SEDORIC 3.0
à NU

SEDORIC et STRATORIC
Versions 3.0 du 01/01/96

Deuxième Partie (Pages 232 à 459)

André Chéramy
54, rue de Sours 28000 CHARTRES
cheramy@infobiogen.fr

Troisième Edition (1998)

Table des matières

Première Partie (pages 1-231)
Avant-propos 3
Comment lire ce livre 4
Nouveautés de la version 3.0 5

La RAM overlay 7
Analyse des commandes SEDORIC 7

Buffer 1 (BUF1) 16
Buffer 2 (BUF2) 17
Buffer 3 (BUF3) 18

BANQUE n°0 19
Initialisation SEDORIC 19
Source de la page 4 version ORIC-1 25
Source de la page 4 version ATMOS 25
Désassemblage de la page 4 SEDORIC 26

BANQUES interchangeables 30
BANQUE n°1 (adresse Cxxx**a**): RENUM, DELETE et MOVE 30
BANQUE n°2 (adresse Cxxx**b**): BACKUP 51
BANQUE n°3 (adresse Cxxx**c**): SEEK, CHANGE et MERGE 68
BANQUE n°4 (adresse Cxxx**d**): COPY 89
BANQUE n°5 (adresse Cxxx**e**): SYS, DNAME, DTRACK, TRACK, INIST, DNUM, DSYS, DKEY et VUSER 103
BANQUE n°6 (adresse Cxxx**f**): INIT 123
BANQUE n°7 (adresse Cxxx**g**): CHKSUM, EXT, PROT, STATUS, SYSTEM ,UNPROT et VISUHIRES 144

Début du NOYAU permanent de SEDORIC (#C800 à #FFFF) 161
Mots Clés SEDORIC 167

XRWTS Routine de gestion des lecteurs 179

Série d'appels à des sous-programmes en ROM 187
Routines SEDORIC d'usage général . . . . . 197, 212 et 236
Routines principales de Ray McLaughlin 292

Entrée SEDORIC: recherche l'adresse d'exécution d'un mot-clé SEDORIC 199
Analyse d'un nom de fichier 203
Prendre un caractère au clavier (remplace EB78 ROM) 221

Deuxième Partie (pages 232-459)
Commandes SEDORIC (avec quelques routines associées, d'usage général) . . . . . 232 et 251
Commandes SEDORIC faisant appel à une BANQUE externe 356

Note sur les coordonnées colonne/ligne ORIC-1 / ATMOS / SEDORIC 363

Gestion de fichiers 374

Table des vecteurs système (#FF43-#FFC6) 456

Copyrights . . . . . 6, 146, 458, 461, 465, 484 à 486 et 488

Troisième Partie (pages 460-630)
ANNEXES 460

ANNEXE n° 1: SEDORIC V2.0 461
ANNEXE n° 2: SEDORIC V2.0 463
ANNEXE n° 3: SEDORIC V2.0 465
ANNEXE n° 4: PATCH 001 475
ANNEXE n° 5: PATCH 002 478
ANNEXE n° 6: Que se passe t-il lors du boot ? 482
ANNEXE n° 7: Rappel de la structure des disquettes SEDORIC 484
ANNEXE n° 8: Que se passe t-il lors d'un SAVE ? 504
ANNEXE n° 9: Que se passe t-il lors d'un DEL ? 511
ANNEXE n° 10: Listing de l'EPROM du MICRODISC 517
ANNEXE n° 11: Le FDC 1793 549
ANNEXE n° 12: F.A.Q 560
ANNEXE n° 13: Exercices de passage ROM <--> RAM overlay 562
ANNEXE n° 14: Utilisation d'une commande SEDORIC sans argument (programme LM) 564
ANNEXE n° 15: Utilisation d'une routine en RAM overlay (programme LM) 565
ANNEXE n° 16: Utilisation d'une commande SEDORIC avec paramètres (programme LM) 566
ANNEXE n° 17: Les bogues de SEDORIC 569
ANNEXE n° 18: Mots clés SEDORIC 575
ANNEXE n° 19: Les Codes de Fonctions 577
ANNEXE n° 20: Futures extensions 584
ANNEXE n° 21: Routines d'intérêt général (par ordre chronologique) 587
ANNEXE n° 22: Routines d'intérêt général (par thèmes) 597
ANNEXE n° 23: Des drives et des DOS pour ORIC 611
ANNEXE n° 24: Directories des disquettes SEDORIC V3.006 et TOOLS V3.006 623
ANNEXE n° 25: Tables et figures 626
ANNEXE n° 26: Table des matières 628

