY]
3431 : RET22:   RET
```

— On sauvegarde le contenu de `CX` sur la pile, car celui-ci va être détruit par l'appel à la routine de la ligne suivante (ligne 3413).

La routine READTIME place dans la variable DAYCNT le nombre de jours écoulé depuis le 1er janvier 1980, dans YEAR l'année, dans MONTH le mois, dans DAY le quantième, dans WEEKDAY le jour de la semaine, la seconde et le centième de seconde dans `DX` et enfin l'heure et la minute dans `CX`.

— On appelle la routine READTIME pour obtenir la date et l'heure (ligne 3414).

— On décale le registre `CL` deux fois à gauche pour placer la minute dans la partie gauche de l'octet (lignes 3415–3416).

— On décale le registre `CX` 4 fois à gauche pour obtenir l'heure et la minute sur l'extrémité gauche (lignes 3417–3419).

— On décale de registre `DH` à droite pour obtenir le nombre de 2 secondes (ligne 3420).

— On effectue une disjonction bit à bit des contenus de `CL` et de `DH` pour obtenir dans `DX` l'heure, la minute et le nombre de deux secondes (lignes 3421–3422).

— On restaure la valeur de `CX` (ligne 3423).

— On place le numéro du mois dans `AX`, on décale 4 fois à gauche le contenu de `AL` pour avoir d ela place pour le jour du mois, on effectue une disjonction avec la variable DAY et on termine la routine (lignes 3424–3431).

5.8.4 Routine de service de la fonction 22 (16h) de création d'un fichier spécifié par un FCB

La routine de service de la fonction `16h` de l'interruption `21h`, de création d'un fichier spécifié par un FCB, est repérée par l'étiquette CREATE :

```
1123 : CREATE: ;System call 22
1124 :         CALL    MOVNAME
1125 :         JC      ERRET3
1126 :         MOV     DI,OFFSET DOSGROUP:NAME1
1127 :         MOV     CX,11
1128 :         MOV     AL,"?"
1129 :         REPNE   SCASB
1130 :         JZ      ERRET3
1131 :         MOV     CS:BYTE PTR [CREATING],-1
1132 :         PUSH    DX
1133 :         PUSH    DS
1134 :         CALL    FINDNAME
1135 :         JNC     EXISTENT
1136 :         MOV     AX,[ENTFREE]    ;First free entry found in FINDNAME
1137 :         CMP     AX,-1
1138 :         JZ      ERRPOP
1139 :         CALL    GETENT          ;Point at that free entry
1140 :         JMP     SHORT FREESPOT
1141 : ERRPOP:
1142 :         POP     DS
1143 :         POP     DX
1144 : ERRET3:
1145 :         MOV     AL,-1
1146 :         RET
1147 :
1148 : EXISTENT:
1149 :         JNZ     ERRPOP          ;Error if attributes don't match
1150 :         OR      BH,BH           ;Check if file is I/O device
1151 :         JS      OPENJMP         ;If so, no action
1152 :         MOV     CX,[SI]         ;Get pointer to clusters
1153 :         JCXZ    FREESPOT
1154 :         CMP     CX,[BP.MAXCLUS]
1155 :         JA      FREESPOT
1156 :         PUSH    BX
1157 :         MOV     BX,CX
1158 :         MOV     SI,[BP.FAT]
1159 :         CALL    RELEASE         ;Free any data already allocated
1160 :         CALL    FATWRT
1161 :         POP     BX
1162 : FREESPOT:
1163 :         MOV     DI,BX
1164 :         MOV     SI,OFFSET DOSGROUP:NAME1
1165 :         MOV     CX,5
1166 :         MOVSB
1167 :         REP     MOVSW
1168 :         MOV     AL,[ATTRIB]
1169 :         STOSB
1170 :         XOR     AX,AX
1171 :         MOV     CL,5
1172 :         REP     STOSW
1173 :         CALL    DATE16
1174 :         XCHG    AX,DX
1175 :         STOSW
1176 :         XCHG    AX,DX
1177 :         STOSW
1178 :         XOR     AX,AX
```

```
1179 :          PUSH    DI
1180 :          MOV     CL,6
1181 : SMALLENT:
1182 :          REP     STOSB
1183 :          PUSH    BX
1184 :          CALL    DIRWRITE
1185 :          POP     BX
1186 :          POP     SI
1187 : OPENJMP:
1188 :          CLC                     ;Clear carry so OPEN won't fail
1189 :          POP     ES
1190 :          POP     DI
1191 :          JMP     DOOPEN
```

— `DS:DX` pointant sur un FCB non ouvert, on appelle la routine MOVNAME (ligne 1124) pour récupérer le nom du fichier spécifié par le FCB, plus précisément pour pointer au début du FCB standard associé s'il s'agit d'un FCB étendu, pour confondre le segment supplémentaire avec le segment de code, pour que la variable chaîne de caractères NAME1 contienne le nom de fichier, tout en majuscules, spécifié par le FCB et pour faire pointer `BP` sur le bloc des paramètres du lecteur de disquette spécifié par le FCB.

— Si le nom de fichier récupéré n'est pas valide, on place `FFh` dans `AL` et on termine la routine, le drapeau CF étant levé (lignes 1125 et 1144–1146).

— Si le caractère d'ambiguïté ' ?' se trouve dans le nom de fichier tout en majuscules, on place `FFh` dans `AL` et on termine la routine, le drapeau CF étant levé (lignes 1126–1130 et 1144–1146). En effet, ce caractère peut servir lors d'une ouverture mais pas d'une création.

— On place `FFh` dans la variable CREATING (ligne 1131).

— On sauvegarde le contenu des registres `DX` et `DS` sur la pile et on fait appel à la routine FINDNAME, partie de la routine GETNAME, pour voir si le nom de fichier apparaît dans le répertoire (lignes 1132–1134).

  Plus précisément, le drapeau CF est levé s'il n'existe pas d'entrée du répertoire ayant ce nom de fichier ; sinon le drapeau de zéro est levé, `BP` pointe sur le bloc des paramètres du lecteur de disquette concerné, les segments de données et supplémentaire sont confondus avec le segment de code, `SI` pointe sur la première unité d'allocation du fichier, DIRBUF pointe sur l'entrée de répertoire correspondante et la chaîne de caractères NAME1 contient le nom du fichier, tout en majuscules.

— Si ce nom de fichier n'apparaît pas dans le répertoire, on recherche la première entrée libre du répertoire (lignes 1135 et 1136). S'il n'en existe pas, autrement dit si toutes les entrées du répertoire sont occupées, on restaure les valeurs de `DS` et de `DX`, on place `FFh` dans `AL` et on termine la routine (lignes 1137–1138 et 1141–1146). Sinon `BX` pointe sur cette entrée libre (lignes 1139 et 1140).

  La routine GETENT est une partie de la routine GETENTRY déjà étudiée. Après exécution de cette routine, `AL` contient le numéro de secteur du répertoire sur lequel se trouve l'entrée, `BX` pointe sur l'entrée dans la zone mémoire repérée par DIRBUF, `DX` pointe sur le premier octet suivant DIRBUF et LASTENT donne le numéro de cette entrée de répertoire.

— Si, par contre, le nom de fichier apparaît déjà dans le répertoire :
  — Si les attributs ne correspondent pas, on restaure les valeurs de `DS` et de `DX`, on place `FFh` dans `AL` et on termine la routine (lignes 1135, 1148–1149 et 1141–1146).
  — S'il s'agit d'un nom de périphérique, on baisse le drapeau CF, on restaure les valeurs de `ES` et `DI` et on va à la partie DOOPEN de la routine OPEN déjà étudiée pour ouvrir le fichier (lignes 1150–1151 et 1187–1191).

  - On place le numéro de la première unité d'allocation dans `CX` (ligne 1152).
  - S'il ne s'agit pas de la première unité d'allocation, on vérifie que le numéro d'unité d'allocation est inférieur au nombre maximum d'unités d'allocation pour une disquette de ce type. Si ce n'est pas le cas, on sauvegarde le contenu de `BX` sur la pile et on libère les données déjà allouées (lignes 1153–1161).
- On renseigne les champs de l'entrée de répertoire dans la copie du secteur de répertoire situé en mémoire centrale, pointée par DIRBUF. Pour cela, on commence par faire pointer `DI` sur le premier octet suivant DIRBUF, `SI` sur le nom du fichier, tout en majuscules, et on recopie les cinq premiers mots, puis un octet, c'est-à-dire les 11 caractères du nom du fichier, y compris son extension (lignes 1162–1167). On renseigne ensuite l'octet d'attribut (lignes 1168 et 1169). On place cinq mots nuls dans le champ réservé pour une utilisation ultérieure (lignes 1170–1172). On renseigne les deux champs suivants avec l'heure et la date du moment (lignes 1173–1177). On place zéro comme champ 'numéro d'unité d'allocation du début du fichier' et 'taille du fichier', pour indiquer que le fichier est vide (lignes 1178–1182).
- On sauvegarde sur la pile les contenus des registres `DI` et `BX`, on enregistre sur la disquette la copie du secteur de répertoire située en mémoire centrale en appelant la routine DIRWRITE et on restaure les contenus des registres `BX` et `SI` (ligne 1179 et 1183–1186).
- On baisse le drapeau CF, on restaure les valeurs de `ES` et `DI` et on va à la partie DOOPEN de la routine OPEN déjà étudiée pour ouvrir le fichier, c'est-à-dire renseigner son FCB (lignes 1187–1191).

5.8.5 Routine de service de la fonction 16 (10h) de fermeture d'un fichier spécifié par un FCB

La routine de service de la fonction `10h` de l'interruption `21h`, de fermeture d'un fichier spécifié par un FCB, est repérée par l'étiquette CLOSE :

```
1009 : OKRET1:
1010 :          MOV     AL,0
1011 :          RET
1012 :
1013 : CLOSE:  ;System call 16
1014 :          MOV     DI,DX
1015 :          CMP     BYTE PTR [DI],-1                ;Check for extended FCB
1016 :          JNZ     NORMFCB3
1017 :          ADD     DI,7
1018 : NORMFCB3:
1019 :          TEST    BYTE PTR [DI.DEVID],0C0H        ;Allow only dirty files
1020 :          JNZ     OKRET1                          ;can't close if I/O device, or not writen
1021 :          MOV     AL,[DI]                         ;Get physical unit number
1022 :          DEC     AL                              ;Make zero = drive A
1023 :          MOV     AH,1                            ;Look for dirty buffer
1024 :          CMP     AX,CS:WORD PTR [BUFDRVNO]
1025 :          JNZ     FNDDIR
1026 : ;Write back dirty buffer if on same drive
1027 :          PUSH    DX
1028 :          PUSH    DS
1029 :          PUSH    CS
1030 :          POP     DS
1031 :          MOV     BYTE PTR [DIRTYBUF],0
1032 :          MOV     BX,[BUFFER]
1033 :          MOV     CX,1
1034 :          MOV     DX,[BUFSECNO]
1035 :          MOV     BP,[BUFDRVBP]
1036 :          CALL    DWRITE
1037 :          POP     DS
1038 :          POP     DX
1039 : FNDDIR:
1040 :          CALL    GETFILE
1041 : BADCLOSEJ:
1042 :          JC      BADCLOSE
1043 :          MOV     CX,ES:[DI.FIRCLUS]
1044 :          MOV     [SI],CX
1045 :          MOV     DX,ES:WORD PTR [DI.FILSIZ]
1046 :          MOV     [SI+2],DX
1047 :          MOV     DX,ES:WORD PTR [DI.FILSIZ+2]
1048 :          MOV     [SI+4],DX
1049 :          MOV     DX,ES:[DI.FDATE]
1050 :          MOV     [SI-2],DX
1051 :          MOV     DX,ES:[DI.FTIME]
1052 :          MOV     [SI-4],DX
1053 :          CALL    DIRWRITE
1054 :
1055 : CHKFATWRT:
```

— `DS:DX` pointe sur un FCB ouvert. S'il s'agit d'un FCB étendu, on se place au début du FCB standard correspondant (lignes 1013–1018).

— S'il s'agit d'un fichier de périphérique (bit 7 du champ DEVID égal à 1) n'ayant pas envoyé de signal de fin de fichier (bit 6 du champ DEVID égal à 1), c'est-à-dire si le champ DEVID est égal à `C0h`, alors on ne doit pas le fermer. On place donc `00h` dans `AL` et on termine la routine (lignes 1019–1020 et 1009-1011).

Rappelons que DEVID est le champ de la structure FCBLOCK spécifiant le numéro d'identification du périphérique :

```
86 : DEVID   DB      ?       ;Device ID number, bits 0-5
87 :                         ;bit 7=0 for file, bit 7=1 for I/O device
88 :                         ;If file, bit 6=0 if dirty
89 :                         ;If I/O device, bit 6=0 if EOF (input)
```

— Sinon on place le numéro de lecteur de disquette dans `AL`, que l'on décrémente (puisque les conventions sont différentes : 1 pour A dans un cas, 0 pour A dans l'autre), et 1 dans `AH`. Si le contenu de `AX` est alors égal à celui de BUFDRVNO, on sauvegarde sur la pile les contenus des registres `DX`, `DS` et `CS`, on place 0 dans DIRTYBUF pour indiquer qu'on n'a pas modifié le contenu de la copie du secteur du répertoire en mémoire vive depuis sa dernière sauvegarde, on fait pointer `BX` sur BUFFER, on place 1 dans `CX` pour indiquer qu'on ne veut écrire qu'un seul secteur, le contenu de BUFSECNO dans `DX`, le contenu de BUFDRVBP dans `DX`, on appelle la routine DWRITE d'écriture d'un secteur sur la disquette, on restaure la valeur de `DS` et on place l'ancienne valeur de `CS` dans `DX` (lignes 1021–1038).

— On appelle la routine GETFILE pour rechercher l'entrée corrrespondante dans le répertoire.

Le drapeau CF est levé si le nom du fichier ne se trouve pas dans le répertoire. Sinon le drapeau de zéro est levé, `BP` pointe sur le bloc des paramètres du lecteur de disquette concerné, les segments des données et supplémentaire sont confondus avec le segment de code, `SI` contient le numéro de la première unité d'allocation du fichier, DIRBUF pointe sur l'entrée de répertoire correspondante, la variable chaîne de caractères NAME1 contient le nom du fichier, tout en majuscules, et `ES:DI` pointe sur le FCB.

— Si on ne trouve pas le nom du fichier dans le répertoire, on place `00h` comme secteur de début du répertoire, `FFh` dans `AL` et on termine la routine (lignes 1042 et 1095–1099).

On est renvoyé à la partie de code BADCLOSE lorsqu'on ne peut pas fermer le fichier :

```
1095 : BADCLOSE:
1096 :          MOV     SI,[BP.FAT]
1097 :          MOV     BYTE PTR [SI-1],0
1098 :          MOV     AL,-1
1099 :          RET
```

— Si on trouve le nom du fichier dans le répertoire, on renseigne l'entrée de répertoire sur la copie du secteur du répertoire située et mémoire centrale et on appelle la routine DIR-WRITE pour l'enregistrer sur la diquette (lignes 1043–1053).

— On passe à la routine CHKFATWRT pour enregistrer sur la disquette la copie de la FAT se trouvant en mémoire centrale si elle ne l'a pas été depuis sa dernière modification (ligne 1055).

5.8.6 Routine de la fonction 19 (13h) de suppression d'un fichier spécifié par un FCB

5.8.6.1 Routine de service

La routine de service de la fonction 19 (`13h`) de l'interruption `21h`, de suppression d'un fichier spécifié par un FCB, est repérée par l'étiquette DELETE :

```
704 : DELETE: ; System call 19
705 :         CALL    MOVNAME
706 :         MOV     AL,-1
707 :         JC      RET4
708 :         MOV     AL,CS:[ATTRIB]
709 :         AND     AL,6                    ;Look only at hidden bits
710 :         CMP     AL,6                    ;Both must be set
711 :         JNZ     NOTALL
712 :         MOV     CX,11
713 :         MOV     AL,"?"
714 :         MOV     DI,OFFSET DOSGROUP:NAME1
715 :         REPE    SCASB                   ;See if name is *.*
716 :         JNZ     NOTALL
717 :         MOV     BYTE PTR CS:[DELALL],0  ;DEL *.* - flag deleting all
718 : NOTALL:
719 :         CALL    FINDNAME
720 :         MOV     AL,-1
721 :         JC      RET4
722 :         OR      BH,BH                   ;Check if device name
723 :         JS      RET4                    ;Can't delete I/O devices
724 : DELFILE:
725 :         MOV     BYTE PTR [DIRTYDIR],-1
726 :         MOV     AH,[DELALL]
727 :         MOV     BYTE PTR [BX],AH
728 :         MOV     BX,[SI]
729 :         MOV     SI,[BP.FAT]
730 :         OR      BX,BX
731 :         JZ      DELNXT
732 :         CMP     BX,[BP.MAXCLUS]
733 :         JA      DELNXT
734 :         CALL    RELEASE
735 : DELNXT:
736 :         CALL    CONTSRCH
737 :         JNC     DELFILE
738 :         CALL    FATWRT
739 :         CALL    CHKDIRWRITE
740 :         XOR     AL,AL
741 :         RET
```

— `DS:DX` pointant sur un FCB, on appelle la routine MOVNAME (ligne 705) pour récupérer le nom du fichier spécifié par le FCB.

Plus précisément on fait pointer `DS:DX` au début du FCB standard associé s'il s'agit d'un FCB étendu, on confond le segment supplémentaire avec le segment de code, la chaîne de caractères NAME1 contient le nom de fichier spécifié par le FCB, tout en majuscules, et on fait pointer `BP` sur le bloc des paramètres du lecteur de disquette spécifié par le FCB.

— Si le nom de fichier n'est pas valide, on place `FFh` dans `AL` et on termine la routine, le drapeau CF étant levé (lignes 706–707).

— Sinon, si les attributs du fichier sont « caché » ou « système », si on ne trouve pas le nom du fichier dans le répertoire ou s'il s'agit d'un fichier de périphérique, on renvoie `FFh` dans `AL` pour indiquer que l'entrée n'a pas été trouvée et on termine la routine (lignes 708–711, 718–723 et 703).

— Si le nom de fichier est « *.* », indiquant par là qu'il faut supprimer toutes les entrées du répertoire, on place donc 0 dans la variable DELALL (lignes 712–717).

— On place `FFh` dans la variable DIRTYDIR pour indiquer qu'on a modifié la copie du secteur du répertoire se trouvant en mémoire centrale depuis sa dernière sauvegarde (ligne 725).

— Après avoir appelé FINDNAME pour trouver le nom du fichier dans une entrée du répertoire, `BX` pointe sur le tampon du secteur du répertoire se trouvant en mémoire centrale et `SI` contient la valeur du champ 'première unité d'allocation' de l'entrée trouvée. On place la valeur de DELALL comme premier octet de celle-ci (lignes 726 et 727).

— On place le numéro de la première unité d'allocation du fichier dans `BX` (ligne 728). Si elle est nulle ou supérieure au nombre maximum d'unités d'allocation, on passe à l'entrée suivante (lignes 730–733).

— On fait pointer `SI` au début de la copie de la FAT en mémoire centrale (ligne 729) et on appelle la routine RELEASE, étudiée ci-dessous, pour libérer toutes les unités d'allocation allouées à ce fichier (ligne 734).

— L'étiquette CONTSRCH renvoie à une partie de la routine GETNAME permettant de trouver l'entrée du répertoire contenant le nom du fichier. Tant qu'on trouve une entrée de nom concordant, on effectue la même chose que ci-dessus (lignes 736 et 737).

— Puisque la copie de la FAT en mémoire centrale a été modifiée, on l'enregistre sur la disquette, on indique que la copie de la FAT en mémoire centrale n'a pas été modifiée depuis sa dernière sauvegarde, on place `00h` dans `AL` pour indiquer que tout s'est bien déroulé et on termine la routine (lignes 738–741).

5.8.6.2 Libération des unités d'allocation d'un fichier

La routine RELEASE libère toutes les unités d'allocation chaînées à partir du numéro d'unité d'allocation contenu dans `BX`, passé en paramètre.

Avant d'appeler cette routine, le segment des données doit être confondu avec le segment de code, `BX` doit contenir le numéro d'une unité d'allocation, `SI` pointer sur le début de la copie de la FAT en mémoire centrale et `BP` sur le bloc des paramètres du lecteur de disquette concerné :

```
2449 : RET12:  RET
2450 :
2451 :
2452 : RELEASE:
2453 :
2454 : ; Inputs:
2455 : ;       DS = CS
2456 : ;       BX = Cluster in file
2457 : ;       SI = FAT pointer
2458 : ;       BP = Base of drive parameters
2459 : ; Function:
2460 : ;       Frees cluster chain starting with [BX]
2461 : ; AX,BX,DX,DI all destroyed. Other registers unchanged.
2462 :
2463 :         XOR     DX,DX
2464 : RELBLKS:
2465 : ; Enter here with DX=0FFFH to put an end-of-file mark
2466 : ; in the first cluster and free the rest in the chain.
2467 :         CALL    UNPACK
2468 :         JZ      RET12
2469 :         MOV     AX,DI
2470 :         CALL    PACK
2471 :         CMP     AX,0FF8H
2472 :         MOV     BX,AX
2473 :         JB      RELEASE
2474 : RET13:  RET
```

— On place `00h` dans `DX`, valeur à placer dans les entrées de la FAT qu'on va désallouer.

— On récupère dans `DI` le contenu de l'entrée de la FAT correspondant au numéro d'unité d'allocation contenu dans `BX`. Si cette entrée est indiquée libre, on termine la routine (lignes 2467–2468 et 2449).

— Sinon on place le contenu de `DI` dans `AX` et on appelle la routine PACK pour changer la valeur, indiquant que cette entrée est désormais libre (lignes 2469–2470).

— Si le contenu était `FF8h`, c'est qu'on est arrivé à la fin de la liste chaînée des numéros d'unités d'allocation constituant le fichier, on termine donc la routine (lignes 2471 et 2473–2474).

— Sinon on recommence (récursivement) pour l'unité d'allocation suivante du fichier (lignes 2472 et 2473).

5.8.7 Routine de service de la fonction 23 (17h) de renommage d'un fichier spécifié par un FCB

5.8.7.1 Routine principale

La routine de service de la fonction `17h` de l'interruption `21h`, de renommage d'un fichier spécifié par un FCB, est repérée par l'étiquette RENAME :