SUITE DES COMMANDES SEDORIC

EXÉCUTION DE LA COMMANDE SEDORIC KEYUSE

Rappel de la syntaxe

KEYUSE code_de_la_commande,chaîne_de_définition_de_la_commande

Permet d'associer une ou plusieurs commandes BASIC et/ou SEDORIC (16 caractères alphanumériques au maximum) à un code de fonctions re-definissables (de 0 à 15). Cette commande, qui ne marche qu'avec l'ATMOS, accepte tous les caractères ASCII sauf #00 et les codes >127. Les définitions initiales peuvent être consultées dans la table des codes de fonctions en C880.

Voir un exemple de l'utilisation de cette commande dans le "Non documenté" de la commande ACCENT (en EB91).

Analyse de la syntaxe et saisie des paramètres

D9B0-	20 7F D2	JSR D27F	CF17/ROM et CF09/ROM et D8CB/ROM évalue un nombre entier à TXTPTR et le retourne dans X (code de la commande utilisateur)
D9B3-	E0 10	CPX #10	X est-il >= #10?
D9B5-	B0 3E	BCS D9F5	si oui, continue en D9F5
D9B7-	8A	TXA	sinon, c'est une commande re-definissable (code de
D9B8-	0A	ASL	#00 à #0F) qui passe dans A et dont les 4 bits
D9B9-	0A	ASL	faibles (b0 à b3) sont "shiftés" dans les
D9BA-	0A	ASL	4 bits forts pour obtenir un index permettant
D9BB-	0A	ASL	d'accéder aux différentes entrées
D9BC-	48	PHA	de la table C880 enfin le résultat est empilé
D9BD-	20 2C D2	JSR D22C	D067/ROM exige une "," place TXTPTR sur l'octet suivant
D9C0-	20 24 D2	JSR D224	JSR CF17/ROM évalue une expression à TXTPTR, retourne avec une valeur numérique dans ACC1 (et #00 dans 28) ou l'adresse d'une chaîne dans D3/D4 (et #FF dans 28) et A, N et Z positionnés selon l'exposant (résultat numérique) ou la longueur de chaîne
D9C3-	20 74 D2	JSR D274	JSR D7D0 et CF09/ROM vérifie si l'expression évaluée à TXTPTR est bien alphanumérique, retourne l'adresse de la chaîne dans 91/92 et sa longueur dans A
D9C6-	C9 11	CMP #11	la longueur est-elle >= #11 (17)
D9C8-	B0 2E	BCS D9F8	si oui, continue en D9F8 (erreur car 16 au maximum)
D9CA-	A8	TAY	teste encore la longueur
D9CB-	F0 2B	BEQ D9F8	la longueur est-elle nulle (erreur car 1 minimum)
D9CD-	68	PLA	récupère le n° de code "shifté" et le passe
D9CE-	AA	TAX	dans X (index d'entrée dans la table C880)
D9CF-	A9 10	LDA #10	
D9D1-	85 F2	STA F2	F2 = #10 pour copier 16 caractères
D9D3-	A9 20	LDA #20	efface la commande actuelle en remplissant
D9D5-	9D 80 C8	STA C880,X	la chaîne correspondante avec des espaces

D9D8-	E8	INX	emplacement suivant dans la chaîne
D9D9-	C6 F2	DEC F2	décrémente le nombre de copies à faire
D9DB-	D0 F8	BNE D9D5	et reboucle en D9D5 tant qu'il en reste
D9DD-	88	DEY	Y = longueur de la chaîne - 1
D9DE-	CA	DEX	X = pointeur dans la chaîne cible, ajusté sur dernière position
D9DF-	B1 91	LDA (91),Y	lit le dernier caractère de la chaîne saisie
D9E1-	09 80	ORA #80	force b7 à 1 (marque le dernier caractère) et
D9E3-	9D 80 C8	STA C880,X	le met en place en dernière position dans table
D9E6-	CA	DEX	position précédente dans la table C880
D9E7-	88	DEY	nombre de caractères restant à copier
D9E8-	30 35	BMI DA1F	simple RTS s'il n'en reste plus
D9EA-	B1 91	LDA (91),Y	lit un caractère de la chaîne (par la fin)
D9EC-	F0 07	BEQ D9F5	caractère nul interdit, branche en D9F5 (erreur)
D9EE-	30 05	BMI D9F5	caractère >= #80 (token) interdit, branche idem en D9F5 (erreur). Quel dommage d'interdire les tokens, voilà qui aurait pu éviter un travail inutile de décodage lors de l'utilisation de FUNCT (+SHIFT) + touche et qui aurait permis de "caser" davantage d'instructions car 16 caractères c'est parfois un peu juste!
D9F0-	9D 80 C8	STA C880,X	et le met en place dans la chaîne de la table
D9F3-	90 F1	BCC D9E6	rebouclage forcé en D9E6 (C mis à 0 en D9C6)
D9F5-	A2 08	LDX #08	pour "ILLEGAL_QUANTITY_ERROR"
D9F7-	2C A2 12	BIT 12A2	et continue en D9FA
D9F8-	A2 12	LDX #12	pour "STRING_TOO_LONG_ERROR"
D9FA-	4C 7E D6	JMP D67E	incrémente X et traite l'erreur n° X