```
744 : RENAME: ;System call 23
745 :         CALL    MOVNAME
746 :         JC      ERRET
747 :         ADD     SI,5
748 :         MOV     DI,OFFSET DOSGROUP:NAME2
749 :         CALL    LODNAME
750 :         JC      ERRET                   ;Report error if second name invalid
751 :         CALL    FINDNAME
752 :         JC      ERRET
753 :         OR      BH,BH                   ;Check if I/O device name
754 :         JS      ERRET                   ;If so, can't rename it
755 :         MOV     SI,OFFSET DOSGROUP:NAME1
756 :         MOV     DI,OFFSET DOSGROUP:NAME3
757 :         MOV     CX,6                    ;6 words (12 bytes)--include attribute byte
758 :         REP     MOVSW                   ;Copy name to search for
759 : RENFIL:
760 :         MOV     DI,OFFSET DOSGROUP:NAME1
761 :         MOV     SI,OFFSET DOSGROUP:NAME2
762 :         MOV     CX,11
763 : NEWNAM:
764 :         LODSB
765 :         CMP     AL,"?"
766 :         JNZ     NOCHG
767 :         MOV     AL,[BX]
768 : NOCHG:
769 :         STOSB
770 :         INC     BX
771 :         LOOP    NEWNAM
772 :         MOV     BYTE PTR [DI],6         ;Stop duplicates with any attributes
773 :         CALL    DEVNAME                 ;Check if giving it a device name
774 :         JNC     RENERR
775 :         PUSH    [LASTENT]               ;Save position of match
776 :         MOV     [LASTENT],-1            ;Search entire directory for duplicate
777 :         CALL    CONTSRCH                ;See if new name already exists
778 :         POP     AX
779 :         JNC     RENERR                  ;Error if found
780 :         CALL    GETENT                  ;Re-read matching entry
781 :         MOV     DI,BX
782 :         MOV     SI,OFFSET DOSGROUP:NAME1
783 :         MOV     CX,5
784 :         MOVSB
785 :         REP     MOVSW                   ;Replace old name with new one
786 :         MOV     BYTE PTR [DIRTYDIR],-1  ;Flag change in directory
787 :         MOV     SI,OFFSET DOSGROUP:NAME3
788 :         MOV     DI,OFFSET DOSGROUP:NAME1
789 :         MOV     CX,6                    ;Include attribute byte
790 :         REP     MOVSW                   ;Copy name back into search buffer
791 :         CALL    CONTSRCH
792 :         JNC     RENFIL
793 :         CALL    CHKDIRWRITE
794 :         XOR     AL,AL
795 :         RET
796 :
797 : RENERR:
```

```
798 :          CALL    CHKDIRWRITE
799 : ERRET:
800 :          MOV     AL,-1
801 : RET5:    RET
```

— `DS:DX` pointe sur un FCB dont les champs 'code de lecteur de disquette' et 'nom de fichier' sont renseignés : le second nom de fichier doit se trouver à l'emplacement d'adresse `DS:DX + 11h`. On appelle la routine MOVNAME (ligne 745) pour récupérer le nom du fichier source spécifié par le FCB.

Plus précisément on fait pointer `DS:DX` au début du FCB standard associé s'il s'agit d'un FCB étendu, on confond le segment supplémentaire avec le segment de code, la chaîne de caractères NAME1 contient le nom de fichier spécifié par le FCB, tout en majuscules, et on fait pointer `BP` sur le bloc des paramètres du lecteur de disquette spécifié par le FCB.

— Si le nom de fichier n'est pas valide, on place `FFh` dans `AL` et on termine la routine, le drapeau CF étant levé (lignes 746 et 799–801).

— Sinon on ajoute 5 à `SI` pour qu'il pointe sur le champ `11h`, champ réservé du FCB contenant le nom du fichier de destination (ligne 747).

— On fait pointer `DI` sur la variable chaîne de caractères NAME2 et on appelle la routine LODNAME, étudiée ci-dessous, pour copier le nom du fichier de destination dans NAME2, tout en le convertissant en majuscules (lignes 748–749).

La variable NAME2 chaîne de caractères contient le nom du fichier de destination :

```
3685 : NAME2   DB      11 DUP (?)
```

— Si le nom du fichier de destination n'est pas valide, on place `FFh` dans `AL` et on termine la routine, le drapeau CF étant levé (lignes 750 et 799–801).

— Sinon on appelle la sous-routine FINDNAME, partie de la routine GETNAME, pour chercher une entrée dans le répertoire, le nom du fichier étant spécifié par le champ adéquat du FCB situé à l'adresse `DS:DX` (ligne 751).

On peut utiliser le caractère d'ambiguïté '?' dans le nom, représentant n'importe quel caractère permis pour un nom de fichier. Le drapeau CF est levé s'il n'existe pas de nom de fichier concordant dans le répertoire. Sinon le drapeau de zéro ZF est levé, `BP` pointe sur le bloc des paramètres du lecteur de disquette concerné, les segments de données et supplémentaire sont confondus avec le segment de code, `SI` pointe sur la première unité d'allocation du fichier, DIRBUF pointe sur l'entrée de répertoire correspondante et la variable chaîne de caractères NAME1 contient le nom du fichier tout en majuscules.

— Si on n'en trouve pas ou s'il s'agit d'un nom de périphérique, on place `FFh` dans `AL` et on termine la routine, le drapeau CF étant levé (lignes 751–754 et 799–801).

— Sinon on fait pointer `SI` sur NAME1, `DI` sur NAME3, on place 6 dans `CX` pour copier 6 mots, soit 12 octets (c'est-à-dire le nom de fichier précédé de l'octet d'attribut), et on effectue cette copie (lignes 755–758).

La variable chaîne de caractères NAME3 contient un nom du fichier ainsi que l'octet d'attribut :

```
3686 : NAME3   DB      12 DUP (?)
```

— On fait pointer `DI` sur NAME1, `SI` sur NAME2, on place 11 dans `CX`, le nombre de caractères d'un nom de fichier, et on passe en revue tous les caractères de NAME2. Lorsqu'il s'agit du caractère d'ambiguïté, on le remplace dans `AL` par le caractère de même position du nom du fichier intermédiaire, et on le reporte dans NAME1 (lignes 760–771).

— On place 6 comme octet d'attribut pour NAME1 et on appelle DEVNAME pour savoir si le nom de fichier placé dans NAME1 est un nom de fichier spécial, en levant CF si c'est le cas (lignes 772–773).

— S'il s'agit du nom d'un périphérique, on appelle la sous-routineCHKDIRWRITE pour savoir si la copie du secteur du répertoire se trouvant actuellement en mémoire centrale a été modifiée depuis sa dernière sauvegarde, en l'enregistrant sur la disquette si besoin est, et on termine la routine, le drapeau CF étant levé (lignes 774 et 797–801).

— S'il ne s'agit pas d'un nom de périphérique, on sauvegarde sur la pile le numéro de la dernière entrée de répertoire lue, on place `FFh` dans la variable LASTENT, pour indiquer de chercher dans tout le répertoire, et on appelle CONTSRCH, partie de la routine FINDNAME, pour voir si le nouveau nom apparaît déjà dans le répertoire (lignes 775–778).

— S'il y apparaît, on appelle la sous-routine CHKDIRWRITE pour savoir si la copie du secteur du répertoire se trouvant actuellement en mémoire centrale a été modifiée depuis la dernière sauvegarde, en l'enregistrant sur la disquette si besoin est, et on termine la routine, le drapeau CF étant levé (lignes 779 et 797–801).

— S'il n'y apparaît pas, on restaure la valeur de la dernière entrée de répertoire lue dans `AX` et on appelle GETENT, partie de la routine GETENTRY, pour obtenir l'entrée libre suivante du répertoire.

  Après exécution, le drapeau CF est levé si aucune entrée libre n'a été trouvée. Sinon `AL` contient le numéro du secteur du répertoire sur lequel se trouve la nouvelle entrée, `BX` pointe sur la nouvelle entrée dans la zone de mémoire centrale repérée par DIRBUF, `DX` pointe sur le premier octet suivant DIRBUF et LASTENT donne le numéro de cette nouvelle entrée de répertoire (lignes 778 et 780).

— On fait pointer `DI` sur la nouvelle entrée dans la zone de mémoire centrale repérée par DIRBUF, `SI` sur NAME1, on recopie les 11 octets, de façon à renseigner le champ 'nom de fichier' de l'entrée de répertoire, remplaçant ainsi l'ancien nom par le nouveau, et on place `FFh` dans la variable DITYDIR, pour indiquer que la copie du secteur de répertoire se trouvant en mémoire centrale a été modifiée depuis sa dernière sauvegarde (lignes 781–786).

— On fait pointer `SI` sur NAME3, `DI` sur NAME1 et on copie le premier nom sur le second, y compris l'octet d'attribut (lignes 787–790).

— On appelle CONTSRCH, partie de la routine FINDNAME, pour voir si le nouveau nom apparaît déjà dans le répertoire (ligne 791).

— S'il n'y apparaît pas, on recommence à partir de létiquette RENFIL (ligne 792).

— S'il y apparaît, on appelle la sous-routine CHKDIRWRITE pour savoir si la copie du secteur du répertoire se trouvant en mémoire centrale a été modifiée depuis sa dernière sauvegarde, en l'enregistrant sur la disquette si besoin est, on place `00h` dans `AL` et on termine la routine (lignes 793–795).

5.8.7.2 Sous-routine de copie d'un nom de fichier en le convertissant en majuscules

La routine LODNAME copie un nom de fichier de `DS:SI` en `ES:DI`, tout en le convertissant en majuscules :

```
834 : LODNAME:
835 : ; This entry point copies a file name from DS,SI
836 : ; to ES,DI converting to upper case.
837 :         CMP     BYTE PTR [SI]," "  ;Don't allow blank as first letter
838 :         STC                        ;In case of error
839 :         JZ      RET5
840 :         MOV     CX,11
841 : MOVCHK:
842 :         CALL    GETLET
843 :         JB      RET5
844 :         JNZ     STOLET             ;Is it a delimiter?
845 :         CMP     AL," "             ;This is the only delimiter allowed
846 :         STC                        ;In case of error
847 :         JNZ     RET5
848 : STOLET:
849 :         STOSB
850 :         LOOP    MOVCHK
851 :         CLC                        ;Got through whole name - no error
852 : RET6:   RET
```

— Si le premier caractère du nom de fichier source est un espace, on lève le drapeau CF et on termine la routine (lignes 837–839).

— Sinon on reporte les 11 caractères du nom de fichier source dans le nom de fichier but, en appelant la sous-routine de conversion en majuscules (qui, de plus, lève l'indicateur ZF s'il s'agit d'un délimiteur et l'indicateur CF s'il s'agit d'un caractère de contrôle autre que TAB), en levant le drapeau CF et en terminant la routine s'il s'y trouve un délimiteur autre que l'espace, en baissant l'indicateur CF sinon (lignes 840–852).

5.8.8 Opérations sur les unités d'allocation

5.8.8.1 Conversion unité d'allocation/secteur en numéro du secteur

La routine FIGREC donne le numéro d'un secteur spécifié par une unité d'allocation et la position du secteur (0 ou 1 pour MS-DOS 1.25) à l'intérieur de celle-ci.

Avant d'appeler la routine, `DX` doit contenir le numéro de l'unité d'allocation, `BL` la position du secteur dans cette unité d'allocation et `BP` doit pointer sur le bloc des paramètres du lecteur de disquette.

Après exécution de la routine, `DX` contient le numéro du secteur :

```
2315 : FIGREC:
2316 :
2317 : ;Inputs:
2318 : ;       DX = Physical cluster number
2319 : ;       BL = Sector postion within cluster
2320 : ;       BP = Base of drive parameters
2321 : ;Outputs:
2322 : ;       DX = physical sector number
2323 : ;No other registers affected.
2324 :
2325 :         PUSH    CX
2326 :         MOV     CL,[BP.CLUSSHFT]
2327 :         DEC     DX
2328 :         DEC     DX
2329 :         SHL     DX,CL
2330 :         OR      DL,BL
2331 :         ADD     DX,[BP.FIRREC]
2332 :         POP     CX
2333 :         RET
```

Remarquons l'hésitation sur la dénomination : on a FIGREC et non FIGCLUS.

Autrement dit, le contenu de `CX` lors de l'appel de la routine est sauvegardé sur la pile, puisqu'on va utiliser ce registre, on place le nombre de secteurs d'une unité d'allocation dans `CL`, on soustrait 2 du numéro physique de l'unité d'allocation, puisque la première unité d'allocation des données porte le numéro 2, et on le multiplie par le nombre de secteurs par unité d'allocation. On ajoute au nombre obtenu le numéro du premier secteur de la première unité d'allocation, ce qui donne le numéro voulu. On restaure la valeur de `CX` et on termine la routine.

5.8.8.2 Routine de mise en tampon d'un des secteurs d'une unité d'allocation

La routine BUFSEC copie en mémoire centrale le secteur de la disquette spécifié par son numéro d'unité d'allocation et sa position dans celle-ci.

Avant d'appeler la routine, `AL` doit contenir 0 si le tampon doit être chargé, 1 sinon, `BP` doit pointer sur le bloc des paramètres du lecteur de disquette, CLUSNUM contenir le numéro de l'unité d'allocation voulue, SECCLUSPOS la position du secteur dans cette unité d'allocation et BYTCNT1 la taille du transfert.

La routine s'assure que le contenu du secteur se trouve dans la copie du secteur se trouvant en mémoire centrale, le chargeant si besoin est.

La variable SECPLUPOS contient la position d'un secteur dans une unité d'allocation, CLUSNUM un numéro d'unité d'allocation, BYTCNT1 la taille, en octets, de ce qui doit être transféré :

```
3695 : SECCLUSPOS DB   ?       ;Position of first sector within cluster
[...]
3708 : CLUSNUM DW      ?
[...]
3713 : BYTCNT1 DW      ?       ;No. of bytes in first sector
```

Après exécution de la routine, `SI` pointe sur la copie du secteur en mémoire centrale, `DI` sur l'adresse du tampon de transfert, `CX` contient le nombred'octets transférés, NEXTADD est mis à jour et TRANS indique qu'un transfert va être effectué.

La variable TRANS indique qu'un transfert va être effectué, NEXTADD l'adresse du prochain transfert :

```
3697 : TRANS   DB      ?
[...]
3704 : NEXTADD DW      ?
```

La routine est BUFSEC :

```
BUFSEC:
; Inputs:
;       AL = 0 if buffer must be read, 1 if no pre-read needed
;       BP = Base of drive parameters
;       [CLUSNUM] = Physical cluster number
;       [SECCLUSPOS] = Sector position of transfer within cluster
;       [BYTCNT1] = Size of transfer
; Function:
;       Insure specified sector is in buffer, flushing buffer before
;       read if necessary.
; Outputs:
;       SI = Pointer to buffer
;       DI = Pointer to transfer address
;       CX = Number of bytes
;       [NEXTADD] updated
;       [TRANS] set to indicate a transfer will occur

        MOV     DX,[CLUSNUM]
        MOV     BL,[SECCLUSPOS]
        CALL    FIGREC
        MOV     [PREREAD],AL
        CMP     DX,[BUFSECNO]
        JNZ     GETSEC
        MOV     AL,[BUFDRVNO]
        CMP     AL,[THISDRV]
        JZ      FINBUF                  ;Already have it?
GETSEC:
        XOR     AL,AL
        XCHG    [DIRTYBUF],AL           ;Read dirty flag and reset it
        OR      AL,AL
        JZ      RDSEC
        PUSH    DX
        PUSH    BP
        MOV     BP,[BUFDRVBP]
        MOV     BX,[BUFFER]
        MOV     CX,1
        MOV     DX,[BUFSECNO]
        CALL    DWRITE
        POP     BP
        POP     DX
RDSEC:
        TEST    BYTE PTR [PREREAD],-1
        JNZ     SETBUF
        XOR     AX,AX
        MOV     [BUFSECNO],AX           ;Set buffer valid in case of disk error
        DEC     AX
        MOV     [BUFDRVNO],AL
        MOV     BX,[BUFFER]
        MOV     CX,1
        PUSH    DX
        CALL    DREAD
```

```
        POP     DX
SETBUF:
        MOV     [BUFSECNO],DX
        MOV     AL,[THISDRV]
        MOV     [BUFDRVNO],AL
        MOV     [BUFDRVBP],BP
FINBUF:
        MOV     BYTE PTR [TRANS],1      ;A transfer is taking place
        MOV     DI,[NEXTADD]
        MOV     SI,DI
        MOV     CX,[BYTCNT1]
        ADD     SI,CX
        MOV     [NEXTADD],SI
        MOV     SI,[BUFFER]
        ADD     SI,[BYTSECPOS]
        RET
```

— On place le numéro d'unité d'allocation dans `DX`, la position du secteur dans cette unité d'allocation dans `BL` et on appelle la routine FIGREC pour obtenir le numéro absolu du secteur dans `DX` (lignes 1677–1679).

— On place la condition de transfert du tampon dans la variable PREREAD (ligne 1680).

La variable PREREAD vaut 0 pour exiger qu'on lise avant de lire le tampon :

```
3698 : PREREAD DB      ?       ;0 means preread; 1 means optional
```

— Si le numéro du secteur contenu dans le tampon de secteur situé en mémoire centrale est celui que l'on vient d'obtenir et si le numéro de lecteur de disquette de ce secteur dans le tampon est celui du lecteur qui nous intéresse, on indique que le transfert s'est bien effectué, on met à jour le contenu de la variable NEXTADD, on fait pointer `SI` sur le tampon de secteur situé en mémoire centrale et on termine la routine (lignes 1681–1685 et 1717–1726).

— Sinon on lit le contenu de la variable DIRTYBUF et on la met à zéro pour indiquer que le tampon n'a pas été modifié depuis sa dernière sauvegarde (lignes 1687–1688).

— Si la copie du secteur se trouvant actuellement dans le tampon de secteur a été modifié depuis sa dernière sauvegarde, on l'enregistre sur la disquette (lignes 1689–1699).

— Si la variable PREREAD ne contient pas `FFh`, on place `00h` comme numéro de secteur dans le tampon, `FFh` comme numéro du lecteur de disquette dont le secteur est dans le tampon, ce qui correspond à des précautions en cas d'erreur de lecture sur le disque, on place l'adresse du tampon dans `AX`, 1 dans `CX`, pour indiquer qu'on ne veut lire qu'un seul secteur, on sauvegarde le contenu de `DX`, on fait appel à la routine DREAD pour lire le secteur voulu et le placer dans le tampon et on restaure la valeur de `DX` (lignes 1700–1711).

— On renseigne les variables BUFSECNO, BUFDRVNO, BUFDRVBP et on termine la routine (lignes 1712–1726).

5.8.8.3 Incrémentation du couple numéro d'unité d'allocation/numéro de secteur

Un numéro d'unité d'allocation de fichier et la position du secteur dans celle-ci étant donnés par les contenus des variables CLUSNUM et SECCLUSPOS, la routine NEXTSEC détermine le numéro d'unité d'allocation et la position du secteur dans celle-ci du secteur suivant du fichier :

```
1767 : NEXTSEC:
1768 :          TEST    BYTE PTR [TRANS],-1
1769 :          JZ      CLRET
1770 :          MOV     AL,[SECCLUSPOS]
1771 :          INC     AL
1772 :          CMP     AL,[BP.CLUSMSK]
1773 :          JBE     SAVPOS
1774 :          MOV     BX,[CLUSNUM]
1775 :          CMP     BX,0FF8H
1776 :          JAE     NONEXT
1777 :          MOV     SI,[BP.FAT]
1778 :          CALL    UNPACK
1779 :          MOV     [CLUSNUM],DI
1780 :          INC     [LASTPOS]
1781 :          MOV     AL,0
1782 : SAVPOS:
1783 :          MOV     [SECCLUSPOS],AL
1784 : CLRET:
1785 :          CLC
1786 :          RET
1787 : NONEXT:
1788 :          STC
1789 :          RET
```

— Si le contenu de la variable TRANS est `FFh`, on baisse le drapeau CF et on termine la routine (lignes 1768–1769 et 1784–1786).

— Sinon on place la position du secteur dans l'unité d'allocation (0 ou 1 pour MS-DOS 1.25) dans `AL`, que l'on incrémente (pour obtenir 1 ou 2). Si ce numéro est inférieur au nombre de secteurs dans une unité d'allocation, on le sauvegarde dans la variable SECCLUSPOS (lignes 1770–1773).

— Sinon si l'entrée de la FAT est `FF8h` pour le numéro d'unité d'allocation, indiquant qu'il s'agit de la dernière unité d'allocation du fichier, on lève le drapeau CF et on termine la routine (lignes 1774–1776 et 1787–1789).

— Sinon on récupère dans la FAT le numéro de l'unité d'allocation suivante du fichier, que l'on place dans la variable CLUSNUM (lignes 1777–1779).

— On incrémente la variable LASTPOS, on place 0 dans SECCLUSPOS, puisque le secteur suivant a nécessairement cette position dans la nouvelle unité d'allocation, on baisse le drapeau CF et on termine la routine (lignes 1780–1786).

La variable LASTPOS contient la dernière position du secteur dans une unité d'allocation :

```
3707 : LASTPOS DW      ?
```

5.8.8.4 Saut à une unité d'allocation de position donnée d'un fichier

La routine FNDCLUS essaie de parvenir à la n-ième unité d'allocation d'un fichier donné, plus précisément à celle de position $min(n, d)$ si d est le position dans le fichier de sa dernière unité d'allocation.

Avant d'appeler cette routine, le segment des données doit être confondu avec le segment de code, `CX` contenir la position n dans le fichier de l'unité d'allocation à laquelle on désire parvenir, `BP` pointer sur le bloc des paramètres du lecteur de la disquette sur laquelle se trouve le fichier, `SI` sur la copie de la FAT en mémoire centrale et `ES:DI` sur le FCB du fichier.

Après son exécution, `BX` contient le numéro sur la disquette de l'unité d'allocation atteinte, `CX` le nombre d'unités d'allocation supplémentaires que le fichier aurait dû contenir pour atteindre celle de position n et `DX` la position dans le fichier de l'unité d'allocation à laquelle on est parvenu, soit $min(n, d)$:

```
1621 : FNDCLUS:
1622 :
1623 : ; Inputs:
1624 : ;       DS = CS
1625 : ;       CX = No. of clusters to skip
1626 : ;       BP = Base of drive parameters
1627 : ;       SI = FAT pointer
1628 : ;       ES:DI point to FCB
1629 : ; Outputs:
1630 : ;       BX = Last cluster skipped to
1631 : ;       CX = No. of clusters remaining (0 unless EOF)
1632 : ;       DX = Position of last cluster
1633 : ; DI destroyed. No other registers affected.
1634 :
1635 :         MOV     BX,ES:[DI.LSTCLUS]
1636 :         MOV     DX,ES:[DI.CLUSPOS]
1637 :         OR      BX,BX
1638 :         JZ      NOCLUS
1639 :         SUB     CX,DX
1640 :         JNB     FINDIT
1641 :         ADD     CX,DX
1642 :         XOR     DX,DX
1643 :         MOV     BX,ES:[DI.FIRCLUS]
1644 : FINDIT:
1645 :         JCXZ    RET10
1646 : SKPCLP:
1647 :         CALL    UNPACK
1648 :         CMP     DI,0FF8H
1649 :         JAE     RET10
1650 :         XCHG    BX,DI
1651 :         INC     DX
1652 :         LOOP    SKPCLP
1653 :         RET
1654 : NOCLUS:
1655 :         INC     CX
1656 :         DEC     DX
1657 :         RET
```

— On place le numéro de la dernière unité d'allocation utilisée par le fichier dans `BX` et la position dans le fichier de la dernière unité d'allocation utilisée dans `DX` (lignes 1635–1636).

— Si le fichier est vide, le nombre d'unités d'allocation que l'on n'a pas pu sauter est le nombre d'unités d'allocation à sauter plus un, la position cherchée est -1 (lignes 1637–1638 et 1654–1657).

— Sinon on soustrait la position dans le fichier de la dernière unité d'allocation utilisée au numéro de la dernière unité d'allocation utilisée par le fichier. Si on trouve un nombre négatif, on restaure le numéro de la dernière unité d'allocation utilisée par le fichier, on place 0 comme position dans le fichier de la dernière unité d'allocation utilisée et on place dans `BX` le numéro de la première unité d'allocation du fichier (lignes 1639–1643).

— S'il n'y a aucune unité d'allocation restante, on termine la routine (lignes 1644–1645).

— Sinon on appelle la routine UNPACK pour obtenir l'entrée dans la FAT de la dernière unité d'allocation. Si on trouve `FF8h`, c'est-à-dire s'il s'agit de la dernière unité d'allocation du fichier, on termine la routine (lignes 1646–1649).

— Sinon on place le numéro de l'unité pointée dans `BX`, on incrémente `DX`, c'est-à-dire la position de la dernière unité d'allocation utilisée et on recommence jusqu'à ce qu'on ait sauté le nombre voulu d'unités d'allocation et on termine la routine (lignes 1650–1653).

5.8.8.5 Ajout d'unités d'allocation nouvelles à un fichier

La routine ALLOCATE essaie d'ajouter un certain nombre d'unités d'allocation nouvelles à la fin d'un fichier.

Avant d'appeler cette routine, le segment des données doit être confondu avec le segment de code, le segment supplémentaire doit contenir le FCB, `BX` le numéro de la dernière unité d'allocation du fichier (0 s'il s'agit d'un fichier vide), `CX` le nombre d'unités d'allocation nouvelles souhaitées, `DX` la position dans le fichier de l'unité d'allocation de numéro contenu dans `BX`, `BP` pointer sur le bloc des paramètres du lecteur de disquette, `SI` sur la copie de la FAT en mémoire centrale et FCB sur le FCB dans le segment supplémentaire.

Après exécution de cette routine, s'il n'y a pas suffisamment d'unités d'allocation libres sur la disquette, le drapeau CF est levé et `CX` contient le nombre maximum de secteurs pouvant être ajoutés à ce fichier. Sinon le drapeau CF est baissé, `BX` contient le numéro de la première unité d'allocation allouée, la copie de la FAT en mémoire centrale est mise à jour, y compris l'octet indiquant une modification de celle-ci depuis sa dernière sauvegarde sur la disquette, et le champ FIRCLUS du FCB est is à jour si le fichier était vide :

```
2362 : ALLOCATE:
2363 :
2364 : ; Inputs:
2365 : ;       DS = CS
2366 : ;       ES = Segment of FCB
2367 : ;       BX = Last cluster of file (0 if null file)
2368 : ;       CX = No. of clusters to allocate
2369 : ;       DX = Position of cluster BX
2370 : ;       BP = Base of drive parameters
2371 : ;       SI = FAT pointer
2372 : ;       [FCB] = Displacement of FCB within segment
2373 : ; Outputs:
2374 : ;       IF insufficient space
2375 : ;         THEN
2376 : ;       Carry set
2377 : ;       CX = max. no. of records that could be added to file
2378 : ;         ELSE
2379 : ;       Carry clear
2380 : ;       BX = First cluster allocated
2381 : ;       FAT is fully updated including dirty bit
2382 : ;       FIRCLUS field of FCB set if file was null
2383 : ; SI,BP unchanged. All other registers destroyed.
2384 :
2385 :         PUSH    [SI]
2386 :         PUSH    DX
```

```
2387 :          PUSH    CX
2388 :          PUSH    BX
2389 :          MOV     AX,BX
2390 : ALLOC:
2391 :          MOV     DX,BX
2392 : FINDFRE:
2393 :          INC     BX
2394 :          CMP     BX,[BP.MAXCLUS]
2395 :          JLE     TRYOUT
2396 :          CMP     AX,1
2397 :          JG      TRYIN
2398 :          POP     BX
2399 :          MOV     DX,0FFFH
2400 :          CALL    RELBLKS
2401 :          POP     AX                      ;No. of clusters requested
2402 :          SUB     AX,CX                   ;AX=No. of clusters allocated
2403 :          POP     DX
2404 :          POP     [SI]
2405 :          INC     DX                      ;Position of first cluster allocated
2406 :          ADD     AX,DX                   ;AX=max no. of cluster in file
2407 :          MOV     DL,[BP.CLUSMSK]
2408 :          MOV     DH,0
2409 :          INC     DX                      ;DX=records/cluster
2410 :          MUL     DX                      ;AX=max no. of records in file
2411 :          MOV     CX,AX
2412 :          SUB     CX,WORD PTR [RECPOS]    ;CX=max no. of records that could be written
2413 :          JA      MAXREC
2414 :          XOR     CX,CX                   ;If CX was negative, zero it
2415 : MAXREC:
2416 :          STC
2417 : RET11:   RET
2418 :
2419 : TRYOUT:
2420 :          CALL    UNPACK
2421 :          JZ      HAVFRE
2422 : TRYIN:
2423 :          DEC     AX
2424 :          JLE     FINDFRE
2425 :          XCHG    AX,BX
2426 :          CALL    UNPACK
2427 :          JZ      HAVFRE
2428 :          XCHG    AX,BX
2429 :          JMP     SHORT FINDFRE
2430 : HAVFRE:
2431 :          XCHG    BX,DX
2432 :          MOV     AX,DX
2433 :          CALL    PACK
2434 :          MOV     BX,AX
2435 :          LOOP    ALLOC
2436 :          MOV     DX,0FFFH
2437 :          CALL    PACK
2438 :          MOV     BYTE PTR [SI-1],1
2439 :          POP     BX
2440 :          POP     CX                      ;Don't need this stuff since we're successful
2441 :          POP     DX
2442 :          CALL    UNPACK
2443 :          POP     [SI]
2444 :          XCHG    BX,DI
2445 :          OR      DI,DI
2446 :          JNZ     RET11
2447 :          MOV     DI,[FCB]
2448 :          MOV     ES:[DI.FIRCLUS],BX
```

```
2449 : RET12:  RET
```

- On sauvegarde sur la pile le décalage de la FAT, la position de la dernière unité d'allocation du fichier, le nombre d'unités d'allocation à allouer et le numéro de la dernière unité d'allocation du fichier (lignes 2385–2388).
- On place dans `AX` et dans `DX` le numéro de la dernière unité d'allocation du fichier et on incrémente `BX` pour qu'il contienne le numéro suivant. Si ce numéro est inférieur au nombre maximum d'unités d'allocation de la disquette, on essaie d'allouer cette unité d'allocation au fichier (lignes 2389–2395).
  - Pour cela on appelle la routine UNPACK pour obtenir le numéro d'allocation suivante du fichier (lignes 2419–2420).
  - Si celle-ci est libre, on place dans `BX` la position de la dernière unité d'allocation actuelle du fichier, dans `AX` le numéro de celle-ci et on appelle la routine PACK pour ajouter cette unité d'allocation libre au fichier (lignes 2421 et 2430–2433).
  - On place le numéro de cette nouvelle unité d'allocation dans `BX` et on recommence autant de fois qu'il y a d'unités d'allocation à allouer (lignes 2434–2435).
  - Pour la dernière unité d'allocation à ajouter, on place `FFFh` comme entrée dans la FAT pour celle-ci, pour indiquer la fin du fichier, on place `FFh` sur l'octet précédant la copie de la FAT en mémoire centrale pour indiquer qu'elle a été modifiée depuis sa dernière sauvegarde sur la disquette, on restaure les valeurs de `BX`, `CX` et `DX`, on appelle la routine UNPACK pour ? ?, on restaure la valeur du décalage de la FAT et on place dans `DI` le contenu de l'entrée de la FAT pour le numéro d'unité d'allocation obtenue. Si cette unité d'allocation n'est pas libre, on termine la routine. Sinon on place le décalage `BX` comme début de la zone des unités d'allocation sur le FCB et on termine la routine (lignes 2436–2449).
- Si ce numéro dépasse le nombre d'unités d'allocation de la disquette, on décrémente la valeur de `AX`. Si elle est ? ?, on retourne à FINDFRE. Sinon on la place dans `BX` et on appelle la routine UNPACK pour obtenir le contenu de l'entrée de la FAT. Si elle est libre, on a trouvé une unité d'allocation libre donc on retourne à HAVFRE. Sinon on place le numéro d'unité d'allocation pointé par celle-ci dans `BX` et on retourne à FINDFRE (lignes 2396–2397 et 2422–2429).
- Si ce numéro est égal au nombre d'unités d'allocation de la disquette, on restaure la valeur de `BX`, on place `FFFh` dans `DX` et on appelle la routine RELBLKS, partie de la routine RELEASE, pour placer une fin de fichier dans la première unité d'allocation du fichier et libérer toutes ses autres unités d'allocation, on restaure le nombre d'unités d'allocation requises dans `AX`, on lui soustrait le nombre d'unités d'allocation que l'on est parvenu à lui allouer, on restaure la position de la première unité d'allocation allouée dans `DX` et le début de la copie de la FAT en mémoire centrale dans `SI`, on incrémente la valeur de `DX`, on place dans `AX` le nombre maximum d'unités d'allocation du fichier, dans `DX` le nombre de secteurs par unité d'allocation, que l'on multiplie par le contenu de `AX` pour obtenir le nombre de secteurs maximum dans le fichier, que l'on place dans `CX`, auquel on soustrait la position du secteur dans le fichier pour obtenir le nombre maximum de secteurs du fichier sur lesquels on peut écrire, que l'on met à zéro si ce nombre est négatif, on lève le drapeau CF et on a terminé la routine (lignes 2398–2417).

5.8.9 Opérations sur les enregistrements

5.8.9.1 Détermination du numéro absolu de l'enregistrement index

La routine GETREC (pour GET index RECord) renvoie dans `DX:AX` le numéro absolu de l'enregistrement index.

Avant d'appeler la routine, `DS:DX` doit pointer sur le FCB. Après son exécution, `CX` est mis à 1, pour rappeler que l'on ne parle que d'un seul enregistrement (ce qui sert pour une utilisation ultérieure), `DX:AX` contient le numéro absolu de l'enregistrement index et `DS:DI` pointe sur le FCB, plus précisément sa partie standard s'il s'agit d'un FCB étendu :

```
2335 : GETREC:
2336 :
2337 : ; Inputs:
2338 : ;       DS:DX point to FCB
2339 : ; Outputs:
2340 : ;       CX = 1
2341 : ;       DX:AX = Record number determined by EXTENT and NR fields
2342 : ;       DS:DI point to FCB
2343 : ; No other registers affected.
2344 :
2345 :         MOV     DI,DX
2346 :         CMP     BYTE PTR [DI],-1        ;Check for extended FCB
2347 :         JNZ     NORMFCB2
2348 :         ADD     DI,7
2349 : NORMFCB2:
2350 :         MOV     CX,1
2351 :         MOV     AL,[DI.NR]
2352 :         MOV     DX,[DI.EXTENT]
2353 :         SHL     AL,1
2354 :         SHR     DX,1
2355 :         RCR     AL,1
2356 :         MOV     AH,DL
2357 :         MOV     DL,DH
2358 :         MOV     DH,0
2359 :         RET
```

— S'il s'agit d'un FCB étendu, on se place au début du FCB standard associé (lignes 2345–2349).

— On place 1 dans `CX` (ligne 2350).

— On place dans `AL` le numéro d'enregistrement (0–127) en cours dans le bloc en cours : il s'agit tout simplement du contenu du champ NR du FCB (ligne 251).

— On place dans `DX` le numéro de bloc en cours : il s'agit tout simplement du contenu du champ EXTENT du FCB (ligne 252).

— On calcule le numéro absolu de l'enregistrement index à partir de ces deux données et on termine la routine (lignes 253–259) : on effectue pour cela une rotation des bits de `AL` vers la gauche, ce qui a pour effet de placer 0 dans le drapeau de rentenue CF ; on effectue une rotation des bits de `DX` vers la droite, ce qui a pour effet de placer le premier bit du numéro de bloc dans le drapeau de retenue CF ; on effectue une rotation des bits de `AL` vers la droite, ce qui permet de placer le premier bit de `DX` comme septième bit de `AL` ; on place les 8 bits de poids faible de `DX` dans `AH`, le registre `AX` contient alors les 16 bits de poids faible du numéro absolu de l'enregistrement index ; on place les 8 bits de poids fort de `DX` dans `DH`, les 8 bits de poids fort du numéro absolu de l'enregistrement index sont alors contenus dans `DL` et on place 0 dans `DH` pour que `DX` contienne les 8 bits de poids fort du numéro absolu de l'enregistrement index.

5.8.9.2 Incrémentation du numéro d'enregistrement index

La routine FINSEQ (fin d'un opération séquentielle) incrémente le numéro d'enregistrement index dans le FCB du fichier, c'est-à-dire met à jour les champs 'numéro de bloc' et 'numéro d'enregistrement dans ce bloc' de celui-ci.

Lors de l'appel de cette routine, `DX:AX` doit contenir le numéro absolu de l'enregistrement index et `ES:DI` pointer sur le FCB :

```
1404 : FINSEQ:
1405 :          JCXZ    SETNREX
1406 :          ADD     AX,1
1407 :          ADC     DX,0
1408 :          JMP     SHORT SETNREX
```

c'est-à-dire que si le drapeau ZF est levé, on incrémente `AX`, sans oublier de reporter l'éventuelle retenue dans `DX`, ce qui place le numéro absolu de l'enregistrement suivant dans `DX:AX`. Dans tous les cas on va à la partie SETNREX (SET les champs NT et EXtent du FCB) pour changer les champs adéquats du FCB :

```
1440 : SETNREX:
1441 :          MOV     CX,AX
1442 :          AND     AL,7FH
1443 :          MOV     ES:[DI.NR],AL
1444 :          AND     CL,80H
1445 :          SHL     CX,1
1446 :          RCL     DX,1
1447 :          MOV     AL,CH
1448 :          MOV     AH,DL
1449 :          MOV     ES:[DI.EXTENT],AX
1450 :          MOV     AL,CS:[DSKERR]
1451 :          RET
```

— Le registre `AX` contient le mot de poids faible du numéro de l'enregistrement index du fichier. L'octet de poids faible, contenu dans `AL`, comporte 8 bits alors que le champ NR du FCB n'en contient que 7. On le masque donc par `7Fh` pour ne retenir que les 7 premiers bits et on place le résultat dans le champ 'numéro d'enregistrement dans le bloc' du FCB (lignes 1441-1443).

— On détermine de même le numéro de bloc et l'on renseigne le champ 'numéro de bloc' du FCB (lignes 1444–1449).

— On place le code d'erreur, obtenu depuis la variable DISKERR, dans `AL` et on termine la routine (lignes 1450–1451).

La variable DSKERR contient un code d'erreur lors d'une opération sur une disquette :

```
3696 : DSKERR  DB      ?
```

5.8.9.3 Routine de préparation au transfert d'enregistrements

La routine SETUP prépare au transfert d'un certain nombre d'enregistrements.

Avant de l'appeler, `DS:DI` doit pointer sur un FCB standard, `DX:AX` contenir le numéro absolu du premier enregistrement du fichier à transférer et `CX` le nombre d'enregistrements à transférer.

Après exécution de la routine, le segment des données est confondu avec le segment de code, `ES:DI` pointe sur le FCB, `BL` contient le numéro de périphérique spécifié par le FCB, `CX` le nombre d'octets à transférer, `BP` pointe sur le bloc des paramètres du lecteur de disquette, `SI` sur la copie de la FAT en mémoire centrale, RECCNT contient le nombre d'enregistrements à transférer, RECPOS le numéro absolu du premier enregistrement à transférer, FCB l'adresse du FCB, NEXTADD l'adresse à laquelle transférer (dans la zone de transfert du disque), SECPOS le numéro du premier secteur du fichier dans lequel se trouve ce qui doit être transféré, BYTPOS le numéro dans le fichier du premier octet à transférer, BYTSECPOS la position du premier octet à transférer dans le secteur adéquat, CLUSNUM le numéro de la première unité d'allocation à transférer, SECCLUSPOS la position du secteur contenant le premier enregistrement à transférer dans cette unité d'allocation, DSKERR le code d'erreur de lecture sur le disque (`00h` s'il n'y a pas d'erreur), TRANS 0 s'il n'y a pas encore eu de transfert et THISDRIVE le numéro de lecteur de disquette (avec 0 pour A). Si SETUP ne trouve rien à transférer, `CX` contient 0.

La variable FCB repère un FCB, RECPOS contient le numéro absolu d'un enregistrement, RECCNT un nombre d'enregistrements, SECPOS une position de secteur, BYTESECPOS un numéro d'octet dans un secteur et BYTPOS un numéro d'octet :

```
3703 : FCB     DW      ?          ;Address of user FCB
[...]
3705 : RECPOS  DB      4 DUP (?)
3706 : RECCNT  DW      ?
[...]
3709 : SECPOS  DW      ?          ;Position of first sector accessed
[...]
3711 : BYTSECPOS DW    ?          ;Position of first byte within sector
3712 : BYTPOS  DB      4 DUP (?)  ;Byte position in file of access
```

La routine est SETUP :

```
1465 : NOFILERR:
1466 :         XOR     CX,CX
1467 :         MOV     BYTE PTR [DSKERR],4
1468 :         POP     BX
1469 :         RET
1470 :
1471 : SETUP:
1472 :
1473 : ; Inputs:
1474 : ;       DS:DI point to FCB
1475 : ;       DX:AX = Record position in file of disk transfer
1476 : ;       CX = Record count
1477 : ; Outputs:
1478 : ;       DS = CS
1479 : ;       ES:DI point to FCB
1480 : ;       BL = DEVID from FCB
1481 : ;       CX = No. of bytes to transfer
1482 : ;       BP = Base of drive parameters
1483 : ;       SI = FAT pointer
1484 : ;       [RECCNT] = Record count
1485 : ;       [RECPOS] = Record position in file
1486 : ;       [FCB] = DI
1487 : ;       [NEXTADD] = Displacement of disk transfer within segment
1488 : ;       [SECPOS] = Position of first sector
```

```
1489 : ;       [BYTPOS] = Byte position in file
1490 : ;       [BYTSECPOS] = Byte position in first sector
1491 : ;       [CLUSNUM] = First cluster
1492 : ;       [SECCLUSPOS] = Sector within first cluster
1493 : ;       [DSKERR] = 0 (no errors yet)
1494 : ;       [TRANS] = 0 (No transfers yet)
1495 : ;       [THISDRV] = Physical drive unit number
1496 : ; If SETUP detects no records will be transfered, it returns 1 level up
1497 : ; with CX = 0.
1498 :
1499 :         PUSH    AX
1500 :         MOV     AL,[DI]
1501 :         DEC     AL
1502 :         MOV     CS:[THISDRV],AL
1503 :         MOV     AL,[DI.DEVID]
1504 :         MOV     SI,[DI.RECSIZ]
1505 :         OR      SI,SI
1506 :         JNZ     HAVRECSIZ
1507 :         MOV     SI,128
1508 :         MOV     [DI.RECSIZ],SI
1509 : HAVRECSIZ:
1510 :         PUSH    DS
1511 :         POP     ES                      ;Set ES to DS
1512 :         PUSH    CS
1513 :         POP     DS                      ;Set DS to CS
1514 :         OR      AL,AL                    ;Is it a device?
1515 :         JNS     NOTDEVICE
1516 :         MOV     AL,0                    ;Fake in drive 0 so we can get SP
1517 : NOTDEVICE:
1518 :         CALL    GETBP
1519 :         POP     AX
1520 :         JC      NOFILERR
1521 :         CMP     SI,64                   ;Check if highest byte of RECPOS is significant
1522 :         JB      SMALREC
1523 :         MOV     DH,0                    ;Ignore MSB if record >= 64 bytes
1524 : SMALREC:
1525 :         MOV     [RECCNT],CX
1526 :         MOV     WORD PTR [RECPOS],AX
1527 :         MOV     WORD PTR [RECPOS+2],DX
1528 :         MOV     [FCB],DI
1529 :         MOV     BX,[DMAADD]
1530 :         MOV     [NEXTADD],BX
1531 :         MOV     BYTE PTR [DSKERR],0
1532 :         MOV     BYTE PTR [TRANS],0
1533 :         MOV     BX,DX
1534 :         MUL     SI
1535 :         MOV     WORD PTR [BYTPOS],AX
1536 :         PUSH    DX
1537 :         MOV     AX,BX
1538 :         MUL     SI
1539 :         POP     BX
1540 :         ADD     AX,BX
1541 :         ADC     DX,0                    ;Ripple carry
1542 :         JNZ     EOFERR
1543 :         MOV     WORD PTR [BYTPOS+2],AX
1544 :         MOV     DX,AX
1545 :         MOV     AX,WORD PTR [BYTPOS]
1546 :         MOV     BX,[BP.SECSIZ]
1547 :         CMP     DX,BX                   ;See if divide will overflow
1548 :         JNC     EOFERR
1549 :         DIV     BX
1550 :         MOV     [SECPOS],AX
```