EXÉCUTION DE LA COMMANDE SEDORIC KEYDEF

Rappel de la syntaxe

KEYDEF n°_de_code_de_fonction

Permet d'assigner le code de fonction indiqué (de 0 à 255, voir leur liste en ANNEXE) à la ou les touches qui seront pressée immédiatement après. Si une seule touche est pressée, la fonction correspondante sera accessible à l'aide de la combinaison FUNCT+touche. Si les touches SHIFT+touche sont pressées, la fonction correspondante sera accessible à l'aide de la combinaison FUNCT+SHIFT+touche. La définition originale des codes de fonctions de 0 à 31 est donnée dans la table des codes de fonctions en C880. Les codes de 32 à 127 correspondent aux mots-clés de SEDORIC (voir aussi le manuel page 103). Les codes de 128 à 253 correspondent aux tokens BASIC. Enfin les codes 254 et 255 correspondent respectivement à DEL et à la génération des n° de ligne. Les affectations d'origine sont données dans la table "KEYDEF" en C800. Voir un exemple de l'utilisation de cette commande dans le "Non documenté" de la commande ACCENT (en EB91).

Saisie du paramètre et exécution de la commande

D9FD-	20 7F D2	JSR D27F	CF17/ROM et CF09/ROM et D8CB/ROM évalue un nombre entier à TXTPTR et le retourne dans X (code de fonction)
DA00-	4E DF 02	LSR 02DF	force le b7 du tampon touche à 0

DA03-	20 02 D3	JSR D302	JSR EB78/ROM si touche frappée alors N = 1 et A = code ASCII correspondant, sinon N = 0
DA06-	10 FB	BPL DA03	reboucle tant qu'une touche n'a pas été pressée
DA08-	AD 08 02	LDA 0208	code de touche normale
DA0B-	AC 09 02	LDY 0209	code de touche CTRL, FUNCT, SHIFT G ou D
DA0E-	C0 A4	CPY #A4	la touche SHIFT Gauche a-t-elle été pressée?
DA10-	F0 04	BEQ DA16	si oui, continue en DA16
DA12-	C0 A7	CPY #A7	la touche SHIFT Droite a-t-elle été pressée?
DA14-	D0 02	BNE DA18	sinon, continue en DA18
DA16-	09 40	ORA #40	si SHIFT, force à 1 b6 du code touche normale
DA18-	29 7F	AND #7F	SHIFT ou pas, force à 0 b6 code touche normale
DA1A-	A8	TAY	le résultat est copié dans Y (index d'écriture)
DA1B-	8A	TXA	tandis que X (code de fonction) est mis en
DA1C-	99 00 C8	STA C800,Y	place dans la table "KEYDEF" en C800
DA1F-	60	RTS	NB: les opérations ORA #40 et AND#7F transforment les codes de touche qui vont de #80 à BF en index d'écriture qui vont de #00 à 3F si FUNCT et de #40 à 7F si FUNCT+SHIFT.

EXÉCUTION DE LA COMMANDE SEDORIC KEYIF

Rappel de la syntaxe

KEYIF code_d'une_touche_clavier GOTO ... (ELSE ...) ou
KEYIF code_d'une_touche_clavier THEN ... (ELSE ...)

Lorsque la touche correspondant au code indiqué (voir manuel page 104) est pressée, le programme continue au n° de ligne indiqué ou exécute la commande spécifiée (fonctionnement dentique à IF THEN ELSE). Cette commande marche même si le clavier est inhibé ou si plusieurs touches sont pressées à la fois.

Non documenté

Toujours le même problème avec l'utilisation possible des minuscules pour entrer les commandes SEDORIC: ça ne marche pas à tous les coups (bogue). Ici, la commande KEYIF se contente de détecter si la touche spécifiée a été pressée et si oui, passe la main à la commande BASIC "IF..." Si dans votre ardeur vous tapez "keyif ... goto ... else" ou "keyif ... then ... else", le "keyif" sera accepté, mais pas "goto", ni "then", ni "else".

Analyse de syntaxe et exécution

DA20-	20 7F D2	JSR D27F	CF17/ROM et CF09/ROM et D8CB/ROM évalue un nombre entier à TXTPTR et le retourne dans X (voir les codes de touche en ANNEXE)
DA23-	08	PHP	sauvegarde les indicateurs 6502
DA24-	78	SEI	interdit les interruptions
DA25-	8A	TXA	empile le code de touche à détecter
DA26-	48	PHA	un code de touche est de la forme 10CCCLLL ou
DA27-	4A	LSR	CCC est le n° de colonne du clavier de 0 à 7 et

DA28-	4A	LSR	LLL est le n° de ligne du clavier de 0 à 7
DA29-	4A	LSR	après les 3 LSR et le AND #07 on a 0000 0CCC
DA2A-	29 07	AND #07	c'est à dire X = n° de la colonne de
DA2C-	AA	TAX	la touche requise (de #00 à 07)
DA2D-	18	CLC	C = 0
DA2E-	A9 FF	LDA #FF	A = 1111 1111
DA30-	2A	ROL	C <- b7 ... b0 <- C Effectue un nombre de
DA31-	CA	DEX	rebouclages égal à X. Le 0 se déplace donc de
DA32-	10 FC	BPL DA30	bit en bit dans A jusqu'au bx. Le seul bit de X
DA34-	AA	TAX	à 0 marque donc la colonne du clavier à activer
DA35-	A9 0E	LDA #0E	A = 14, n° registre correspondant au port A du 6522
DA37-	20 22 D3	JSR D322	JSR F590/ROM place X dans le registre A du PSG 8912 (Programmable Sound Generator) (n°1 à 13 pour son, n°14 pour clavier)
DA3A-	68	PLA	récupère le code de touche à détecter
DA3B-	29 07	AND #07	force à 0 les bits b7 à b3, puis force à 1 les
DA3D-	09 B8	ORA #B8	b7, b5 et b4, ce qui donne un code de ligne de la forme 1011 0LLL où LLL est le n° de ligne du clavier de 0 à 7
DA3F-	20 3A D8	JSR D83A	teste si touche définie par X = n° de colonne et par A = N° de ligne a été pressée, si oui A = #08 et sinon A = #00
DA42-	85 D0	STA D0	sauve la réponse dans D0 pour la gestion du ELSE
DA44-	28	PLP	récupère les indicateurs
DA45-	20 EB D1	JSR D1EB	JSR CA73/ROM exécute la commande "IF"
DA48-	4E FC 04	LSR 04FC	force à 0 le b7 de 04FC (flag "IF")
DA4B-	60	RTS	

AUTRE SÉRIE DE ROUTINES GÉNÉRALES SEDORIC

XPMAP prend le premier secteur de bitmap dans BUF2, vérifie le format, met à zéro le b7 de 2F (flag"première bitmap chargée").

Cette routine a été modifiée par Ray afin de pouvoir utiliser une double bitmap. En DA4C, le code d'origine (LDA #14 et LDY #02) a été remplacé (les 4 octets modifiés sont indiqués en gras). Le principe usuel a été utilisé: pour palier au manque local de place, une partie du code est remplacée par un saut à une routine insérée ailleurs et dans laquelle on peut à "loisir" reprendre le code supprimé et ajouter les opérations supplémentaires. N'hésitez pas à aller voir le "greffon" en E62E.