```
1551 :          MOV     [BYTSECPOS],DX
1552 :          MOV     DX,AX
1553 :          AND     AL,[BP.CLUSMSK]
1554 :          MOV     [SECCLUSPOS],AL
1555 :          MOV     AX,CX                   ;Record count
1556 :          MOV     CL,[BP.CLUSSHFT]
1557 :          SHR     DX,CL
1558 :          MOV     [CLUSNUM],DX
1559 :          MUL     SI                      ;Multiply by bytes per record
1560 :          MOV     CX,AX
1561 :          ADD     AX,[DMAADD]             ;See if it will fit in one segment
1562 :          ADC     DX,0
1563 :          JZ      OK                      ;Must be less than 64K
1564 :          MOV     AX,[DMAADD]
1565 :          NEG     AX                      ;Amount of room left in segment
1566 :          JNZ     PARTSEG                 ;All 64K available?
1567 :          DEC     AX                       ;If so, reduce by one
1568 : PARTSEG:
1569 :          XOR     DX,DX
1570 :          DIV     SI                      ;How many records will fit?
1571 :          MOV     [RECCNT],AX
1572 :          MUL     SI                      ;Translate that back into bytes
1573 :          MOV     BYTE PTR [DSKERR],2     ;Flag that trimming took place
1574 :          MOV     CX,AX
1575 :          JCXZ    NOROOM
1576 : OK:
1577 :          MOV     BL,ES:[DI.DEVID]
1578 :          MOV     SI,[BP.FAT]
1579 :          RET
1580 :
1581 : EOFERR:
1582 :          MOV     BYTE PTR [DSKERR],1
1583 :          XOR     CX,CX
1584 : NOROOM:
1585 :          POP     BX                      ;Kill return address
1586 :          RET
```

— On sauvegarde momentanément le contenu de `AX` sur la pile, on récupère le numéro du périphérique à partir du FCB (avec 1 pour A) dans `AL`, on le décrémente (pour obtenir 0 pour A) et on le place dans la variable THISDRIV.

— On place dans `AL` le numéro de périphérique (avec 1 comme septième bit pour un fichier d'entrées-sorties, 1 comme sixième bit si le fichier n'a pas été enregistré sur la disquette depuis la dernière modification) tel que le spécifie le FCB (ligne 1503).

— On place dans `SI` la taille d'un enregistrement du fichier, celle spécifiée par le FCB si elle est non nulle, la valeur 128 par défaut sinon (lignes 1504–1509).

— On confond le segment supplémentaire avec le segment des données en cours et on prend comme nouveau segment des données le segment de code (lignes 1510–1513).

— S'il s'agit d'un fichier de périphérique, on place 0 dans `AL` (lignes 1514–1517).

— On appelle la routine GETBP pour faire pointer `BP` sur le bloc des paramètres du lecteur de disquette (ligne 1518).

— On sauvegarde sur la pile le contenu de `AX`. Si on n'a pas trouvé le fichier, on place 0 dans `CX` pour indiquer que la routine n'a rien trouvé à transférer, le code d'erreur `04h` dans DSKERR, les 16 bits de poids faible du numéro du premier enregistrement à transférer dans `BX` et on termine la routine (lignes 1519–1520 et 1465–1469).

— Si la taille d'un enregistrement est supérieur à 64 octets, on peut ne pas tenir compte de l'octet de poids fort du numéro du premier enregistrement du fichier à transférer (lignes

1521–1524).

— On place le nombre d'enregistrements à transférer dans la variable RECCNT, le numéro absolu du premier enregistrement à transférer dans la variable RECPOS et l'adresse du FCB dans la variable FCB (lignes 1525–1528).

— On initialise la variable NEXTADD avec l'adresse du début de la zone de transfert de la disquette (lignes 1529 et 1530).

— On place le code d'erreur `00h` dans la variable DSKERR, pour indiquer qu'il n'y a pas d'erreur pour l'instant, et 0 dans la variable TRANS pour indiquer qu'il n'y a pas encore eu de transfert (lignes 1531 et 1532).

— On sauvegarde le mot (en fait l'octet) de poids fort du numéro du premier enregistrement à transférer dans `BX`, puisque le contenu de `DX` va être détruit par la multiplication que l'on va effectuer. À ce moment `AX` contient le mot de poids faible du numéro du premier enregistrement à transférer. On le multiplie par la taille d'un enregistrement, pour obtenir, entre autre, le mot de poids faible du numéro du premier octet à transférer, que l'on place dans le mot de poids faible de la variable BYTPOS (lignes 1533–1535).

— On sauvegarde sur la pile l'octet de poids fort du résultat de la multiplication précédente, on replace le mot (l'octet dans les faits) de poids fort du numéro du premier enregistrement à transférer, on le multiplie par la taille d'un enregistrement et on lui ajoute l'octet de poids fort du résultat de la multiplication antérieure, obtenant ainsi dans `DX:AX` le numéro du premier octet à transférer ((lignes 1536–1541).

— Si on a une retenue, c'est qu'il n'y a pas suffisamment de place. On place donc `01h` comme code d'erreur dans DSKERR pour indiquer que le transfert n'a pas pu avoir lieu, 0 dans `CX` pour la même raison, on récupère l'adresse de retour sur la pile et on termine la routine (lignes 1542 et 1581–1586).

  Sinon on place la valeur obtenue dans le mot de poids fort de la variable BYTPOS.

— On place dans `DX:AX` le numéro du premier octet du fichier à transférer. La division par la taille d'un secteur donne le numéro de secteur du fichier dans lequel il se trouve.

  Si le résultat ne tient pas sur un octet, c'est qu'il n'y a pas suffisamment de place. On place donc `01h` comme code d'erreur dans DSKERR pour indiquer que le transfert n'a pas pu avoir lieu, 0 dans `CX` pour la même raison, on récupère l'adresse de retour sur la pile et on termine la routine

  Sinon on place ce numéro de secteur dans la variable SECPOS (lignes 1543–1550).

— On place dans BYTSECPOS la position dans ce secteur du premier octet à transférer, qui est le reste dans la division euclidienne précédente (ligne 1551).

— On place le numéro de secteur dans `DX` et la position du secteur dans l'unité d'allocation correspondante dans la variable SECPLUSPOS (lignes 1552–1554).

— On place le nombre d'enregistrements à transférer dans `AX`, le nombre de secteurs d'une unité d'allocation dans `CL`, on décale `DX` d'autant fois ce nombre à droite, ce qui donne le numéro de l'unité d'allocation dans laquelle se trouve le premier enregistrement à transférer, que l'on place dans la variable CLUSNUM (lignes 1556–1558).

— On multiplie le nombre d'enregistrements, placé ci-dessus dans `AX`, par le nombre d'octets par enregistrement, que l'on place dans `CX` et que l'on ajoute à l'adresse du début de la zone de transfert de la disquette pour voir si cela va tenir dans le segment, c'est-à-dire s'il est inférieur à 64 kiO (lignes 1555 et 1559–1561).

  Si ce n'est pas le cas, on place l'adresse du début de la zone de transfert de la disquette dans `AX`, que l'on inverse pour obtenir l'espace disponible dans le segment, que l'on réduit de 1 si les 64 kiO sont disponibles. On place 0 dans `DX` et on détermine le nombre d'enregistrements nécessaires, que l'on place dans la variable RECCNT. On calcule également

le nombre d'octets correspondants, que l'on place dans `CX`. On place le code d'erreur `02h` dans la variable DSKERR pour indiquer qu'il n'y avait pas suffisamment de place dans la zone de transfert pour effectuer le transfert des enregistrements, on récupère l'adresse de retour sur la pile et on termine la routine (lignes 1563–1575 et 1584–1586).

— Sinon on place le numéro de périphérique spécifié par le FCB dans `BL`, le décalage de la copie de la FAT en mémoire centrale dans `SI` et on termine la routine (lignes 1563 et 1576–1579).

5.8.9.4 Mise à jour du FCB après une lecture séquentielle

Après une lecture séquentielle, la routine SETFCB met à jour les deux champs 'bloc en cours' et 'enregistrement en cours dans ce bloc' du FCB :

```
1955 : SETFCB:
1956 :          MOV     SI,[FCB]
1957 :          MOV     AX,[NEXTADD]
1958 :          MOV     DI,AX
1959 :          SUB     AX,[DMAADD]             ;Number of bytes transfered
1960 :          XOR     DX,DX
1961 :          MOV     CX,ES:[SI.RECSIZ]
1962 :          DIV     CX                      ;Number of records
1963 :          CMP     AX,[RECCNT]             ;Check if all records transferred
1964 :          JZ      FULLREC
1965 :          MOV     BYTE PTR [DSKERR],1
1966 :          OR      DX,DX
1967 :          JZ      FULLREC                 ;If remainder 0, then full record transfered
1968 :          MOV     BYTE PTR [DSKERR],3     ;Flag partial last record
1969 :          SUB     CX,DX                   ;Bytes left in last record
1970 :          PUSH    ES
1971 :          MOV     ES,[DMAADD+2]
1972 :          XCHG    AX,BX                   ;Save the record count temporarily
1973 :          XOR     AX,AX                   ;Fill with zeros
1974 :          SHR     CX,1
1975 :          JNC     EVENFIL
1976 :          STOSB
1977 : EVENFIL:
1978 :          REP     STOSW
1979 :          XCHG    AX,BX                   ;Restore record count to AX
1980 :          POP     ES
1981 :          INC     AX                      ;Add last (partial) record to total
1982 : FULLREC:
1983 :          MOV     CX,AX
1984 :          MOV     DI,SI                   ;ES:DI point to FCB
1985 : SETCLUS:
1986 :          MOV     AX,[CLUSNUM]
1987 :          MOV     ES:[DI.LSTCLUS],AX
1988 :          MOV     AX,[LASTPOS]
1989 :          MOV     ES:[DI.CLUSPOS],AX
1990 : ADDREC:
1991 :          MOV     AX,WORD PTR [RECPOS]
1992 :          MOV     DX,WORD PTR [RECPOS+2]
1993 :          JCXZ    RET28                   ;If no records read, don't change position
1994 :          DEC     CX
1995 :          ADD     AX,CX                   ;Update current record position
1996 :          ADC     DX,0
1997 :          INC     CX
1998 : RET28:   RET
```

— On fait pointer `SI` sur le FCB en cours, `DI` sur le tampon de transfert, on place le nombre d'octets déjà transférés dans `AX`, calculé en soustrayant à l'adresse en cours dans le tampon de transfert l'adresse du début du tampon, 0 dans `DX`, pour se préparer à la division qui suit, et la taille d'un enregistrement dans `CX`. En divisant `DX:AX` par `CX` on obtient dans `AX` le nombre d'enregistrements complets transférés et dans `DX` le nombre d'octets restants (lignes 1955–1962).

— Si ce nombre n'est pas égal au numéro d'enregistrement voulu, on place le code d'erreur `01h` dans DSKERR et on termine la routine (ligne 1963–1967).

— Si `DX` est non nul, on place le code d'erreur `03h` dans DSKERR pour indiquer qu'il ne reste plus qu'un enregistrement partiel à transférer, le dernier, on place le nombre d'octets disponibles dans ce dernier enregistrement dans `CX`, on sauvegarde sur la pile l'adresse du segment supplémentaire, on place l'adresse du segment de transfert dans `ES`, on sauvegarde momentanément le nombre d'enregistrements dans `BX` et on remplit la fin de l'enregistrement avec des zéros (lignes 1968–1978).

— On replace le nombre d'enregistrements dans `AX`, on restaure la valeur de `ES`, on ajoute le dernier enregistrement au total, on place ce nombre dans `CX`, on fait pointer `ES:DI` sur le FCB, on renseigne les champs 'bloc en cours' et 'enregistrement en cours dans ce bloc' du FCB, (lignes 1979–1989).

— Si on n'a pas eu besoin de lire un nouvel enregistrement, on ne change pas la position. Sinon, on décrémente `CX` que l'on ajoute à `AX`, on ajoute la retenue à `DX`, on incrémente `CX` et on termine la routine (lignes 1990–1998).

5.8.10 Routine de service de la fonction 20 (14h) de lecture séquentielle dans un fichier spécifié par un FCB

5.8.10.1 La routine de service

La routine de service de la fonction 20 de l'interruption `21h`, de lecture séquentielle dans un fichier spécifié par un FCB, est repérée par l'étiquette SEQRD :

```
1396 : SEQRD:  ;System call 20
1397 :         CALL    GETREC
1398 :         CALL    LOAD
1399 :         JMP     SHORT FINSEQ
```

— `DS:DX` pointant sur un FCB ouvert, on appelle la routine GETREC pour placer dans `DX:AX` le numéro absolu de l'enregistrement index d'opération séquentielle.

— On appelle ensuite la routine LOAD, étudiée ci-dessous, pour transférer cet enregistrement dans la zone de transfert.

— Puis on passe à la partie FINSEQ pour incrémenter le numéro d'enregistrement index dans le FCB.

5.8.10.2 Routine de chargement d'enregistrements d'un fichier en mémoire centrale

Objet.- La routine LOAD permet de transférer des enregistrements d'un fichier depuis la disquette dans la zone de transfert en mémoire centrale.

Avant d'appeler la routine, `DS:DI` doit pointer sur un FCB standard, `DX:AX` contenir le numéro absolu du premier enregistrement à lire dans le fichier et `CX` le nombre d'enregistrements à lire.

Après exécution de la routine, `DX:AX` donne le numéro absolu du dernier enregistrement lu, `CX` le nombre d'octets lus, `ES:DI` pointe sur le FCB et les champs 'numéro de bloc' et 'numéro de l'enregistrement dans ce bloc' du FCB ont été mis à jour.

Les problèmes à résoudre.- La taille par défaut d'un enregistrement est de 128 bits. L'opération de chargement est assez facile pour cette taille : pour le premier enregistrement, on charge le premier secteur et on se place au début de la zone de transfert ; pour le second enregistrement, on n'a pas besoin de charger à nouveau le premier secteur, il suffit de déplacer l'adresse de lecture dans la zone de transfert ; pour le troisième enregistrement, par contre, il faut charger le second secteur, en détruisant ou non la copie du premier secteur dans la zone de transfert ; et ainsi de suite.

Prenons par contre une taille d'enregistrement de 670 octets, par exemple :

- Pour le premier enregistrement, il faut charger deux secteurs (pas nécessairement consécutifs sur la disquette) dans la zone de tranfert et se placer au début de la zone de transfert pour en commencer la lecture.
- Pour le second enregistrement, on n'a plus besoin du premier secteur mais il faut charger un secteur supplémentaire et se placer au décalage 670 de la zone de transfert si on n'a pas détruit la copie du premier secteur, au décalage 670 - 512 = 158 sinon.

Dans le cas général, on devra garder au moins le dernier secteur chargé (sauf dans le cas particulier où le dernier enregistrement arrivait à la fin de celui-ci), se placer à un certain décalage de ce secteur, charger quelques secteurs suivants du fichier, non nécessairement consécutifs sur la disquette, et enfin charger un dernier secteur qui ne sera rempli que partiellement par l'enregistrement.

Code commenté.- La routine LOAD est précédée de la sous-routine RDERR :

```
1859 : RDERR:
1860 :          XOR     CX,CX
1861 :          JMP     WRTERR
1862 :
1863 : RDLASTJ:JMP     RDLAST
1864 :
1865 : LOAD:
1866 :
1867 : ; Inputs:
1868 : ;        DS:DI point to FCB
1869 : ;        DX:AX = Position in file to read
1870 : ;        CX = No. of records to read
1871 : ; Outputs:
1872 : ;        DX:AX = Position of last record read
1873 : ;        CX = No. of bytes read
1874 : ;        ES:DI point to FCB
1875 : ;        LSTCLUS, CLUSPOS fields in FCB set
1876 :
1877 :          CALL    SETUP
1878 :          OR      BL,BL                   ;Check for named device I/O
1879 :          JS      READDEV
1880 :          MOV     AX,ES:WORD PTR [DI.FILSIZ]
1881 :          MOV     BX,ES:WORD PTR [DI.FILSIZ+2]
```

```
1882 :          SUB     AX,WORD PTR [BYTPOS]
1883 :          SBB     BX,WORD PTR [BYTPOS+2]
1884 :          JB      RDERR
1885 :          JNZ     ENUF
1886 :          OR      AX,AX
1887 :          JZ      RDERR
1888 :          CMP     AX,CX
1889 :          JAE     ENUF
1890 :          MOV     CX,AX
1891 : ENUF:
1892 :          CALL    BREAKDOWN
1893 :          MOV     CX,[CLUSNUM]
1894 :          CALL    FNDCLUS
1895 :          OR      CX,CX
1896 :          JNZ     RDERR
1897 :          MOV     [LASTPOS],DX
1898 :          MOV     [CLUSNUM],BX
1899 :          CMP     [BYTCNT1],0
1900 :          JZ      RDMID
1901 :          CALL    BUFRD
1902 : RDMID:
1903 :          CMP     [SECCNT],0
1904 :          JZ      RDLASTJ
1905 :          CALL    NEXTSEC
1906 :          JC      SETFCB
1907 :          MOV     BYTE PTR [TRANS],1      ;A transfer is taking place
1908 : ONSEC:
1909 :          MOV     DL,[SECCLUSPOS]
1910 :          MOV     CX,[SECCNT]
1911 :          MOV     BX,[CLUSNUM]
1912 : RDLP:
1913 :          CALL    OPTIMIZE
1914 :          PUSH    DI
1915 :          PUSH    AX
1916 :          PUSH    DS
1917 :          MOV     DS,[DMAADD+2]
1918 :          PUSH    DX
1919 :          PUSH    BX
1920 :          PUSHF                           ;Save carry flag
1921 :          CALL DREAD
1922 :          POPF                            ;Restore carry flag
1923 :          POP     DI                      ;Initial transfer address
1924 :          POP     AX                      ;First sector transfered
1925 :          POP     DS
1926 :          JC      NOTBUFFED               ;Was one of those sectors in the buffer?
1927 :          CMP     BYTE PTR [DIRTYBUF],0   ;Is buffer dirty?
1928 :          JZ      NOTBUFFED               ;If not no problem
1929 : ;We have transfered in a sector from disk when a dirty copy of it is in the buffer.
1930 : ;We must transfer the sector from the buffer to correct memory address
1931 :          SUB     AX,[BUFSECNO]           ;How many sectors into the transfer?
1932 :          NEG     AX
1933 :          MOV     CX,[BP.SECSIZ]
1934 :          MUL     CX                      ;How many bytes into the transfer?
1935 :          ADD     DI,AX
1936 :          MOV     SI,[BUFFER]
1937 :          PUSH    ES
1938 :          MOV     ES,[DMAADD+2]           ;Get disk transfer segment
1939 :          SHR     CX,1
1940 :          REP     MOVSW
1941 :          JNC     EVENMOV
1942 :          MOVSB
1943 : EVENMOV:
```

```
1944 :          POP     ES
1945 : NOTBUFFED:
1946 :          POP     CX
1947 :          POP     BX
1948 :          JCXZ    RDLAST
1949 :          CMP     BX,0FF8H
1950 :          JAE     SETFCB
1951 :          MOV     DL,0
1952 :          INC     [LASTPOS]               ;We'll be using next cluster
1953 :          JMP     SHORT RDLP
```

— On appelle la routine SETUP de préparation au chargement (ligne 1877).

Avant d'appeler cette routine, `DS:DI` doit pointer sur un FCB standard, `DX:AX` contenir le numéro absolu du premier enregistrement du fichier à transférer et `CX` le nombre d'enregistrements à transférer, ce qui est le cas.

Après exécution de la routine, le segment des données est confondu avec le segment de code, `ES:DI` pointe sur le FCB, `BL` contient le numéro de lecteur de disquette donné par le FCB, `CX` le nombre d'octets à transférer, `BP` pointe sur le bloc des paramètres du lecteur de disquette, `SI` sur la copie de la FAT en mémoire centrale, la variable RECCNT contient le nombre d'enregistrements à transférer, RECPOS le numéro absolu du premier enregistrement à transférer, FCB pointe sur le FCB, NEXTADD contient l'adresse dans le tampon de transfert du disque à laquelle on est parvenu, SECPOS le numéro du premier secteur du fichier dans lequel se trouve le début de l'enregistrement à transférer, BYTPOS le numéro dans le fichier du premier octet à transférer, BYTSECPOS la position du premier octet à transférer dans le secteur adéquat, CLUSNUM la position dans le fichier de l'unité d'allocation contenant ce secteur, SECCLUSPOS la position de ce secteur dans cette unité d'allocation, DSKERR le code d'erreur de lecture sur le disque (`00h` s'il n'y a pas d'erreur), TRANS indique si on est en train ou non d'effectuer un transfert (avec respectivement pour valeurs `01h` et `00h`) et THISDRIV le numéro de lecteur de disquette (avec 0 pour A). Si SETUP ne trouve rien à transférer, `CX` contient 0.

Cas d'un fichier de périphérique

— Si le nom du fichier est celui d'un périphérique, on est renvoyé à la partie READDEV, étudiée ci-dessous, de lecture sur un tel fichier (lignes 1878–1879).

Enregistrement non vide

— Sinon on place dans `BX:AX` la taille du fichier moins la position du premier octet à transférer, c'est-à-dire le nombre d'octets du fichier non encore transférés (lignes 1880–1883).

Si on obtient un nombre négatif ou si le contenu de `AX` est nul, c'est qu'il n'y avait pas de données dans l'enregistrement. On place donc 0 dans `CX`, pour indiquer que rien n'a été transféré, et on va à la partie WRTERR pour terminer la routine avec le code d'erreur `01h` (lignes 1884–1887 et 1859–1861).

Une erreur d'écriture est traitée par la partie WRTERR :

```
2054 : WRTERR:
2055 :          MOV     BYTE PTR [DSKERR],1
2056 : LVDSK:
2057 :          MOV     AX,WORD PTR [RECPOS]
2058 :          MOV     DX,WORD PTR [RECPOS+2]
2059 :          MOV     DI,[FCB]
2060 :          RET
```

qui consiste à placer le code d'erreur `01h` dans la variable DSKERR, la position du premier octet à transférer dans `DX:AX`, à faire pointer `DI` sur le FCB et à terminer la routine.

— Si le contenu de `AX` est plus petit que `CX`, on place le contenu de `AX` dans `CX` (lignes 1888–1891).

Découpage de l'enregistrement

— Comme nous l'avons vu ci-dessus, l'enregistrement index s'étend, dans le cas général, sur la fin d'un secteur déjà chargé, sur un certain nombre de secteurs, à charger complètement, et sur le début d'un dernier secteur, également à charger. On appelle la routine BREAKDOWN, étudiée ci-dessous, pour effectuer ce découpage, plus précisément pour placer dans la variable BYTCNT1 le nombre d'octets à lire dans le premier secteur (déjà chargé), dans SECCNT le nombre de secteurs entiers à transférer et à lire complètement et, enfin, dans BYTCNT2 le nombre d'octets à lire au début du dernier secteur à charger (ligne 1892).

La variable SECCNT contient le nombre de secteurs entiers à transférer et à lire complètement, BYTCNT2 le nombre d'octets à lire au début du dernier secteur à charger :

```
3714 : BYTCNT2 DW      ?       ;No. of bytes in last sector
3715 : SECCNT  DW      ?       ;No. of whole sectors
```

— On place la position dans le fichier de la première unité d'allocation à transférer, contenue dans la variable CLUSNUM, qui a été renseignée par la routine SETUP, dans `CX` et on appelle la routine FNDCLUS pour voir si elle existe dans le fichier (lignes 1893–1894).

Avant d'appeler cette routine, le segment des données doit être confondu avec le segment de code, `CX` contenir la position n dans le fichier de l'unité d'allocation dont on veut vérifier l'existence, `BP` pointer sur le bloc des paramètres du lecteur de la disquette sur laquelle se trouve le fichier, `SI` sur la copie de la FAT en mémoire centrale et `ES:DI` sur le FCB du fichier, ce qui est le cas.

Après son exécution, `BX` contient le numéro de l'unité d'allocation atteinte, `CX` le nombre d'unités d'allocation supplémentaires que le fichier aurait dû contenir pour atteindre celle de position n et `DX` la position dans le fichier de l'unité d'allocation à laquelle on est parvenu, soit $min(n, d)$.

— Si le nombre d'unités d'allocation supplémentaires que le fichier aurait dû contenir pour atteindre celle de position le nombre d'enregistrements à lire est non nul, on termine la routine sur une erreur (lignes 1895–1896 et 1859–1860).

— Sinon on place la position dans le fichier de l'unité d'allocation voulue dans la variable LASTPOS et le numéro sur la disquette de celle-ci dans la variable CLUSNUM (lignes 1897–1898).

Lecture de la fin du premier secteur

— S'il y a des octets à lire dans le secteur déjà chargé, on appelle la routine BUFRD, étudiée ci-dessous, pour les lire dans le tampon et les placer dans la zone de transfert en mémoire centrale (lignes 1899-1902).

Lecture des secteurs entiers

— S'il y a des secteurs entiers à lire, on appelle la routine NEXTSEC pour déterminer le numéro dans le fichier du secteur suivant, ainsi qu'éventuellement celui de l'unité d'allocation si on en se trouvait au dernier secteur de l'unité d'allocation se trouvant dans la zone de transfert. Si on n'en trouve pas, on termine par la partie SETFCB pour mettre à jour les champs du FCB concernant la recherche séquentielle (lignes 1902–1906).

— Sinon on initialise la lecture : on place 1 dans la variable TRANS pour indiquer qu'un transfert est en train de s'effectuer, la position du premier secteur à lire dans cette unité d'allocation dans `DL`, le nombre de secteurs à lire dans `CX`, la position de l'unité d'allocation dans le fichier dans `BX`.

On va optimiser le nombre d'opérations de lecture à effectuer sur le disque en lisant en une seule fois tous les secteurs physiquement consécutifs. On entre donc dans une boucle tant qu'il reste des secteurs à lire.

— La première instruction du corps de cette boucle consiste à appeler la routine OPTIMIZE, étudiée ci-dessous, pour déterminer le nombre de secteurs physiquement consécutifs que l'on peut transférer de la disquette vers la zone de transfert en une seule opération de lecture (lignes 1907–1913).

Avant d'appeler la routine OPTIMIZE, le segment des données et le segment de code doivent être confondus, `BX` doit contenir le numéro de l'unité d'allocation, `CX` le nombre de secteurs à transférer, `DL` la position du secteur dans l'unité d'allocation, le registre `BP` doit pointer sur le bloc des paramètres du lecteur de disquette et NEXTADD contenir l'adresse dans la zone de transfert à laquelle nous sommes parvenus, ce qui est le cas.

Après exécution de la routine, `AX` contient le nombre de secteurs qu'il restera à charger après l'opération de lecture en une seule fois sur la disquette, `BX` l'adresse de transfert, `CX` le nombre de secteurs à transférer lors des opérations suivantes de lecture sur la disquette, `DX` l'adresse du secteur, `DI` le numéro d'unité d'allocation suivante du fichier (il y a des unités d'allocation intermédiaires sur la disquette entre l'unité d'allocation atteinte et celle-ci), CLUSNUM le numéro de l'unité d'allocation atteinte, NEXTADD est mis à jour et le drapeau CF est baissé si le premier secteur qui devait être transféré se trouvait déjà dans le tampon.

— On sauvegarde sur la pile les valeurs de `DI`, `AX` et `DS`, on place l'adresse de segment de la zone de transfert dans `DS`, on sauvegarde sur la pile les valeurs de `DX`, `BX` et du registre des drapeaux et on appelle DREAD pour lire le plus grand nombre de secteurs possible en une seule opération de lecture sur la disquette et les placer dans la zone de transfert (lignes 1914–1921).

— On restaure la valeur du registre des drapeaux, on fait pointer `DI` sur le début de la zone de transfert, on place dans `AX` le numéro du premier secteur transféré et on restaure la valeur de `DS` (lignes 1922–1925).

— Si on a transféré un secteur qui se trouvait déjà dans la zone de transfert et qu'il n'avait pas été modifié en mémoire centrale, on doit le placer à l'emplacement correct dans cette zone.

Pour cela on place dans `AX` le nombre de secteurs transférés, que l'on multiplie par la taille d'un secteur pour obtenir le nombre d'octets transférés, nombre que l'on ajoute à la valeur de `DI` pour obtenir l'adresse du dernier octet transféré, on fit pointer `SI` sur le début du tampon, on sauvegarde la valeur du segment supplémentaire sur la pile, que l'on remplace par celle du segment dans lequel se trouve le tampon, on divise la valeur de `CX` par 2, puisqu'on va copier mot par mot et non octet par octet, et on reporte le contenu des secteurs transférés depuis l'endroit où on en était dans le tampon, si le nombre d'octets était impair on copie un octet de plus, et, enfin, on restaure la valeur du segment supplémentaire (lignes 1926–1944).

— On restaure les valeurs de `CX` et de `BX`. Si `CX` est nul, on va à la partie « copie partielle du dernier secteur ». Sinon si `BX` est plus petit que `FF8h`, on va à la partie « mise à jour des champs du FCB relatifs à la lecture séquentielle ». Dans le dernier cas, on place 0 dans `DL` pour indiquer qu'il faudra utiliser le premier secteur de la nouvelle unité d'allocation, on incrémente le numéro de la dernière unité d'allocation du fichier utilisée en lecture séquentielle et on retourne au début du corps de la boucle pour lire d'autres secteurs physiquement consécutifs (lignes 1945–1953).

Lecture du début du dernier secteur

— Une fois lus tous les secteurs entiers, on appelle la routine RDLAST, étudiée ci-dessous, pour lire le début du dernier secteur à lire (lignes 1902–1904, 1948 et 1863).

5.8.10.3 Routine de lecture sur un fichier de périphérique

La routine de lecture sur un fichier de périphérique est READDEV :

```
1817 : READDEV:
1818 :          PUSH    ES
1819 :          LES     DI,DWORD PTR [DMAADD]
1820 :          INC     BL
1821 :          JZ      READCON
1822 :          INC     BL
1823 :          JNZ     ENDRDDEV
1824 : READAUX:
1825 :          CALL    AUXIN
1826 :          STOSB
1827 :          CMP     AL,1AH
1828 :          LOOPNZ  READAUX
1829 :          JMP     SHORT ENDRDDEV
1830 :
1831 : READCON:
1832 :          PUSH    CS
1833 :          POP     DS
1834 :          MOV     SI,[CONTPOS]
1835 :          OR      SI,SI
1836 :          JNZ     TRANBUF
1837 :          CMP     BYTE PTR [CONBUF],128
1838 :          JZ      GETBUF
1839 :          MOV     WORD PTR [CONBUF],0FF80H        ;Set up 128-byte buffer with no template
1840 : GETBUF:
1841 :          PUSH    CX
1842 :          PUSH    ES
1843 :          PUSH    DI
1844 :          MOV     DX,OFFSET DOSGROUP:CONBUF
1845 :          CALL    BUFIN                           ;Get input buffer
1846 :          POP     DI
1847 :          POP     ES
1848 :          POP     CX
1849 :          MOV     SI,2 + OFFSET DOSGROUP:CONBUF
1850 :          CMP     BYTE PTR [SI],1AH               ;Check for Ctrl-Z in first character
1851 :          JNZ     TRANBUF
1852 :          MOV     AL,1AH
1853 :          STOSB
1854 :          MOV     AL,10
1855 :          CALL    OUT                             ;Send linefeed
1856 :          XOR     SI,SI
1857 :          JMP     SHORT ENDRDCON
```

— On sauvegarde le contenu de `ES` sur la pile et on fait pointer `DI` sur le début de la zone de transfert (lignes 1818–1819).

— Si le numéro de fichier est `FFh`, le périphérique est le clavier. On confond alors le segment des données avec le segment de code (lignes 1820–1821 et 1831–1833).

— Si le contenu de la variable CONTPOS est non nul, on continue avec TRANBUF (lignes 1834–1836).

La variable CONTPOS est déclarée ligne 3653 :

```
3653 : CONTPOS DW      0
```

— Si, par contre, le contenu de la variable CONTPOS est nul :

  — Si le contenu de la variable CONBUF est différent de 128, on y place 128 et on place `FFh` à l'octet suivant pour indiquer qu'il n'y a pas de patron (lignes 1837–1840).

La variable CONBUF repère 131 caractères, servant de tampon de la console :

```
3679 : CONBUF  DB      131 DUP (?)            ;The rest of INBUF and console buffer
```

- On sauvegarde sur la pile le contenu des registres `CX`, `ES`, `DI`, on place l'adresse de CONBUF dans `DX` et on fait appel à la routine BUFIN de saisie d'un caractère au clavier, à placer dans le tampon CONBUF et en en attendant un si besoin est, puis on restaure les valeurs de `DI`,`ES` et `CX` (lignes 1840–1848).
- Si le premier caractère de CONBUF n'est pas `1Ah` (CTRL-Z), indication d'une fin d'un fichier en MS-DOS, on continue à TRANBUF (lignes 1849–1850).
- Sinon on y place CTRL-Z, indication de fin de fichier en MS-DOS, on envoie un saut de ligne en écho à l'ecran, on place 0 dans `SI` et on va à ENDRDCON déjà étudié (lignes 1852–1857).

— Si le numéro de fichier est non nul, on va à ENDRDDEV (lignes 1822–1823), qui vient d'être étudié (commençant à la ligne 1808).

— Le dernier cas est la lecture sur le périphérique auxiliaire. Dans ce cas, on appelle AUXIN pour obtenir un caractère de celui-ci, en attendant qu'il y en ait un si nécessaire, que l'on stocke dans le tampon et on recommence tant qu'on n'a pas obtenu le caractère `1Ah`. Sinon on va à ENDRDDEV (lignes 1825–1829).

5.8.10.4 Transfert d'un enregistrement dans le tampon en mémoire centrale

La partie TRANBUF permet de placer un enregistrement dans le tampon de transfert en mémoire centrale :

```
1791 : TRANBUF:
1792 :          LODSB
1793 :          STOSB
1794 :          CMP     AL,13                           ;Check for carriage return
1795 :          JNZ     NORMCH
1796 :          MOV     BYTE PTR [SI],10
1797 : NORMCH:
1798 :          CMP     AL,10
1799 :          LOOPNZ  TRANBUF
1800 :          JNZ     ENDRDCON
1801 :          CALL    OUT                             ;Transmit linefeed
1802 :          XOR     SI,SI
1803 :          OR      CX,CX
1804 :          JNZ     GETBUF
1805 :          OR      AL,1                            ;Clear zero flag--not end of file
1806 : ENDRDCON:
1807 :          MOV     [CONTPOS],SI
1808 : ENDRDDEV:
1809 :          MOV     [NEXTADD],DI
1810 :          POP     ES
1811 :          JNZ     SETFCBJ                         ;Zero set if Ctrl-Z found in input
1812 :          MOV     DI,[FCB]
1813 :          AND     ES:BYTE PTR [DI.DEVID],0FFH-40H ;Mark as no more data available
1814 : SETFCBJ:
1815 :          JMP     SETFCB
```

— Lorsqu'on passe à cette partie, `SI` pointe sur CONTPOS, la position, et `DI` sur l'adresse en cours dans le tampon de transfert. On transfère un caractère de ? ? vers le tampon (lignes 1791–1793).

— Si le caractère transféré est celui de fin de ligne (de code ASCII 13), on ajoute un caractère ASCII 10 dans le tampon, les deux caractères ASCII de codes 13 et 10 constituant l'indicateur d'une fin de ligne en MS-DOS, (lignes 1794–1797) et on recommence tant qu'on n'est pas parvenu à une fin de ligne (lignes 1798–1799).

— Lorsqu'on est parvenu à une fin de ligne, on en effectue un écho à l'écran (lignes 1800–1801) et on place 0 dans `SI` pour se préparer à la ligne suivante (ligne 1802).

— Si le contenu de `CX` est non nul, on va à GETBUF (lignes 1803–1804), partie de la routine READDEV.

— Sinon on baisse le drapeau ZF pour indiquer qu'on n'est pas parvenu à une fin de ligne, on place le contenu de `SI` dans CONTPOS, celui de `DI`, l'adresse en cours dans le tampon, dans NEXTADD et on restaure le contenu de `ES` (lignes 1806–1810).

— Si c'est le caractère CTRL-Z, marque de fin de fichier pour MS-DOS, on place `FF40h` dans le champ DEVID du FCB pour indiquer qu'il n'y a plus de données disponibles (lignes 1811–1814).

— Dans tous les cas, on va à SETFCB (lignes 1814–1815) pour mettre à jour les champs de lecture séquentielle du FCB.

5.8.10.5 Découpage de l'enregistrement à lire

La routine BREAKDOWN détermine le nombre d'octets à transférer depuis la fin du premier secteur, le nombre de secteurs entiers et le nombre d'octets à transférer depuis le début du dernier secteur.

Avant d'appeler la routine, le segment des données et le segment de code doivent être confondus, `CX` contenir la taille, en octets, de ce qu'il faut transférer et BYTSECPOS la position du premier octet à transférer dans le premier secteur à transférer. Après exécution de la routine, BYTCNT1 contient le nombre d'octets à transférer dans le premier secteur, SECCNT le nombre de secteurs entiers à transférer et BYTCNT2 le nombre d'octets à transférer du dernier secteur :

```
1588 : BREAKDOWN:
1589 :
1590 : ;Inputs:
1591 : ;       DS = CS
1592 : ;       CX = Length of disk transfer in bytes
1593 : ;       BP = Base of drive parameters
1594 : ;       [BYTSECPOS] = Byte position witin first sector
1595 : ;Outputs:
1596 : ;       [BYTCNT1] = Bytes to transfer in first sector
1597 : ;       [SECCNT] = No. of whole sectors to transfer
1598 : ;       [BYTCNT2] = Bytes to transfer in last sector
1599 : ;AX, BX, DX destroyed. No other registers affected.
1600 :
1601 :         MOV     AX,[BYTSECPOS]
1602 :         MOV     BX,CX
1603 :         OR      AX,AX
1604 :         JZ      SAVFIR          ;Partial first sector?
1605 :         SUB     AX,[BP.SECSIZ]
1606 :         NEG     AX              ;Max number of bytes left in first sector
1607 :         SUB     BX,AX           ;Subtract from total length
1608 :         JAE     SAVFIR
1609 :         ADD     AX,BX           ;Don't use all of the rest of the sector
1610 :         XOR     BX,BX           ;And no bytes are left
1611 : SAVFIR:
1612 :         MOV     [BYTCNT1],AX
1613 :         MOV     AX,BX
1614 :         XOR     DX,DX
1615 :         DIV     [BP.SECSIZ]     ;How many whole sectors?
1616 :         MOV     [SECCNT],AX
1617 :         MOV     [BYTCNT2],DX    ;Bytes remaining for last sector
1618 : RET10:  RET
```

— On place la position du premier octet du premier secteur à transférer dans `AX` et le nombre d'octets à transférer dans `BX` (lignes 1601–1602).

— Si la position du premier octet dans le premier secteur est non nulle, le premier secteur doit être transféré partiellement. On soustrait la taille d'un secteur à `AX` et on le complémente pour obtenir le nombre d'octets qui ne seront pas récupérés dans le premier secteur. On le soustrait du nombre d'octets à transférer (lignes 1603–1607).

— Si on n'utilise pas tout le reste du secteur, on ajoute le nombre obtenu à `AX` et on place 0 dans `BX` (lignes 1608–1611).

— `AX` contient alors le nombre d'octets du premier secteur à transférer, que l'on place dans BYTCNT1 (ligne 1612).

— On place dans `AX` le nombre d'octets à transférer moins ceux qui seront récupérés dand le premier secteur, 0 dans `DX`, parce qu'on va effectuer une division, et on divise le contenu de `DX:AX` par la taille d'un secteur pour obtenir le nombre de secteurs à transférer entièrement, que l'on place dans SECCNT (lignes 1613–1616).

— Le reste dans la division euclidienne, obtenu dans `DX`, est le nombre d'octets à transférer depuis le dernier secteur. On le place dans BYTCNT2 et on termine la routine (ligne 1617–1618).

5.8.10.6 Lecture d'un secteur dans le tampon de transfert

La routine BUFRD copie n octets du tampon de transfert d'un secteur et les place dans la zone pointée par `ES:DI`.

Avant d'appeler cette routine, `AL` contient la valeur de PREREAD pour savoir si on doit lire le secteur quoi qu'il advienne et `CX` le nombre d'octets à lire :

```
1728 : BUFRD:
1729 :          XOR     AL,AL           ;Pre-read necessary
1730 :          CALL    BUFSEC
1731 :          PUSH    ES
1732 :          MOV     ES,[DMAADD+2]
1733 :          SHR     CX,1
1734 :          JNC     EVENRD
1735 :          MOVSB
1736 : EVENRD:
1737 :          REP     MOVSW
1738 :          POP     ES
1739 :          RET
```

— Si `AL` est nul, il y a besoin de lire un secteur auparavant, on appelle donc la routine BUFSEC pour transférer un secteur de la disquette dans le tampon de transfert en mémoire centrale (lignes 1729–1730).

Au retour de cette routine, `SI` pointe sur l'octet non encore lu dans le tampon et `DI` sur l'adresse NEXTADD de la zone de transfert.

— On sauvegarde le contenu de `ES` sur la pile, on fait pointer `ES` sur le segment du tampon, si le nombre d'octets à lire est impair alors on lit un octet, puis on lit mot par mot, on restaure la valeur de `ES` et on termine la routine (lignes 1731–1739).

5.8.10.7 Routine de lecture au début du dernier secteur

Lors de la lecture séquentielle, il faut lire la fin du secteur déjà dans le tampon, lire des secteurs entiers et lire le début du dernier secteur. Pour ce dernier cas, on utilise la routine RDLAST :

```
2000 : RDLAST:
2001 :          MOV     AX,[BYTCNT2]
2002 :          OR      AX,AX
2003 :          JZ      SETFCB
2004 :          MOV     [BYTCNT1],AX
2005 :          CALL    NEXTSEC
2006 :          JC      SETFCB
2007 :          MOV     [BYTSECPOS],0
2008 :          CALL    BUFRD
2009 :          JMP     SHORT SETFCB
```

— On place le nombre d'octets à lire dans ce dernier secteur, contenu dans la variable BYTCNT2, dans `AX`. S'il y a des octets à lire dans le dernier secteur, c'est-à-dire si ce nombre est non nul, on le place dans BYTCNT1, on appelle la routine NEXTSEC pour déterminer le secteur suivant du fichier, et éventuellement le numéro de l'unité d'allocation associée si on passe à une nouvelle unité d'allocation, on place 0 dans BYTSECPOS pour indiquer qu'il faut lire depuis le début de ce secteur et on appelle la routine BUFRD pour transférer ces octets dans la zone de transfert (lignes 2001–2002 et 2004–2008).

— On va à la partie SETFCB pour renseigner les champs adéquats du FCB (lignes 2003 et 2009).

5.8.10.8 Routine d'optimisation dans la lecture de secteurs

Étant donnés un secteur du fichier, spécifié par le numéro d'unité d'allocation p dans lequel il se trouve et sa position dans celle-ci, ainsi qu'un nombre n de secteurs, la routine OPTIMIZE regarde si les unités d'allocation de numéro $p+1, p+2, \ldots$ appartiennent au fichier, sans dépasser le nombre de secteurs voulu, afin d'optimiser le nombre d'opérations de lecture sur la disquette.

Avant d'appeler la routine, le segment des données et le segment de code doivent être confondus, `BX` contenir le numéro d'unité d'allocation, `CX` le nombre de secteurs à lire, `DL` la position du secteur dans l'unité d'allocation, `BP` pointer sur le bloc des paramètres du lecteur de disquette et NEXTADD contenir une adresse dans la zone de transfert.

Après exécution de la routine, `AX` contient le nombre de secteurs qu'il restera à charger après l'opération de lecture sur la disquette, `BX` l'adresse de transfert, `CX` le nombre de secteurs à transférer lors de cette opération de lecture sur la disquette, `DX` l'adresse du secteur, `DI` le numéro d'unité d'allocation suivante du fichier (il y a un trou entre l'unité d'allocation atteinte et celle-ci), CLUSNUM le numéro de l'unité d'allocation atteinte, NEXTADD est mis à jour, le drapeau CF est baissé si le premier secteur qui devait être transféré se trouve déjà dans le tampon :

```
2228 : OPTIMIZE:
2229 :
2230 : ; Inputs:
2231 : ;        DS = CS
2232 : ;        BX = Physical cluster
2233 : ;        CX = No. of records
2234 : ;        DL = sector within cluster
2235 : ;        BP = Base of drives parameters
2236 : ;        [NEXTADD] = transfer address
2237 : ; Outputs:
2238 : ;        AX = No. of records remaining
2239 : ;        BX = Transfer address
2240 : ;        CX = No. or records to be transferred
```