DA4C	**20 2E E6**	JSR E62E	nouveau sous-programme permettant d'insérer la modification. Ce petit détour charge dans AY les coordonnées du premier secteur de la double bitmap et met à zéro le b7 de 2F (flag"première bitmap chargée").
DA4F	**EA**	NOP	

Charge le secteur de bitmap de coordonnées AY dans BUF2 et vérifie le format

DA50-	20 60 DA	JSR DA60	XPBUF2 charge dans BUF2 le secteur Y de la piste A
DA53-	AE 00 C2	LDX C200	teste le premier octet (n°0) (#FF en principe)
DA56-	E8	INX	qui doit passer à 0 (donc Z = 1)
DA57-	F0 74	BEQ DACD	si oui, OK branche sur un simple RTS
DA59-	A2 0A	LDX #0A	sinon, prépare une "UNKNOW'N_FORMAT_ERROR"
DA5B-	D0 22	BNE DA7F	et la traite en D67E (il y a un renvoi)

XPBUF1 charge dans BUF1 le secteur Y de la piste A

DA5D-	A2 C1	LDX #C1	X = HH de BUF1
DA5F-	2C A2 C2	BIT C2A2	et continue en DA65

XPBUF2 charge dans BUF2 le secteur Y de la piste A

DA60-	A2 C2	LDX #C2	X = HH de BUF2
DA62-	2C A2 C3	BIT C3A2	et continue en DA65

XPBUF3 charge dans BUF3 le secteur Y de la piste A

DA63-	A2 C3	LDX #C3	X = HH de BUF3

Charge à la page X le secteur Y de la piste A

DA65-	8E 04 C0	STX C004	HH de l'adresse de chargement d'un secteur
DA68-	A2 00	LDX #00	prépare LL de RWBUF
DA6A-	8E 03 C0	STX C003	LL de l'adresse de chargement d'un secteur

XPAY charge dans RWBUF le secteur Y de la piste A

DA6D-	8D 01 C0	STA C001	piste A -> PISTE

DA70- 8C 02 C0 STY C002 secteur Y -> SECTEUR

XPRSEC lit un secteur selon DRIVE, PISTE, SECTEUR et RWBUF

DA73-	A2 88	LDX #88	paramètre lecture pour XRWTS (gestion des drives)
DA75-	20 CD CF	JSR CFCD	XRWTS X = commande revient avec Z = 1 si pas d'erreur ou avec Z = 0 si erreur (V = 1 si la disquette est protégée en écriture)
DA78-	F0 53	BEQ DACD	si pas d'erreur branche en DACD (simple RTS)
DA7A-	A2 03	LDX #03	prépare "DISK_I/O_ERROR"
DA7C-	50 01	BVC DA7F	saute si ce n'était pas une disquette protégée
DA7E-	E8	INX	sinon prépare "WRITE_PROTECTED_ERROR"
DA7F-	4C 7E D6	JMP D67E	et traite l'erreur n° 4 ou 5

XSCAT sauve BUF3 (secteur de DIR) selon POSNMP et POSNMS

DA82-	AD 25 C0	LDA C025	POSNMP piste du nom cherché dans catalogue
DA85-	AC 26 C0	LDY C026	POSNMS secteur du nom cherché dans catalogue
DA88-	D0 0A	BNE DA94	branchement forcé: n° de secteur jamais nul

XSMAP (sauve le secteur de bitmap sur la disquette), l'entrée a été déportée en DC80

Tout au début de cette routine, en DA8A, le code d'origine (LDA #14 et LDY #02) a été modifié par Ray afin de supporter la double bitmap (les 4 octets différents sont indiqués en gras ci-dessous). Le principe usuel a été utilisé: pour palier au manque local de place, une partie du code est remplacée par un saut à une routine insérée ailleurs et dans laquelle on peut à "loisir" reprendre le code supprimé et ajouter les opérations supplémentaires. N'hésitez pas à aller voir le "greffon" en DC80.

DA8A **4C 80 DC** JMP DC80 nouveau sous-programme permettant d'insérer la modification

DA8D- **EA** NOP

Reprend le cours normal de XSMAP: sauve BUF2 sur la disquette dans le Y ième secteur de la piste 20

DA8E-	A2 C2	LDX #C2	HH de BUF2 (secteur de bitmap)
DA90-	2C A2 C1	BIT C1A2	continue en #DA96

XSBUF1 sauve BUF1 au secteur Y de la piste A

DA91-	A2 C1	LDX #C1	HH de BUF1
DA93-	2C A2 C3	BIT C3A2	continue en #DA96

XSBUF3 sauve BUF3 au secteur Y de la piste A

DA94-	A2 C3	LDX #C3	HH de BUF3

Sauve la page X dans le secteur Y de la piste A

DA96-	8E 04 C0	STX C004	place HH de RWBUF

```
DA99-  A2 00      LDX #00      prépare LL de RWBUF
DA9B-  8E 03 C0   STX C003     place LL de RWBUF
```

XSAY sauve la page indiquée par RWBUF dans le secteur Y de la piste A

```
DA9E-  8D 01 C0   STA C001     piste A -> PISTE
DAA1-  8C 02 C0   STY C002     secteur Y -> SECTEUR
```

XSVSEC écrit un secteur selon DRIVE, PISTE, SECTEUR et RWBUF

```
DAA4-  A2 A8      LDX #A8      paramètre écriture pour XRWTS (gestion des drives)
DAA6-  D0 CD      BNE DA75     suite forcée XRWTS (X = commande, erreur: Z = 0, disquette protégée:
                               V = 1)
```

Sauve BUF1 selon DRIVE, PISTE et SECTEUR

```
DAA8-  A9 00      LDA #00
DAAA-  A0 C1      LDY #C1
DAAC-  8D 03 C0   STA C003     RWBUF pointe sur BUF1
DAAF-  8C 04 C0   STY C004
DAB2-  D0 F0      BNE DAA4     branchement forcé vers XSVSEC (ci-dessus)
```

Affiche le nom de fichier présent à POSNMX dans BUF3

```
DAB4-  AE 27 C0   LDX C027     X = POSNMX position du nom dans le secteur de catalogue
DAB7-  A0 08      LDY #08      index pour afficher 9 caractères
DAB9-  20 C3 DA   JSR DAC3     lit et affiche Y+1 caractères (de Y à 0 inclus)
DABC-  A9 2E      LDA #2E      code ASCII de "."
DABE-  20 2A D6   JSR D62A     XAFCAR affiche le caractère ASCII contenu dans A
DAC1-  A0 02      LDY #02      index pour afficher 3 caractères
```

Lit Y caractères à POSNMX dans BUF3 et les affiche

```
DAC3-  BD 00 C3   LDA C300,X   lit caractère et l'affiche
DAC6-  20 2A D6   JSR D62A     XAFCAR affiche le caractère ASCII contenu dans A
DAC9-  E8         INX          indexe caractère suivant
DACA-  88         DEY          décrémente le nombre de caractères à lire
DACB-  10 F6      BPL DAC3     reboucle s'il en reste (si pas négatif)
DACD-  60         RTS          et retourne
```

XVBUF1 rempli BUF1 de zéros

```
DACE-  A9 C1      LDA #C1      A = HH de BUF1
DAD0-  2C A9 C2   BIT C2A9     continue en DAD6
```

XVBUF2 rempli BUF2 de zéros

```
DAD1-  A9 C2      LDA #C2      A = HH de BUF2
```

```
DAD3- 2C A9 C3  BIT C3A9      continue en DAD6
```

XVBUF3 rempli BUF3 de zéros

```
DAD4- A9 C3     LDA #C3       A = HH de BUF3
```

Rempli de zéros une page mémoire à partir de HH = A et LL=#00

```
DAD6- 85 0F     STA 0F
DAD8- A9 00     LDA #00       adresse du buffer à vider -> 0E/0F
DADA- 85 0E     STA 0E
DADC- A0 00     LDY #00       remet à zéro index Y
DADE- 98        TYA           remet à zéro X
DADF- 91 0E     STA (0E),Y    copie X à l'adresse pointée
DAE1- C8        INY           vise adresse suivante
DAE2- D0 FB     BNE DADF      et reboucle tant que pas nul
DAE4- 60        RTS           (vide de LL=#00 à LL=#FF soit 256 octets)
```

Charge dans BUF3 le secteur pointé par POSNMP et POSNMS puis exécute XBUCA

```
DAE5- AD 25 C0  LDA C025      piste (POSNMP)
DAE8- AC 26 C0  LDY C026      secteur (POSNMS)
DAEB- 20 63 DA  JSR DA63      XPBUF3 charge dans BUF3 le secteur Y de la piste A
```

XBUCA transfère le nom de fichier contenu dans BUFNOM dans le secteur de catalogue contenu dans BUF3, à la position POSNMX

(pour mise à jour de "l'entrée" de catalogue sur la disquette)

```
DAEE- AE 27 C0  LDX C027      position du nom de fichier dans le secteur de catalogue
DAF1- A0 F0     LDY #F0       artifice génial! lit dans BUFNOM à partir de
DAF3- B9 39 BF  LDA BF39,Y    BF39 + F0 = C029 = premier caractère du nom dans BUFNOM
DAF6- 9D 00 C3  STA C300,X    et écrit à partir de POSNMX dans BUF3
DAF9- E8        INX           index écriture caractère suivant
DAFA- C8        INY           index lecture caractère suivant
DAFB- D0 F6     BNE DAF3      jusqu'à ce que Y atteigne #00 soit 16 octets
DAFD- 60        RTS           de #F0 à #(1)00
```