```
2241 : ;       DX = Physical sector address
2242 : ;       DI = Next cluster
2243 : ;       Carry clear if a sector to transfer is in the buffer
2244 : ;       Carry set otherwise
2245 : ;       [CLUSNUM] = Last cluster accessed
2246 : ;       [NEXTADD] updated
2247 : ; BP unchanged. Note that segment of transfer not set.
2248 :
2249 :         PUSH    DX
2250 :         PUSH    BX
2251 :         MOV     AL,[BP.CLUSMSK]
2252 :         INC     AL              ;Number of sectors per cluster
2253 :         MOV     AH,AL
2254 :         SUB     AL,DL           ;AL = Number of sectors left in first cluster
2255 :         MOV     DX,CX
2256 :         MOV     SI,[BP.FAT]
2257 :         MOV     CX,0
2258 : OPTCLUS:
2259 : ;AL has number of sectors available in current cluster
2260 : ;AH has number of sectors available in next cluster
2261 : ;BX has current physical cluster
2262 : ;CX has number of sequential sectors found so far
2263 : ;DX has number of sectors left to transfer
2264 : ;SI has FAT pointer
2265 :         CALL    UNPACK
2266 :         ADD     CL,AL
2267 :         ADC     CH,0
2268 :         CMP     CX,DX
2269 :         JAE     BLKDON
2270 :         MOV     AL,AH
2271 :         INC     BX
2272 :         CMP     DI,BX
2273 :         JZ      OPTCLUS
2274 :         DEC     BX
2275 : FINCLUS:
2276 :         MOV     [CLUSNUM],BX    ;Last cluster accessed
2277 :         SUB     DX,CX           ;Number of sectors still needed
2278 :         PUSH    DX
2279 :         MOV     AX,CX
2280 :         MUL     [BP.SECSIZ]     ;Number of sectors times sector size
2281 :         MOV     SI,[NEXTADD]
2282 :         ADD     AX,SI           ;Adjust by size of transfer
2283 :         MOV     [NEXTADD],AX
2284 :         POP     AX              ;Number of sectors still needed
2285 :         POP     DX              ;Starting cluster
2286 :         SUB     BX,DX           ;Number of new clusters accessed
2287 :         ADD     [LASTPOS],BX
2288 :         POP     BX              ;BL = sector postion within cluster
2289 :         CALL    FIGREC
2290 :         MOV     BX,SI
2291 : ;Now let's see if any of these sectors are already in the buffer
2292 :         CMP     [BUFSECNO],DX
2293 :         JC      RET100          ;If DX > [BUFSECNO] then not in buffer
2294 :         MOV     SI,DX
2295 :         ADD     SI,CX           ;Last sector + 1
2296 :         CMP     [BUFSECNO],SI
2297 :         CMC
2298 :         JC      RET100          ;If SI <= [BUFSECNO] then not in buffer
2299 :         PUSH    AX
2300 :         MOV     AL,[BP.DEVNUM]
2301 :         CMP     AL,[BUFDRVNO]   ;Is buffer for this drive?
2302 :         POP     AX
```

```
2303 :          JZ      RET100          ;If so, then we match
2304 :          STC                     ;No match
2305 : RET100: RET
```

— On sauvegarde sur la pile la position du secteur dans l'unité d'allocation et le numéro de celle-ci, puisque DX et BX vont être surchargés dans le cœur de cette routine. On place le nombre de secteurs par unité d'allocation dans AH, le nombre de secteurs à transférer dans cette unité d'allocation dans AL, le nombre de secteurs qu'il faut transférer dans DX, l'adresse de la copie de la FAT en mémoire centrale dans SI et 0 dans CX pour indiquer le nombre de secteurs trouvés jusqu'à maintenant (lignes 249–2264).

— On appelle la routine UNPACK pour placer dans DI le numéro de l'unité d'allocation suivante du fichier (ligne 2265).

— Si le nombre de secteurs restants dans l'unité d'allocation plus le nombre de secteurs ? ? n'est pas égal au nombre de secteurs restant à transférer, on ? ? (lignes 2266–2269).

La partie BLKDON est définie lignes 2306–2312 :

```
2306 : BLKDON:
2307 :          SUB     CX,DX           ;Number of sectors in cluster we don't want
2308 :          SUB     AH,CL           ;Number of sectors in cluster we accepted
2309 :          DEC     AH              ;Adjust to mean position within cluster
2310 :          MOV     [SECCLUSPOS],AH
2311 :          MOV     CX,DX           ;Anyway, make the total equal to the request
2312 :          JMP     SHORT FINCLUS
```

— On place le nombre de secteurs disponible dans l'unité d'allocation suivante dans AL et on incrémente le numéro de l'unité d'allocation. Si le numéro trouvé est égal au numéro voulu, on recommence à partir de l'étiquette OPTCLUS (lignes 2270–2273).

— Sinon on revient au numéro antérieur d'unité d'allocation, puisque la dernière n'appartenait pas au fichier (lignes 2274–2275).

— On place le numéro de la dernière unité d'allocation du fichier atteinte, contenu dans BX, dans la variable CLUSNUM (ligne 2276).

— On soustrait le contenu de CX à celui de DX pour obtenir dans DX le nombre de secteurs encore nécessaires, que l'on sauvegarde sur la pile, puisque le contenu de DX va être détruit par la multiplication suivante (ligne 2277–2278).

— On place le nombre de secteurs à transférer dans AX, que l'on multiplie par la taille d'un secteur pour obtenir le nombre d'octets à transférer, ce qui permet de mettre à jour l'adresse du prochain octet du tampon (l'ancienne adresse plus le nombre d'octets à transférer) (lignes 2279–2283).

— On place dans AX le nombre de secteurs encore nécessaires et dans DX le numéro de départ des unités d'allocation, on soustrait de BX ce numéro de départ pour obtenir le nombre de nouvelles unités d'allocation auxquelles on a accédé, qu'on ajoute à la variable LASTPOS pour mettre à jour la dernière unité d'allocation à laquelle on a accédé (lignes 2284–2287).

— On récupère dans BL la position du secteur dans l'unité d'allocation et on appelle la routine FIGREC pour obtenir dans DX le numéro du secteur et on place le contenu de SI dans BX (lignes 2288–2290).

— On vérifie si tous les secteurs souhaités sont déjà dans le tampon. Pour cela, si le contenu de DX est strictement supérieur au contenu de BUFSECNO, on fait pointer SI juste après le dernier secteur qui se trouve dans le tampon. Si le contenu de SI est inférieur ou égal à BUFSECNO alors il n'est pas dans le tampon, on sauvegarde le contenu de AX sur la pile, on place le numéro de lecteur de disquette dans AL, on lève CF s'il ne correspond pas au numéro de lecteur du tampon et on termine la routine (lignes 2291–2305).

5.8.11 Routine de service de la fonction 21 (15h) d'écriture séquentielle

5.8.11.1 Routine principale

La routine de service de la fonction 21 de l'interruption `21h`, d'écriture séquentielle, est repérée par l'étiquette SEQWRT :

```
1401 : SEQWRT: ;System call 21
1402 :         CALL    GETREC
1403 :         CALL    STORE
1404 : FINSEQ:
```

— `DS:DX` pointant sur un FCB ouvert, on appelle, comme dans le cas de SEQRD, la routine GETREC pour obtenir dans `DX:AX` le numéro absolu de l'enregistrement index.

— On appelle ensuite la routine STORE, étudiée ci-dessous, pour écrire les enregistrements sur la disquette (ou sur le tampon du secteur).

— Puis on passe à la partie FINSEQ pour mettre à jour le numéro d'enregistrement index dans le FCB.

5.8.11.2 Routine d'écriture d'enregistrements d'un fichier sur la disquette

La routine STORE enregistre sur la disquette des enregistrements se trouvant sur la zone de transfert en mémoire centrale.

Avant de l'appeler, `DS:DI` doit pointer sur le FCB, `DX:AX` sur la zone de transfert de ce qu'il faut écrire sur le fichier et `CX` contenir le nombre d'enregistrements à écrire. Il n'y a pas besoin de préciser le numéro du premier enregistrement, puisque celui-ci est spécifié par le FCB.

Après exécution de la routine, `DX:AX` donne la position du dernier enregistrement écrit, `CX` le nombre d'enregistrements écrits, `ES:DI` pointe sur le FCB et les champs 'bloc' et 'numéro d'enregistrement dans le bloc' du FCB ont été mis à jour dans le FCB :

```
2064 : WRTEOFJ:
2065 :         JMP     WRTEOF
2066 :
2067 : STORE:
2068 :
2069 : ; Inputs:
2070 : ;       DS:DI point to FCB
2071 : ;       DX:AX = Position in file of disk transfer
2072 : ;       CX = Record count
2073 : ; Outputs:
2074 : ;       DX:AX = Position of last record written
2075 : ;       CX = No. of records written
2076 : ;       ES:DI point to FCB
2077 : ;       LSTCLUS, CLUSPOS fields in FCB set
2078 :
2079 :         CALL    SETUP
2080 :         CALL    DATE16
2081 :         MOV     ES:[DI.FDATE],AX
2082 :         MOV     ES:[DI.FTIME],DX
2083 :         OR      BL,BL
2084 :         JS      WRTDEV
2085 :         AND     BL,3FH                  ;Mark file as dirty
2086 :         MOV     ES:[DI.DEVID],BL
2087 :         CALL    BREAKDOWN
2088 :         MOV     AX,WORD PTR [BYTPOS]
2089 :         MOV     DX,WORD PTR [BYTPOS+2]
2090 :         JCXZ    WRTEOFJ
2091 :         DEC     CX
```

```
2092 :          ADD     AX,CX
2093 :          ADC     DX,0                    ;AX:DX=last byte accessed
2094 :          DIV     [BP.SECSIZ]             ;AX=last sector accessed
2095 :          MOV     CL,[BP.CLUSSHFT]
2096 :          SHR     AX,CL                   ;Last cluster to be accessed
2097 :          PUSH    AX
2098 :          MOV     AX,ES:WORD PTR [DI.FILSIZ]
2099 :          MOV     DX,ES:WORD PTR [DI.FILSIZ+2]
2100 :          DIV     [BP.SECSIZ]
2101 :          OR      DX,DX
2102 :          JZ      NORNDUP
2103 :          INC     AX                      ;Round up if any remainder
2104 : NORNDUP:
2105 :          MOV     [VALSEC],AX             ;Number of sectors that have been written
2106 :          POP     AX
2107 :          MOV     CX,[CLUSNUM]            ;First cluster accessed
2108 :          CALL    FNDCLUS
2109 :          MOV     [CLUSNUM],BX
2110 :          MOV     [LASTPOS],DX
2111 :          SUB     AX,DX                   ;Last cluster minus current cluster
2112 :          JZ      DOWRT                   ;If we have last clus, we must have first
2113 :          JCXZ    HAVSTART                ;See if no more data
2114 :          PUSH    CX                      ;No. of clusters short of first
2115 :          MOV     CX,AX
2116 :          CALL    ALLOCATE
2117 :          POP     AX
2118 :          JC      WRTERR
2119 :          MOV     CX,AX
2120 :          MOV     DX,[LASTPOS]
2121 :          INC     DX
2122 :          DEC     CX
2123 :          JZ      NOSKIP
2124 :          CALL    SKPCLP
2125 : NOSKIP:
2126 :          MOV     [CLUSNUM],BX
2127 :          MOV     [LASTPOS],DX
2128 : DOWRT:
2129 :          CMP     [BYTCNT1],0
2130 :          JZ      WRTMID
2131 :          MOV     BX,[CLUSNUM]
2132 :          CALL    BUFWRT
2133 : WRTMID:
2134 :          MOV     AX,[SECCNT]
2135 :          OR      AX,AX
2136 :          JZ      WRTLAST
2137 :          ADD     [SECPOS],AX
2138 :          CALL    NEXTSEC
2139 :          MOV     BYTE PTR [TRANS],1      ;A transfer is taking place
2140 :          MOV     DL,[SECCLUSPOS]
2141 :          MOV     BX,[CLUSNUM]
2142 :          MOV     CX,[SECCNT]
2143 : WRTLP:
2144 :          CALL    OPTIMIZE
2145 :          JC      NOTINBUF                ;Is one of the sectors buffered?
2146 :          MOV     [BUFSECNO],0            ;If so, invalidate the buffer since we're
2147 :          MOV     WORD PTR [BUFDRVNO],0FFH ;       completely rewritting it
2148 : NOTINBUF:
2149 :          PUSH    DI
2150 :          PUSH    AX
2151 :          PUSH    DS
2152 :          MOV     DS,[DMAADD+2]
2153 :          CALL    DWRITE
```

```
2154 :          POP     DS
2155 :          POP     CX
2156 :          POP     BX
2157 :          JCXZ    WRTLAST
2158 :          MOV     DL,0
2159 :          INC     [LASTPOS]                   ;We'll be using next cluster
2160 :          JMP     SHORT WRTLP
2161 : WRTLAST:
2162 :          MOV     AX,[BYTCNT2]
2163 :          OR      AX,AX
2164 :          JZ      FINWRT
2165 :          MOV     [BYTCNT1],AX
2166 :          CALL    NEXTSEC
2167 :          MOV     [BYTSECPOS],0
2168 :          CALL    BUFWRT
2169 : FINWRT:
2170 :          MOV     AX,[NEXTADD]
2171 :          SUB     AX,[DMAADD]
2172 :          ADD     AX,WORD PTR [BYTPOS]
2173 :          MOV     DX,WORD PTR [BYTPOS+2]
2174 :          ADC     DX,0
2175 :          MOV     CX,DX
2176 :          MOV     DI,[FCB]
2177 :          CMP     AX,ES:WORD PTR [DI.FILSIZ]
2178 :          SBB     CX,ES:WORD PTR [DI.FILSIZ+2]
2179 :          JB      SAMSIZ
2180 :          MOV     ES:WORD PTR [DI.FILSIZ],AX
2181 :          MOV     ES:WORD PTR [DI.FILSIZ+2],DX
2182 : SAMSIZ:
2183 :          MOV     CX,[RECCNT]
2184 :          JMP     SETCLUS
```

Préparation à l'écriture

— On appelle la routine SETUP de préparation à l'écriture (ligne 2079).

Avant d'appeler cette routine, `DS:DI` doit pointer sur le FCB, `DX:AX` sur l'adresse dans la zone en mémoire centrale de ce qu'il faut écrire et `CX` contenir le nombre d'enregistrements qu'il faut écrire sur le fichier, ce qui est le cas.

Après exécution de la routine, le segment des données est confondu avec le segment de code, `ES:DI` pointe sur le FCB, `BL` contient le champ 'numéro de périphérique' du FCB, `CX` contient le nombre d'octets à écrire, `BP` pointe sur le bloc des paramètres du lecteur de disquette, `SI` sur la copie de la FAT en mémoire centrale, RECCNT contient le nombre d'enregistrements à écrire, RECPOS la position du premier enregistrement qu'il faut écrire, FCB pointe sur le FCB, NEXTADD contient l'adresse dans la zone de transfert à partir de laquelle il faut écrire, SECPOS la position du premier secteur sur lequel on doit écrire, BYTPOS le numéro dans le fichier du premier octet sur lequel il faut écrire, BYTSECPOS la position du premier octet à écrire dans le secteur adéquat, CLUSNUM le numéro de la première unité d'allocation sur laquelle il faut écrire, SECCLUSPOS la position dans cette première unité d'allocation du premier secteur sur lequel il faut écrire, DSKERR le code d'erreur de lecture sur le disque (`00h` s'il n'y a pas d'erreur), TRANS l'indicateur de transfert (nul tant qu'il n'y a pas eu d'écriture) et THISDRIV le numéro de lecteur de disquette (avec 0 pour A). Si SETUP n'a rien trouvé à écrire, il place `00h` dans `CX`.

— On appelle la routine DATE16 pour obtenir la date et l'heure, que l'on reporte dans le FCB (lignes 2080–2082).

Cas d'un fichier de périphérique

— S'il s'agit d'un fichier de périphérique, on va à la partie WRTDEV, étudiée ci-dessous, d'écriture dans un fichier de périphérique. Sinon, on indique qu'on est en train d'écrire sur le fichier, et qu'on ne peut donc pas y effectuer de lecture fiable (lignes 2083–2086).

Découpage de l'enregistrements sur plusieurs secteurs

— Comme nous l'avons vu, l'enregistrement index s'étend, dans le cas général, sur la fin du secteur déjà dans le tampon de secteur, sur un certain nombre de secteurs, qui seront complètement reportés sur la diquette, et sur le début d'un dernier secteur, qui sera placé sur le tampon de secteur. On appelle la routine BREAKDOWN pour effectuer ce découpage, plus précisément pour placer dans la variable BYTCNT1 le nombre d'octets à écrire à la fin du tampon de secteur (et enregistrer celui-ci sur la disquette s'il est alors totalement rempli), dans SECCNT le nombre de secteurs sur lesquels écrire complètement et, enfin, dans BYTCNT2 le nombre d'octets à écrire au début du tampon de secteur qui restera en mémoire vive, s'il n'est pas complètement rempli (ligne 2087).

Place suffisante sur la disquette pour cette écriture ?

— On place dans `DX:AX` la position du premier octet à écrire (lignes 2088–2089).

— Si le nombre d'octets à écrire est nul, on termine la routine en appelant la routine WRITEOF, étudiée ci-dessous (lignes 2090 et 2064–2065).

— Sinon on décrémente le nombre d'octets à écrire, on lui ajoute la position de l'index pour indiquer la nouvelle position de l'index, on divise celle-ci par la taille d'un secteur pour obtenir le numéro du dernier secteur du fichier sur lequel on aura écrit entièrement, puis par le nombre de secteurs par unité d'allocation pour obtenir la position dans le fichier de la dernière unité d'allocation du fichier sur laquelle on aura écrit, que l'on place sur la pile (lignes 2091–2097).

— On place la taille du fichier dans `DX:AX`, on la divise par la taille d'un secteur, si `DX` est non nul on incrémente `AX` pour arrondir et on place le nombre de secteurs déjà remplis dans le fichier dans la variable VALSEC (lignes 2098–2105).

La variable VALSEC contient le nombre de secteurs du fichier sur lesquels on a déjà écrit :

```
3710 : VALSEC  DW      ?       ;Number of valid (previously written) sectors
```

— On restaure dans `AX` la position dans le fichier de la dernière unité d'allocation sur laquelle on veut écrire, on place dans `CX` le numéro de la première unité d'allocation à laquelle on veut accéder et on appelle la routine FNDCLUS pour savoir s'il s'agit d'une unité d'allocation du fichier (lignes 2106–2108).

Avant d'appeler cette routine, le segment des données doit être confondu avec le segment de code, `CX` contenir la position n dans le fichier de l'unité d'allocation dont on veut savoir s'il lui appartient, `BP` pointer sur le bloc des paramètres du lecteur de la disquette sur laquelle se trouve le fichier, `SI` sur la copie de la FAT en mémoire centrale et `ES:DI` sur le FCB du fichier, ce qui est le cas.

Après son exécution, `BX` contient le numéro (sur la disquette) de l'unité d'allocation atteinte, `CX` le nombre d'unités d'allocation supplémentaires que le fichier devrait contenir pour atteindre celle de position n et `DX` la position dans le fichier de l'unité d'allocation à laquelle on est parvenu, soit $min(n, d)$.

— On place le numéro de l'unité d'allocation atteinte, renvoyé par FNDCLUS dans `BX`, dans la variable CLUSNUM et la position de celle-ci dans le fichier, renvoyée dans `DX` par FNDCLUS, dans la variable LASTPOS (lignes 2109–2110).

— On soustrait cette position de la position de la dernière unité d'allocation du fichier à laquelle on a accédé, pour obtenir le nombre d'unités d'allocation dont on a besoin. Si on trouve 0, on doit partir de celle-ci ; on va donc à la partie DOWRT, étudiée ci-dessous, pour commencer l'écriture (lignes 2111–2112).

— Sinon on va à la partie HAVSTART, étudiée ci-dessous, pour voir s'il y a assez d'unités d'allocation dans le fichier ou si on peut lui ajouter le nombre nécessaire d'unités d'allocation (ligne 2113). Si on y parvient, la partie HAVSTART nous fait aller, comme ci-dessus, à la partie DOWRT pour commencer l'écriture.

— Si on ne parvient pas à ajouter suffisamment d'unités d'allocation dans le fichier, on sauvegarde sur la pile le contenu de `CX`, c'est-à-dire le nombre d'unités d'allocation dont on a besoin, on place dans `CX` le nombre d'unités d'allocation dont on a besoin et on appelle la routine ALLOCATE pour essayer d'ajouter ce nombre d'unités d'allocation nouvelles au fichier (lignes 2114–2116).

Avant d'appeler la routine ALLOCATE, le segment des données doit être confondu avec le segment de code, le segment supplémentaire doit contenir le FCB, `BX` le numéro de la dernière unité d'allocation du fichier (0 s'il s'agit d'un fichier vide), `CX` le nombre d'unités d'allocation nouvelles souhaitées, `DX` la position dans le fichier de l'unité d'allocation de numéro contenu dans `BX`, `BP` doit pointer sur le bloc des paramètres du lecteur de disquette, `SI` sur la copie de la FAT en mémoire centrale et FCB contenir le décalage du FCB dans le segment supplémentaire.

Après exécution de cette routine, s'il n'y a pas suffisamment d'unités d'allocation libres sur la disquette, le drapeau CF est levé et `CX` contient le nombre maximum de secteurs pouvant être ajoutés à ce fichier. Sinon le drapeau CF est baissé, `BX` contient le numéro de la première unité d'allocation allouée, la copie de la FAT en mémoire centrale est mise à jour, y compris l'octet indiquant une modification de celle-ci depuis sa dernière sauvegarde sur la disquette, le champ FIRCLUS du FCB est mis à jour si le fichier était vide.

— On replace dans `AX` le nombre d'unités d'allocation nouvelles dont on a besoin dans le fichier. Si on n'est pas parvenu à obtenir le nombre voulu d'unités d'allocation, on indique le numéro d'erreur correspondante. Sinon on place le nombre d'unités d'allocation nouvelles dont on a besoin dans le fichier moins une dans `CX`, la position dans le fichier de la dernière unité d'allocation plus un dans `DX` et si la valeur de `CX` est non nulle on va à l'étiquette SKPCLP, repérant une partie de FNDCLUS : on appelle la routine UNPACK pour obtenir l'entrée dans la FAT de la dernière unité d'allocation ; si on trouve `FF8h`, c'est-à-dire s'il s'agissait de la dernière unité d'allocation du fichier, on termine la routine ; sinon on place le numéro de l'unité pointée dans `BX`, on incrémente `DX`, c'est-à-dire la position de la dernière unité d'allocation utilisée et on recommence jusqu'à avoir sauté le plus grand nombre possible d'unités d'allocation et on termine la routine (lignes 2117–2124).

— S'il reste suffisamment de place dans le fichier pour y placer les enregistrements ou si on a pu lui ajouter d'autres unités d'allocation pour ce faire, on place le numéro de la dernière unité d'allocation sur laquelle on a écrit quelque chose sur le fichier dans la variable CLUSNUM et la position de celle-ci dans le fichier dans la variable LASTNUM (lignes 2125–2127).

Écriture à la fin du secteur se trouvant dans le tampon

— Si le nombre d'octets à écrire à la fin du secteur du fichier se trouvant actuellement dans le tampon est non nul, on place le numéro de cette unité d'allocation dans `BX` et on appelle la routine BUFWRT, étudiée ci-dessous, pour écrire dans le tampon, sans l'enregistrer sur la disquette (lignes 2128–2132).

Écriture des secteurs complets

— On écrit ensuite sur le disque les secteurs entiers. Pour cela, on place dans `AX` le nombre de secteurs entiers à écrire. S'il n'y en a pas, on passe à la dernière phase, celle d'écriture au début du dernier secteur. Sinon on ajoute ce nombre de secteurs à la position du dernier secteur du fichier sur lequel on a écrit, que l'on sauvegarde dans la variable SECPOS, on appelle la routine NEXTSEC pour placer dans les variables CLUSNUM et SECCLUSPOS le numéro d'unité d'allocation et la position du secteur dans celle-ci du secteur suivant celui qui était spécifié par ces variables, on place 1 dans la variable TRANS pour indiquer qu'un transfert est en train de s'effectuer, on place le contenu de SECPLUSPOS dans `DL`, de CLUSNUM dans `BX`, de SECCNT dans `CX` pour se préparer à l'écriture de ces secteurs (lignes 2133–2142).

— L'écriture s'effectue, de façon analogue à la lecture, en optimisant le nombre d'accès au disque pour les opérations d'écriture. Pour cela on utilise une boucle qui détermine le nombre de secteurs consécutifs sur le disque qu'on peut écrire en une seule opération d'écriture, les écrit, puis passe à l'intervalle suivant de secteurs consécutifs sur le disque jusqu'à ce qu'on ait écrit tous les secteurs souhaités (lignes 2143–2160).

Écriture au début du dernier secteur

— On place le nombre d'octets à écrire sur ce dernier secteur dans `AX`. S'il est nul on passe à la phase de terminaison de l'écriture. Sinon on place ce nombre dans la variable BYTCNT1 et on appelle la routine NEXTSEC pour placer dans les variables CLUSNUM et SECCLUSPOS le numéro d'unité d'allocation et la position du secteur dans celle-ci du secteur suivant celui qui était spécifié par ces variables, c'est-à-dire le dernier secteur entier écrit, on place 0 dans la variable BYTSECPOS, pour indiquer qu'il faudra écrire depuis le début de ce secteur, et on appelle la routine BUFWRT pour écrire sur le tampon de secteur, sans l'enregistrer immédiatement sur la disquette (lignes 2161–2168).

Fin de l'écriture

— On place dans `AX` l'adresse à laquelle on est parvenu dans la zone de transfert, que l'on soustrait à l'adresse du début de cette zone de transfert pour obtenir le nombre d'octets écrits, que l'on ajoute à l'ancienne position du dernier octet écrit dans le fichier pour obtenir la nouvelle taille du fichier dans `CX:AX`. Si la taille a changé, on place la nouvelle taille sur le FCB (lignes 2169–2182).

— On place le nombre de secteurs RECCNT dans `CX` et on va à l'étiquette SETCLUS pour exécuter une partie de la routine SETFCB, à savoir mettre à jour la position de l'enregistrement index et on termine la routine (lignes 2183–2184).

5.8.11.3 Écriture sur un fichier spécial

Avant d'appeler la routine WRTDEV, le registre `BL` contient le numéro de périphérique tel que le donne le FCB :

```
2011 : WRTDEV:
2012 :          PUSH    DS
2013 :          LDS     SI,DWORD PTR [DMAADD]
2014 :          OR      BL,40H
2015 :          INC     BL
2016 :          JZ      WRTCON
2017 :          INC     BL
2018 :          JZ      WRTAUX
2019 :          INC     BL
2020 :          JZ      ENDWRDEV        ;Done if device is NUL
2021 : WRTLST:
```

```
2022 :          LODSB
2023 :          CMP     AL,1AH
2024 :          JZ      ENDWRDEV
2025 :          CALL    LISTOUT
2026 :          LOOP    WRTLST
2027 :          JMP     SHORT ENDWRDEV
2028 :
2029 : WRTAUX:
2030 :          LODSB
2031 :          CALL    AUXOUT
2032 :          CMP     AL,1AH
2033 :          LOOPNZ  WRTAUX
2034 :          JMP     SHORT ENDWRDEV
2035 :
2036 : WRTCON:
2037 :          LODSB
2038 :          CMP     AL,1AH
2039 :          JZ      ENDWRDEV
2040 :          CALL    OUT
2041 :          LOOP    WRTCON
2042 : ENDWRDEV:
2043 :          POP     DS
2044 :          MOV     CX,[RECCNT]
2045 :          MOV     DI,[FCB]
2046 :          JMP     SHORT ADDREC
```

— On sauvegarde l'adresse du segment des donnés sur la pile, on fait pointer `SI` sur le début de la zone en mémoire centrale de ce qu'il faut écrire sur le périphérique et on ne garde que les 4 bits de poids fort de `BL` (lignes 2011–2014).

— Si le numéro de fichier est `F0h`, le périphérique est l'écran. On envoie un à un les caractères de la zone de transfert en mémoire centrale à l'écran jusqu'à ce qu'on rencontre le caractère de code ASCII `1Ah`, indicateur de fin de fichier. On restaure alors le contenu du registre `DS`, on place le nombre d'enregistrements antérieurs dans `CX`, on fait pointer `DI` sur le FCB et on va à ADDREC, qui est une partie de la routine SETFCB : si on n'a pas lu de nouvel enregistrement, on ne modifie pas le champ 'position de l'enregistrement index' du FCB, sinon on le met à jour et on termine la routine (lignes 2015–2016, 2036–2046 et 1990–1998).

— Si le numéro de fichier est `E0h`, il s'agit du périphérique auxiliaire. On fait la même chose que dans le cas précédent, pour le périphérique auxiliaire au lieu de l'écran (lignes 2017–2018, 2042–2046 et 1990–1998).

— S'il s'agit du périphérique nul, on n'a rien de particulier à faire. On se contente donc de faire comme ci-dessus, à partir de la restauration du contenu du registre `DS` (lignes 2019–2020, 2042–2046 et 1990–1998).

— C'est analogue s'il s'agit de l'imprimante (lignes 2021–2027).

5.8.11.4 Place suffisante dans le fichier ?

En spécifiant un numéro d'unité d'allocation et un entier n, la portion de code HAVSTART regarde si le fichier auquel est allouée cette unité d'allocation est chaînée à au moins n unités d'allocation au-delà de celle dont le numéro a été passé en paramètre et, dans le cas contraire, s'il reste suffisamment d'unités d'allocation libres sur la disquette pour les allouer au fichier.

Avant d'appeler cette partie de code, `AX` doit contenir le nombre d'unités d'allocation souhaitées et `BX` un numéro d'unité d'allocation.

Après son exécution, `BX` contient le numéro d'unité d'allocation auquel on est parvenu, `CX` le nombre d'unités d'allocation supplémentaires qu'il aurait fallu que le fichier contienne pour que `BX` contienne le numéro de l'unité d'allocation de position n au-delà de celle passée en paramètre, `DX:AX` la position du secteur, DSKERR le code d'erreur `01h` si on n'est pas parvenu à la n-ième unité d'allocation au-delà de celle passée en paramètre et `DI` pointe sur le FCB :

```
2048 : HAVSTART:
2049 :          MOV     CX,AX
2050 :          CALL    SKPCLP
2051 :          JCXZ    DOWRTJ
2052 :          CALL    ALLOCATE
2053 :          JNC     DOWRTJ
2054 : WRTERR:
2055 :          MOV     BYTE PTR [DSKERR],1
2056 : LVDSK:
2057 :          MOV     AX,WORD PTR [RECPOS]
2058 :          MOV     DX,WORD PTR [RECPOS+2]
2059 :          MOV     DI,[FCB]
2060 :          RET
2061 :
2062 : DOWRTJ: JMP     DOWRT
```

— On place le nombre n d'unités d'allocation souhaitées, passé en paramètre *via* `AX`, dans `CX` et on appelle la routine SKPCLP, partie de la routine FNDCLUS, pour essayer de passer n unités d'allocation dans le fichier à partir de celle en cours, de placer son numéro dans `BX` et sa position dans le fichier dans `DX` ; sinon on essaie d'aller le plus loin possible, en plaçant dans `CX` le nombre d'unités supplérmentaires qu'il faudrait pour atteindre la n-ième (lignes 2048–2050).

— Si le contenu de `CX`, c'est-à-dire le nombre d'unités d'allocation qu'on n'a pas pu sauter, est nul alors tout va bien et on va à la partie DOWRT de la routine STORE pour écrire l'enregistrement (lignes 2051 et 2062).

— Sinon on appelle la routine ALLOCATE pour essayer d'allouer le nombre nécessaire d'unités d'allocation au fichier et, si on y parvient, on va à la partie DOWRT de la routine STORE (ligne 2052–2053).

— Si on n'y parvient pas, on place le code d'erreur `01h` dans DSKERR, la position de l'enregistrement dans `DX:AX`, on fait pointer `DI` vers le FCB du fichier et on termine la routine (lignes 2054–2060).

5.8.11.5 Écriture dans le tampon du secteur se trouvant en mémoire centrale

La routine BUFWRT permet d'écrire sur la copie en mémoire centrale du secteur, sans l'enregistrer immédiatement sur la disquette :

```
1741 : BUFWRT:
1742 :          MOV     AX,[SECPOS]
1743 :          INC     AX              ;Set for next sector
```

```
1744 :          MOV     [SECPOS],AX
1745 :          CMP     AX,[VALSEC]     ;Has sector been written before?
1746 :          MOV     AL,1
1747 :          JA      NOREAD          ;Skip preread if SECPOS>VALSEC
1748 :          MOV     AL,0
1749 : NOREAD:
1750 :          CALL    BUFSEC
1751 :          XCHG    DI,SI
1752 :          PUSH    DS
1753 :          PUSH    ES
1754 :          PUSH    CS
1755 :          POP     ES
1756 :          MOV     DS,[DMAADD+2]
1757 :          SHR     CX,1
1758 :          JNC     EVENWRT
1759 :          MOVSB
1760 : EVENWRT:
1761 :          REP     MOVSW
1762 :          POP     ES
1763 :          POP     DS
1764 :          MOV     BYTE PTR [DIRTYBUF],1
1765 :          RET
```

— On incrémente la position du secteur dans le fichier qui se trouvera dans le tampon après exécution de la routine (lignes 1741–1744).

— On place 1 dans `AL` si on a déjà enregistrer le tampon de ce secteur sur la disquette, 0 sinon (lignes 1745–1749).

— On appelle la routine BUFSEC pour copier en mémoire centrale le secteur suivant de la disquette, celui spécifié par le numéro d'unité d'allocation et la position du secteur dans celle-ci (ligne 1750).

Avant d'appeler la routine, `AL` doit contenir 0 si le tampon doit être chargé, 1 sinon, `BP` doit pointer sur le bloc des paramètres du lecteur de disquette, CLUSNUM contenir le numéro de l'unité d'allocation voulue, SECCLUSPOS la position du secteur voulu dans cette unité d'allocation et BYTCNT1 la taille du transfert, ce qui est le cas.

La routine BUFSEC regarde si le contenu du secteur spécifié se trouve dans le tampon de secteur, et le charge dans le cas contraire.

Après exécution de la routine, `SI` pointe sur le tampon du secteur, `DI` sur l'adresse de la zone de transfert, `CX` contient le nombre d'octets à transférer, NEXTADD est mis à jour et TRANS indique q'un transfert va être effectué.

— On fait pointer `DI` sur le tampon du secteur et `SI` sur l'adresse de la zone de transfert, puisqu'on va écrire sur le secteur (et non le lire), on sauvegarde sur la pile les adresses du segment des données et du segment supplémentaire, on confond le segment supplémentaire avec le segment de code, on prend comme adresse de segment des données le mot de poids fort de l'adresse de la zone de transfert, on divise le nombre d'octets à transférer par 2 puisqu'on va le faire mot par mot et non octet par octet, on commence par transférer un octet si le nombre d'octets à transférer est impair puis on transfère le nombre de mots nécessaires (lignes 1751–1761).

— On restaure les valeurs des adresses du segment des données et du segment supplémentaire, on indique qu'on a modifié le tampon de secteur depuis sa dernière sauvegarde sur la disquette et on termine la routine (lignes 1762–1765).

5.8.11.6 Terminaison de l'écriture séquentielle

La partie WRTEOF permet de terminer l'écriture séquentielle.

Avant de l'appeler, `CX` contient le nombre d'octets qu'il reste à écrire et `DX:AX` la position du premier octet à écrire :

```
2189 : WRTEOF:
2190 :          MOV     CX,AX
2191 :          OR      CX,DX
2192 :          JZ      KILLFIL
2193 :          SUB     AX,1
2194 :          SBB     DX,0
2195 :          DIV     [BP.SECSIZ]
2196 :          MOV     CL,[BP.CLUSSHFT]
2197 :          SHR     AX,CL
2198 :          MOV     CX,AX
2199 :          CALL    FNDCLUS
2200 :          JCXZ    RELFILE
2201 :          CALL    ALLOCATE
2202 :          JC      WRTERRJ
2203 : UPDATE:
2204 :          MOV     DI,[FCB]
2205 :          MOV     AX,WORD PTR [BYTPOS]
2206 :          MOV     ES:WORD PTR [DI.FILSIZ],AX
2207 :          MOV     AX,WORD PTR [BYTPOS+2]
2208 :          MOV     ES:WORD PTR [DI.FILSIZ+2],AX
2209 :          XOR     CX,CX
2210 :          JMP     ADDREC
2211 :
2212 : RELFILE:
2213 :          MOV     DX,OFFFH
2214 :          CALL    RELBLKS
2215 : SETDIRT:
2216 :          MOV     BYTE PTR [SI-1],1
2217 :          JMP     SHORT UPDATE
2218 :
2219 : KILLFIL:
2220 :          XOR     BX,BX
2221 :          XCHG    BX,ES:[DI.FIRCLUS]
2222 :          OR      BX,BX
2223 :          JZ      UPDATE
2224 :          CALL    RELEASE
2225 :          JMP     SHORT SETDIRT
```

— Si les contenus de `AX` et de `DX` sont logiquement disjoints, on va à la partie KILLFIL pour vider le fichier (lignes 2190–2192) :
  — On place 0 comme numéro de première unité d'allocation dans le FCB et on récupère le numéro de l'ancienne première unité d'allocation du fichier dans `BX`.
  — Si celui-ci était déjà zéro on va à la partie UPDATE pour faire pointer `DI` sur le FCB, placer la position de l'octet dans le secteur comme taille du fichier dans le FCB, placer 0 dans `CX` et aller à ADDREC pour mettre à jour la position de l'enregistrement (lignes 2220–2223 et 2203–2210).
  — Sinon on appelle la routine RELEASE pour libérer la chaîne des unités d'allocation à partir de celle contenue dans `BX`, on indique que la copie de la FAT en mémoire centrale a été modifiée depuis sa dernière sauvegarde sur la disquette et on va à la partie UPDATE comme dans le premier cas (ligne 2224–2225 et 2215–2217).
— Sinon on soustrait 1 à la position du premier octet à écrire, on divise par la taille d'un secteur ainsi que par le nombre de secteurs dans une unité d'allocation pour obtenir dans

`CX` la position dans le fichier de l'unité d'allocation sur laquelle se trouve l'octet à écrire (lignes 2193–2198).

— On appelle la routine FNDCLUS pour savoir s'il y a suffisamment de place sur le fichier, en lui allouant des unités d'allocation supplémentaires si besoin est, pour écrire jusque là (ligne 2199).

Celle-ci renvoie dans `CX` le nombre d'unités d'allocation nécessaires pour pouvoir écrire jusque là et dans `BX` le numéro de l'unité d'allocation à laquelle on est parvenu.

— Si `CX` est nul, c'est qu'il y a suffisamment de place. On place donc `FFh` dans `DX` et on appelle la routine RELBLKS, c'est-à-dire la routine RELEASE sans la remise à zéro de `DX`, pour indiquer que l'unité d'allocation dont le numéro est contenu dans `BX` est la dernière du fichier et pour libérer les autres unités d'allocation du fichier à partir de celle-ci, on indique que la copie de la FAT en mémoire centrale a été modifiée depuis sa dernière sauvegarde sur la disquette et on va à UPDATE (lignes 2200 et 2212–2217).

— Sinon on appelle la routine ALLOCATE pour essayer d'ajouter au fichier le nombre contenu dans `CX` d'unités d'allocation, renvoyer un code d'erreur si on n'y parvient pas ou continuer comme ci-dessus si on y parvient (lignes 2201–2202).

5.8.12 Routine de service de la fonction 33 (21h) de lecture directe dans un fichier spécifié par un FCB

5.8.12.1 Routine principale

La routine de service de la fonction 33 de l'interruption `21h`, de lecture directe dans un fichier spécifié par un FCB, est repérée par l'étiquette RNDRD :

```
1410 : RNDRD:  ;System call 33
1411 :         CALL    GETRRPOS1
1412 :         CALL    LOAD
1413 :         JMP     SHORT FINRND
```

— `DS:DX` pointant sur un FCB ouvert, on appelle la routine GETRRPOS1, étudiée ci-dessous, pour placer dans `DX:AX` le numéro absolu de l'enregistrement à lire, tel que spécifié par le champ RR du FCB.

— On appelle ensuite la routine LOAD pour transférer cet enregistrement dans la zone de transfert, de la même façon que lors d'une lecture séquentielle.

— On passe à la partie FINRND, étudiée ci-dessous, pour mettre à jour les champs RR, NR et EXTENT du FCB.

5.8.12.2 Récupération du numéro d'enregistrement à lire

La récupération du numéro d'enregistrement à lire est effectué par la routine GETRRPOS1 :

```
1453 : GETRRPOS1:
1454 :         MOV     CX,1
1455 : GETRRPOS:
1456 :         MOV     DI,DX
1457 :         CMP     BYTE PTR [DI],-1
1458 :         JNZ     NORMFCB1
1459 :         ADD     DI,7
1460 : NORMFCB1:
1461 :         MOV     AX,WORD PTR [DI.RR]
1462 :         MOV     DX,WORD PTR [DI.RR+2]
1463 :         RET
```

— On initialise `CX` à 1 pour indiquer que l'on veut lire un seul enregistrement (ligne 1454).

— On fait pointer `DI` sur le FCB. S'il s'agit d'un FCB étendu, on le fait pointer sur le FCB standard associé (lignes 1455–1460).

— On place, pour terminer, le numéro depuis le début du fichier de l'enregistrement à lire dans `DX:AX` (lignes 1461–1463).

5.8.12.3 Incrémentation du numéro d'enregistrement

La routine FINRND (fin d'un opération directe) incrémente le numéro d'enregistrement en opération directe dans le FCB du fichier, ce qui peut être utile si l'on continue par une opération séquentielle.

Lors de l'appel de cette routine, `DX:AX` doit contenir le numéro absolu de l'enregistrement à placer et `ES:DI` pointer sur le FCB :

```
1434 : FINRND:
1435 :          MOV     ES:WORD PTR [DI.RR],AX
1436 :          MOV     ES:[DI.RR+2],DL
1437 :          OR      DH,DH
1438 :          JZ      SETNREX
1439 :          MOV     ES:[DI.RR+3],DH ;Save 4 byte of RECPOS only if significant
1440 : SETNREX:
```

c'est-à-dire que l'on place le numéro d'enregistrement suivant celui qui vient d'être lu dans le champ RR du FCB et que l'on va à la partie SETNREX, déjà étudiée à propos de FINSEQ, pour le placer également dans les NR et EXTENT du FCB pour une lecture séquentielle.

5.8.13 Routine de service de la fonction 34 (22h) d'écriture directe dans un fichier spécifié par un FCB

La routine de service de la fonction 34 de l'interruption `21h`, d'écriture directe dans un fichier spécifié par un FCB, est repérée par l'étiquette RNDWRT :

```
1415 : RNDWRT: ;System call 34
1416 :          CALL    GETRRPOS1
1417 :          CALL    STORE
1418 :          JMP     SHORT FINRND
```

- `DS:DX` pointant sur un FCB ouvert, on appelle la routine GETRRPOS1 pour placer dans `DX:AX` le numéro absolu de l'enregistrement qu'il faut écrire, tel que spécifié par le champ RR du FCB.
- On appelle ensuite la routine STORE pour transférer cet enregistrement depuis la zone de transfert, de la même façon que lors d'une écriture séquentielle.
- Puis on passe à la partie FINRND pour remplir les champs RR, NR et EXTENT du FCB.

5.8.14 Routine de service de la fonction 36 (24h) d'initialisation du champ RR d'un FCB

La routine de service de la fonction 36 de l'interruption `21h`, d'initialisation du champ RR d'un FCB, est repérée par l'étiquette SETRNDREC :

```
2688 : SETRNDREC: ;System call 36
2689 :         CALL    GETREC
2690 :         MOV     [DI+33],AX
2691 :         MOV     [DI+35],DL
2692 :         CMP     [DI.RECSIZ],64
2693 :         JAE     RET15
2694 :         MOV     [DI+36],DH      ;Set 4th byte only if record size < 64
2695 : RET16:  RET
```

— `DS:DX` pointant sur un FCB, on appelle la routine GETREC pour placer dans `DX:AX` le numéro absolu de l'enregistrement index et faire pointer `DS:DI` sur le FCB, plus précisément sa partie standard s'il s'agit d'un FCB étendu (ligne 2689).

— On place le contenu de `AX` dans le premier mot du champ RR du FCB, celui de `DL` dans l'octet suivant et, si la taille d'un enregistrement est inférieure à 64, celui de `DH` dans l'octet suivant, puis on termine la routine (lignes 2690–2695).

5.8.15 Routine de service de la fonction 39 (27h) de lecture directe de plusieurs enregistrements

La routine de service de la fonction 39 de l'interruption `21h`, de lecture directe de plusieurs enregistrements, est repérée par l'étiquette BLKRND :