Charge dans BUF3 le secteur pointé par POSNMP et POSNMS puis exécute XCABU

```
DAFE- AD 25 C0  LDA C025      piste (POSNMP)
DB01- AC 26 C0  LDY C026      secteur (POSNMS)
DB04- 20 63 DA  JSR DA63      XPBUF3 charge dans BUF3 le secteur Y de la piste A
```

XCABU met dans BUFNOM le nom présent dans BUF3 à la position POSNMX

```
DB07- AE 27 C0  LDX C027      position du nom de fichier dans le secteur de catalogue
DB0A- A0 F0     LDY #F0       artifice génial! (si, si, si!) (voir ci-dessus en DAF3)
```

DB0C-	BD 00 C3	LDA C300,X	lit à partir de POSNMX dans BUF3 puis
DB0F-	99 39 BF	STA BF39,Y	écrit dans BUFNOM à partir de C029
DB12-	E8	INX	index lecture caractère suivant
DB13-	C8	INY	index écriture caractère suivant
DB14-	D0 F6	BNE DB0C	jusqu'à ce que Y atteigne #00 soit 16 octets
DB16-	60	RTS	de #F0 à #(1)00

Comparaison du nom cherché (BUFNOM) et du nom pointé par X dans le catalogue (BUF3)

DB17-	A0 F4	LDY #F4	lit dans BUFNOM à partir de
DB19-	B9 35 BF	LDA BF35,Y	BF35 + F4 = C029 = premier caractère du nom dans BUFNOM
DB1C-	C9 3F	CMP #3F	est-ce un "?"? (joker)
DB1E-	F0 05	BEQ DB25	si oui, branche en DB25 (saute la comparaison)
DB20-	DD 00 C3	CMP C300,X	sinon, compare avec le caractère correspondant du catalogue
DB23-	D0 1C	BNE DB41	si différent, branche en DB41 (ajuste POSNMX sur “l’entrée” suivante et reprend la comparaison en DB17)
DB25-	E8	INX	si identique, incrémente
DB26-	C8	INY	les index X et Y (vise caractère suivant)
DB27-	D0 F0	BNE DB19	reboucle jusqu'à Y = #00 (soit 12 caractères)
DB29-	AE 27 C0	LDX C027	nom identique: remet X = POSNMX
DB2C-	60	RTS	(X entrée nom_à_comparer = X sortie nom_trouvé)

Vérifie que la disquette en place est bien une disquette SEDORIC, cherche le fichier dans le catalogue, revient avec le bon secteur de catalogue en place (de coordonnées POSNMP et POSNMS) et avec X = POSNMX, pointant sur "l'entrée" cherchée ou avec Z = 1 si le fichier n'a pas été trouvé

DB2D-	20 4C DA	JSR DA4C	XPMAP prend le premier secteur de bitmap dans BUF2, vérifie le format, met à zéro le b7 de 2F (flag”première bitmap chargée”).

XTVNM cherche sur le lecteur courant le fichier dont le nom est indiqué dans BUFNOM.

A la sortie, POSNMX, POSNMP, et POSNMS contiennent la position du nom dans le catalogue (BUF3), et Z = 1 si le fichier n'est pas trouvé

DB30-	A9 14	LDA #14	piste 20
DB32-	A0 04	LDY #04	secteur 4 = premier secteur de catalogue
DB34-	8D 25 C0	STA C025	POSNMP piste du nom cherché dans le catalogue
DB37-	8C 26 C0	STY C026	POSNMS secteur du nom cherché dans le catalogue
DB3A-	20 63 DA	JSR DA63	XPBUF3 charge dans BUF3 le secteur Y de la piste A
DB3D-	A2 10	LDX #10	pointe sur la première entrée (premier nom de fichier)
DB3F-	D0 07	BNE DB48	branchement forcé en DB48: mise à jour de POSNMX

Ajuste POSNMX sur “l’entrée” suivante du catalogue et reprend la recherche dans le catalogue du fichier indiqué dans BUFNOM (Z = 1 si fini)

DB41-	AD 27 C0	LDA C027	A = POSNMX (ancienne valeur)
DB44-	18	CLC	prépare addition
DB45-	69 10	ADC #10	ajoute 16 (ligne suivante du catalogue)