```
1420 : BLKRD:  ;System call 39
1421 :         CALL    GETRRPOS
1422 :         CALL    LOAD
1423 :         JMP     SHORT FINBLK
1424 :
1425 : BLKWRT: ;System call 40
1426 :         CALL    GETRRPOS
1427 :         CALL    STORE
1428 : FINBLK:
1429 :         LDS     SI,DWORD PTR [SPSAVE]
1430 :         MOV     [SI.CXSAVE],CX
1431 :         JCXZ    FINRND
1432 :         ADD     AX,1
1433 :         ADC     DX,0
1434 : FINRND:
```

— `DS:DX` pointant sur un FCB et `CX` contenantt un nombre non nul d'enregistrements, on appelle la routine GETRRPOS pour placer dans `DX:AX` le numéro absolu du premier enregistrement à lire, tel que spécifié par le champ RR du FCB (ligne 1421).

— On appelle ensuite la routine LOAD pour transférer le nombre voulu d'enregistrements dans la zone de transfert, de la même façon que lors d'une lecture séquentielle (ligne 1422).

— On sauvegarde sur la pile le nombre d'enregistrements non lus et, si celui-ci est non nul, on place le numéro d'enregistrement suivant dans `DX:AX` (lignes 1423 et 1428–1433).

— Puis on passe à la partie FINRND pour mettre à jour les champs RR, NR et EXTENT du FCB (ligne 1434).

5.8.16 Routine de service de la fonction 40 (28h) d'écriture directe de plusieurs enregistrements

La routine de service de la fonction 40 de l'interruption `21h`, d'écriture directe de plusieurs enregistrements, est repérée par l'étiquette BLKWRT :

```
1425:BLKWRT: ;System call 40
1426:        CALL    GETRRPOS
1427:        CALL    STORE
1428:FINBLK:
1429:        LDS     SI,DWORD PTR [SPSAVE]
1430:        MOV     [SI.CXSAVE],CX
1431:        JCXZ    FINRND
1432:        ADD     AX,1
1433:        ADC     DX,0
1434:FINRND:
```

— `DS:DX` pointant sur un FCB et `CX` contenantt un nombre non nul d'enregistrements, on appelle la routine GETRRPOS pour placer dans `DX:AX` le numéro absolu du premier enregistrement à écrire, celui spécifié par le champ RR du FCB (ligne 1426).

— On appelle ensuite la routine STORE pour enregister sur la disquette les enregistrements se trouvant dans la zone de transfert, de la même façon que lors d'une écriture séquentielle (ligne 1427).

— On sauvegarde sur la pile le nombre d'enregistrements non écrits et, si celui-ci est non nul, on place le numéro d'enregistrement suivant dans `DX:AX` (lignes 1423 et 1428–1433).

— Puis on passe à la partie FINRND pour mettre à jour les champs RR, NR et EXTENT du FCB (ligne 1434).

5.8.17 Routine de service de la fonction 35 (23h) de détermination de la taille d'un fichier

La routine de service de la fonction 35 de l'interruption `21h`, de détermination de la taille d'un fichier spécifié par un FCB, est repérée par l'étiquette FILESIZE :

```
2573:FILESIZE: ;System call 35
2574:        CALL    GETFILE
2575:        MOV     AL,-1
2576:        JC      RET14
2577:        ADD     DI,33           ;Write size in RR field
2578:        MOV     CX,ES:[DI.RECSIZ-33]
2579:        OR      CX,CX
2580:        JNZ     RECOK
2581:        MOV     CX,128
2582:RECOK:
2583:        XOR     AX,AX
2584:        XOR     DX,DX           ;Intialize size to zero
2585:        OR      BH,BH           ;Check for named I/O device
2586:        JS      DEVSIZ
2587:        INC     SI
2588:        INC     SI              ;Point to length field
2589:        MOV     AX,[SI+2]       ;Get high word of size
2590:        DIV     CX
2591:        PUSH    AX              ;Save high part of result
2592:        LODSW                   ;Get low word of size
2593:        DIV     CX
2594:        OR      DX,DX           ;Check for zero remainder
2595:        POP     DX
```

```
2596:         JZ      DEVSIZ
2597:         INC     AX              ;Round up for partial record
2598:         JNZ     DEVSIZ          ;Propagate carry?
2599:         INC     DX
2600:DEVSIZ:
2601:         STOSW
2602:         MOV     AX,DX
2603:         STOSB
2604:         MOV     AL,0
2605:         CMP     CX,64
2606:         JAE     RET14           ;Only 3-byte field if RECSIZ >= 64
2607:         MOV     ES:[DI],AH
2608:         RET
```

— On appelle la routine GETFILE pour rechercher l'entrée dans le répertoire du fichier dont le FCB est passé en paramètre à l'adresse `DS:DX` (ligne 2574).

Le drapeau CF est levé s'il n'existe pas de fichier de nom correspondant dans le répertoire. Sinon le drapeau de zéro est baissé, `BP` pointe sur le bloc des paramètres du lecteur de disquette concerné, les segments de données et supplémentaire sont confondus avec le segment de code, `SI` contient le numéro de la première unité d'allocation du fichier, DIRBUF pointe sur l'entrée de répertoire correspondante dans la copie du secteur du répertoire se trouvant en mémoire centrale, la vaeiable chaîne de caractères NAME1 contient le nom du fichier, tout en majuscules, `ES:DI` pointe sur le FCB et `BX` pointe sur le tampon du secteur du répertoire en mémoire centrale.

— On place le code d'erreur `FFh` dans `AL` et, si on n'a trouvé aucune entrée concordante, on termine la routine (lignes 2575–2576).

— Sinon on ajoute 33 à `DI` pour qu'il pointe sur le champ RR du FCB et on place la taille d'un enregistrement dans `CX`. Si cette taille n'a pas été initialisée, c'est-à-dire si elle est nulle, on l'initialise à la valeur par défaut, à savoir 128 (lignes 2577–2582).

— On initialise la taille du fichier à zéro (lignes 2583–2584).

— S'il ne s'agit pas d'un fichier de périphérique, on fait pointer `SI` sur le champ 'taille du fichier', on place dans `AX` le mot de poids fort de la taille, que l'on divise par la taille d'un enregistrement et dont on sauvegarde le quotient sur la pile, on place ensuite dans `AX` le mot de poids faible de la taille du fichier, que l'on divise également par la taille d'un enregistrement. Si le reste en est nul, on place le quotient sauvegardé sur la pile dans `DX`, sinon on incrémente `AX` pour obtenir la partie entière plus une et on tient compte de la retenue (lignes 2585–2600).

— On place la valeur de `AX` dans le premier mot du champ RR du FCB, celle de `DL` dans l'octet suivant et, si la taille d'un secteur est inférieure à 64, celle de `DH` dans l'octet suivant puis on termine la routine (lignes 2601–2608).

5.8.18 Routine de service de la fonction 41 (29h) de création d'un FCB

5.8.18.1 Routine principale

La routine de service de la fonction 41 de l'interruption `21h`, de création d'un FCB est repérée par l'étiquette MAKEFCB :

```
3189 : MAKEFCB: ;Interrupt call 41
3190 : DRVBIT  EQU     2
3191 : NAMBIT  EQU     4
3192 : EXTBIT  EQU     8
3193 :         MOV     DL,0                    ;Flag--not ambiguous file name
3194 :         TEST    AL,DRVBIT               ;Use current drive field if default?
3195 :         JNZ     DEFDRV
3196 :         MOV     BYTE PTR ES:[DI],0      ;No - use default drive
3197 : DEFDRV:
3198 :         INC     DI
3199 :         MOV     CX,8
3200 :         TEST    AL,NAMBIT               ;Use current name fiels as defualt?
3201 :         XCHG    AX,BX                   ;Save bits in BX
3202 :         MOV     AL," "
3203 :         JZ      FILLB                   ;If not, go fill with blanks
3204 :         ADD     DI,CX
3205 :         XOR     CX,CX                   ;Don't fill any
3206 : FILLB:
3207 :         REP     STOSB
3208 :         MOV     CL,3
3209 :         TEST    BL,EXTBIT               ;Use current extension as default
3210 :         JZ      FILLB2
3211 :         ADD     DI,CX
3212 :         XOR     CX,CX
3213 : FILLB2:
3214 :         REP     STOSB
3215 :         XCHG    AX,CX                   ;Put zero in AX
3216 :         STOSW
3217 :         STOSW                           ;Initialize two words after to zero
3218 :         SUB     DI,16                   ;Point back at start
3219 :         TEST    BL,1                    ;Scan off separators if not zero
3220 :         JZ      SKPSPC
3221 :         CALL    SCANB                   ;Peel off blanks and tabs
3222 :         CALL    DELIM                   ;Is it a one-time-only delimiter?
3223 :         JNZ     NOSCAN
3224 :         INC     SI                      ;Skip over the delimiter
3225 : SKPSPC:
3226 :         CALL    SCANB                   ;Always kill preceding blanks and tabs
3227 : NOSCAN:
3228 :         CALL    GETLET
3229 :         JBE     NODRV                   ;Quit if termination character
3230 :         CMP     BYTE PTR[SI],":"        ;Check for potential drive specifier
3231 :         JNZ     NODRV
3232 :         INC     SI                      ;Skip over colon
3233 :         SUB     AL,"@"                  ;Convert drive letter to binary drive number
3234 :         JBE     BADDRV                  ;Valid drive numbers are 1-15
3235 :         CMP     AL,CS:[NUMDRV]
3236 :         JBE     HAVDRV
3237 : BADDRV:
3238 :         MOV     DL,-1
3239 : HAVDRV:
3240 :         STOSB                           ;Put drive specifier in first byte
3241 :         INC     SI
3242 :         DEC     DI                      ;Counteract next two instructions
3243 : NODRV:
3244 :         DEC     SI                      ;Back up
```

```
3245 :          INC     DI                              ;Skip drive byte
3246 :          MOV     CX,8
3247 :          CALL    GETWORD                         ;Get 8-letter file name
3248 :          CMP     BYTE PTR [SI],"."
3249 :          JNZ     NODOT
3250 :          INC     SI                              ;Skip over dot if present
3251 :          MOV     CX,3                            ;Get 3-letter extension
3252 :          CALL    MUSTGETWORD
3253 : NODOT:
3254 :          LDS     BX,CS:DWORD PTR [SPSAVE]
3255 :          MOV     [BX.SISAVE],SI
3256 :          MOV     AL,DL
3257 :          RET
3258 :
3259 : NONAM:
3260 :          ADD     DI,CX
3261 :          DEC     SI
3262 :          RET
3263 :
3264 : GETWORD:
3265 :          CALL    GETLET
3266 :          JBE     NONAM                           ;Exit if invalid character
3267 :          DEC     SI
3268 : MUSTGETWORD:
3269 :          CALL    GETLET
3270 :          JBE     FILLNAM
3271 :          JCXZ    MUSTGETWORD
3272 :          DEC     CX
3273 :          CMP     AL,"*"                          ;Check for ambiguous file specifier
3274 :          JNZ     NOSTAR
3275 :          MOV     AL,"?"
3276 :          REP     STOSB
3277 : NOSTAR:
3278 :          STOSB
3279 :          CMP     AL,"?"
3280 :          JNZ     MUSTGETWORD
3281 :          OR      DL,1                            ;Flag ambiguous file name
3282 :          JMP     MUSTGETWORD
3283 : FILLNAM:
3284 :          MOV     AL," "
3285 :          REP     STOSB
3286 :          DEC     SI
3287 : RET21:   RET
```

`DS:SI` pointe sur une ligne de commande, `ES:DI` sur la zone de mémoire à remplir par le FCB non ouvert et `AL` contient l'action à effectuer.

Première phase de remplissage du FCB

— On place `00h` dans `DL` pour indiquer que, pour l'instant, le nom du fichier n'est pas ambigu (ligne 3193).

— Si le bit 1 de `AL` est 1, il ne faut initialiser l'octet de numéro de disquette du FCB que si un lecteur de disquette est spécifié sur la ligne de commande. On place donc dans ce cas `00h` sur cet octet dans le FCB (lignes 3194–3197).

— On incrémente `DI` pour qu'il pointe sur le premier octet du champ 'nom de fichier' du FCB, on place la longueur d'un nom de fichier sans son extension, à savoir 8, dans `CX`, l'octet de commande dans `BX` et le caractère espace dans `AL` (lignes 3198–3202).

— Si le premier caractère du nom de fichier spécifié par la ligne de commande est un espace, on remplit le champ 'nom de fichier' du FCB avec des espaces. Sinon on passe au champ

suivant du FCB sans le renseigner. Dans tous les cas on se retrouve avec `CX` nul et `DI` pointant sur le champ ‘extension’ du FCB (lignes 3203–3207).

— On place la longueur de l’extension, à savoir 3, dans `CL` et, sauf s’il ne faut pas changer l’extension, on remplit le champ ‘extension’ du FCB avec des caractères espace. Dans les deux cas on se retrouve avec `CX` nul et `DI` pointant sur le champ ‘bloc en cours’ du FCB (lignes 3208–3214).

— On initialise à zéro les deux champs ‘bloc en cours’ et ‘taille d’un enregistrement’ du FCB (lignes 3215–3217).

— On revient au début du FCB. Si l’octet de commande demande de retirer les séparateurs, on appelle les routines SCANB et DELIM, étudiées ci-dessous, pour passer les caractères délimiteurs de la ligne de commande (lignes 3218–3227).

— On place `FFh` dans `DL` et l’identificateur du lecteur dans le premier octet du FCB (lignes 3238–3240).

— On incrémente `SI` et on décrémente `DI` pour aller dans le sens contraire des deux instructions suivantes (lignes 3241–3242).

—

5.8.18.2 Élimination des espaces de la ligne de commande

La routine SCANB élimine les caractères espace et tabulation de la ligne de commande :

```
3289 : SCANB:
3290 :          LODSB
3291 :          CALL    SPCHK
3292 :          JZ      SCANB
3293 :          DEC     SI
3294 :          RET
```

c'est-à-dire qu'on recherche en arrière le premier caractère espace et tabulation de la ligne de commande à partir de la position sur laquelle on se trouve, et qu'on lève le drapeau CF lorsqu'on en a trouvé un, `DI` pointant alors sur celui-ci.

La routine SPCHK, fin de la routine GETLET déjà étudiée :

```
3335 : SPCHK:
3336 :          CMP     AL,9            ;Filter out tabs too
3337 :          JZ      RET101
3338 : ;WARNING! " " MUST be the last compare
3339 :          CMP     AL," "
3340 : RET101: RET
```

lève le drapeau CF si `AL` contient le code ASCII de la tabulation ou de l'espace.

La routine DELIM, également fin de la routine GETLET déjà étudiée :

```
3318 :          IF      IBM
3319 : DELIM:
3320 :          ENDIF
3321 :          CMP     AL,":"          ;Allow ":" as separator in IBM version
3322 :          JZ      RET21
3323 :          IF      NOT IBM
3324 : DELIM:
3325 :          ENDIF
3326 :
3327 :          CMP     AL,"+"
3328 :          JZ      RET101
3329 :          CMP     AL,"="
3330 :          JZ      RET101
3331 :          CMP     AL,";"
3332 :          JZ      RET101
3333 :          CMP     AL,","
3334 :          JZ      RET101
3335 : SPCHK:
3336 :          CMP     AL,9            ;Filter out tabs too
3337 :          JZ      RET101
3338 : ;WARNING! " " MUST be the last compare
3339 :          CMP     AL," "
3340 : RET101: RET
```

lève le drapeau ZF si `AL` contient le code ASCII de l'un des caractères '+', '=', ';', ',', tabulation ou espace.

5.9 Routines de service de gestion des programmes

5.9.1 Routines de service de la fonction 00h et de l'interruption 20h de terminaison d'un programme

Nous les étudions ensemble puisque la fonction `00h` de l'interruption `21h` effectue exactement la même chose que l'interruption `20h`.

5.9.1.1 Routine de service de la fonction 00h de terminaison d'un programme

La routine de service de la fonction `00h` de l'interruption `21h`, de terminaison d'un programme, est repérée par l'étiquette ABORT :

```
1356 : ABORT:
1357 :          LDS     SI,CS:DWORD PTR [SPSAVE]
1358 :          MOV     DS,[SI.CSSAVE]
1359 :          XOR     AX,AX
1360 :          MOV     ES,AX
1361 :          MOV     SI,SAVEXIT
1362 :          MOV     DI,EXIT
1363 :          MOVSW
1364 :          MOVSW
1365 :          MOVSW
1366 :          MOVSW
1367 :          MOVSW
1368 :          MOVSW
1369 : ERROR:
```

Le registre `CS` contient l'adresse du segment dans lequel se trouvent le bloc de contrôle et le PSP.

— On place dans `SI` le pointeur de pile sauvegardé dans la variable SPSAVE, pile commençant par une structure STKPTRS, dans `DS` l'adresse de segment de code sauvegardée, on fait commencer le segment supplémentaire au tout début de la mémoire centrale, on place dans `SI` le décalage 10, c'est-à-dire le décalage dans le PSP de l'adresse de terminaison du programme, pour que `DS:SI` pointe sur celle-ci, et dans `DI` le décalage de l'étiquette EXIT, correspondant au vecteur de l'interruption `22h`, pour que `ES:DI` pointe sur celui-ci, et on copie les 12 octets des vecteurs d'interruption des interruptions `22h`, `23h` et `24h` (lignes 1357–1368).

— On continue par la partie ERROR déjà étudiée.

On confond les segment des données et supplémentaire avec le segment de code. On appelle la routine WRTFATS pour enregistrer sur la disquette la copie de la FAT se trouvant en mémoire centrale ainsi que la copie du secteur se trouvant dans le tampon de secteur en mémoire centrale, si on ne l'avait pas fait depuis les dernières modifications de ceux-ci. On inhibe les interruptions masquables, on revient à la pile antérieure, on fait commencer le segment des données au tout début de la mémoire centrale, on fait pointer `SI` sur le vecteur d'interruption de l'interruption `22h`, on place dans `DI` l'adresse de la variable EXITHOLD, dans laquelle on place les 4 octets du vecteur d'interruption de l'interruption `22h`. On restaure les valeurs des registres, on se remet à l'écoute des interruptions masquables et on appelle l'interruption `22h` pour terminer le programme en cours.

Rappelons que la routine de service de l'interruption `22h` de terminaison d'un programme ne sera implémentée qu'avec l'interpréteur de commandes.

5.9.1.2 Routine de service de l'interruption 20h de terminaison d'un programme

Nous avons vu que la routine de service de l'interruption 20h est repérée par l'étiquette QUIT :

```
266 : QUIT:
267 :         MOV     AH,0
268 :         JMP     SHORT SAVREGS
```

Elle place 0 dans le registre AH pour indiquer la fonction 00h de l'interruption 21h et on saute à l'étiquette SAVREGS, qui se trouve au début de ENTRY, routine de service de l'interruption 21h. Elle effectue donc bien exactement la même chose que la fonction 00h de l'interruption 21h.

5.9.2 Routine de service de l'interruption 00h de division par zéro

Nous avons vu que MS-DOS surcharge la routine de service de IO.COM concernant la division par zéro, à savoir l'interruption 00h. La nouvelle routine de service de cette interruption est repérée par l'étiquette DIVOV (pour *DIVision OVerflow*) :

```
3618 : ; Default handler for division overflow trap
3619 : DIVOV:
3620 :         PUSH    SI
3621 :         PUSH    AX
3622 :         MOV     SI,OFFSET DOSGROUP:DIVMES
3623 :         CALL    OUTMES
3624 :         POP     AX
3625 :         POP     SI
3626 :         INT     23H             ;Use Ctrl-C abort on divide overflow
3627 :         IRET
```

On sauvegarde sur la pile les contenus des registres que l'on va utiliser, SI et AX, on affiche un message d'erreur, on restaure les contenus des registres sauvegardés et on appelle l'interruption 23h d'arrêt d'un programme.

Le message d'erreur de division par zéro est DIVMES :

```
3645 : DIVMES  DB      13,10,"Divide overflow",13,10,"$"
```

La sous-routine d'affichage d'un message d'erreur est OUTMES :

```
3170 : RET20:  RET
[...]
3181 : OUTMES: ;String output for internal messages
3182 :         LODS    CS:BYTE PTR [SI]
3183 :         CMP     AL,"$"
3184 :         JZ      RET20
3185 :         CALL    OUT
3186 :         JMP     SHORT OUTMES
```

Le message est une chaîne de caractères se terminant, convention de MS-DOS, par le caractère '$'. On affiche donc tous les caractères de cette chaîne de caractères jusqu'à ce qu'on rencontre '$', que l'on n'affiche pas.

5.9.3 Routine de service de la fonction 38 (26h) de création d'un nouveau PSP

La routine de service de la fonction 38 de l'interruption `21h`, de création d'un nouveau PSP, est repérée par l'étiquette NEWBASE :

```
3353 : NEWBASE: ; Interrupt call 38
3354 :         MOV     ES,DX
3355 :         LDS     SI,CS:DWORD PTR [SPSAVE]
3356 :         MOV     DS,[SI.CSSAVE]
3357 :         XOR     SI,SI
3358 :         MOV     DI,SI
3359 :         MOV     AX,DS:[2]
3360 :         MOV     CX,80H
3361 :         REP     MOVSW
3362 :
3363 : SETMEM:
```

— À l'appel de la fonction, `DX` contient l'adresse de segment du début du programme.

— On confond le segment supplémentaire avec le segment du début du programme (lignes 3354).

— On place dans `SI` la valeur du pointeur de pile sauvegardé (ligne 3355).

— On confond le segment des données avec le segment de code sauvegardé (3356).

— On initialise `SI` et `DI` à zéro, `AX` à la valeur du troisième octet du segment des données, donc ? ?, `CX` à `80h`, décalage du PSP donnant la longueur, en octets, de la ligne de commande, et on recopie la ligne de commande (lignes 3357–3361).

— On continue par la routine SETMEM de construction d'un PSP au début du segment spécifié en paramètre (ligne 3363).

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

Après son exécution, les adresses des segments des données et supplémentaire sont l'adresse de segment spécifiée, et les 22 premiers octets de ce segment sont renseignés pour constituer un PSP.