DB47-	AA	TAX	mise à jour de X (pour la comparaison des noms)
DB48-	8E 27 C0	STX C027	mise à jour de POSNMX
DB4B-	EC 02 C3	CPX C302	l'octet n°2 contient POSNMX maximale = (n° de l'entrée + 1) x #10 où le #10 représente 16 caractères par entrée et le 1 représente les 16 premiers octets réservés aux renseignements concernant le directory (dont l'adresse du directory suivant)
DB4E-	D0 C7	BNE DB17	si la fin n'est pas atteinte on peut comparer
DB50-	AD 00 C3	LDA C300	sinon (#00), on prend l'adresse du directory suivant
DB53-	AC 01 C3	LDY C301	(secteur de catalogue suivant = octets n°0 et 1)
DB56-	D0 DC	BNE DB34	reboucle si pas nul (n° secteur jamais nul)
DB58-	60	RTS	si nul, pas trouvé (c'était le dernier secteur de catalogue) dans ce cas retourne au programme appelant avec Z = 1

XTRVCA cherche une place libre dans le catalogue.

A la sortie, POSNMX, POSNMP et POSNMS indiquent la position de la place réservée.

DB59-	20 A5 DB	JSR DBA5	cherche POSNMX première place libre dans directory
DB5C-	D0 34	BNE DB92	teste Z branche si position libre trouvée
DB5E-	AD 08 C2	LDA C208	si aucune place de libre, A = nombre de secteurs de catalogue
DB61-	C9 05	CMP #05	y en a t-il déjà 5 ou plus?
DB63-	B0 0A	BCS DB6F	si oui, branche en DB6F
DB65-	AD 02 C0	LDA C002	sinon, lit SECTEUR (dernier secteur de catalogue)
DB68-	69 03	ADC #03	et ajoute 3 (4, 7, 10, 13 et 16 sont réservés)
DB6A-	A8	TAY	comme d'habitude Y = n° de secteur
DB6B-	A9 14	LDA #14	et A = n° de piste
DB6D-	D0 03	BNE DB72	branchement forcé: saute ligne suivante
DB6F-	20 6C DC	JSR DC6C	XLIBSE cherche un secteur libre dans la bitmap dans BUF2, retourne avec A = n° de piste et Y = n° de secteur (sinon "DISK_FULL_ERROR")
DB72-	8D 00 C3	STA C300	les octets n°#00 et #01 du dernier secteur de catalogue
DB75-	8C 01 C3	STY C301	indiquent maintenant les coordonnées du catalogue suivant
DB78-	EE 08 C2	INC C208	le nombre de secteurs de catalogue est mis à jour
DB7B-	20 8A DA	JSR DA8A	l'entrée de cette routine XSMAP sauve BUF2 (bitmap) sur la disquette a été déportée en DC80 pour adaptation à la double bitmap. Un JSR DC80 plus direct aurait fait gagner un peu de temps.
DB7E-	20 82 DA	JSR DA82	XSCAT sauve BUF3 (secteur de catalogue) idem
DB81-	AD 00 C3	LDA C300	
DB84-	AC 01 C3	LDY C301	copie les coordonnées du catalogue suivant
DB87-	8D 25 C0	STA C025	dans POSNMP et POSNMS
DB8A-	8C 26 C0	STY C026	
DB8D-	20 D4 DA	JSR DAD4	XVBUF3 rempli de 0 le BUFFER3 (catalogue suivant)
DB90-	A2 10	LDX #10	indexe première entrée dans ce secteur de catalogue
DB92-	8A	TXA	copie la position de l'entrée libre dans A
DB93-	8E 27 C0	STX C027	et dans POSNMX (première place libre dans directory)
DB96-	18	CLC	prépare addition
DB97-	69 10	ADC #10	ajoute 16
DB99-	8D 02 C3	STA C302	mise à jour place libre suivante dans directory
DB9C-	EE 04 C2	INC C204	

```
DB9F- D0 1E     BNE DBBF    mise à jour du nombre de fichiers (bitmap)
DBA1- EE 05 C2  INC C205
DBA4- 60        RTS
```

Cherche POSNMX de la première place libre dans directory

Retourne avec Z = 0 si une place libre est trouvée et X = sa position dans le secteur Y de la piste A ou avec Z = 1 si rien trouvé.

```
DBA5- A9 14     LDA #14     piste 20
DBA7- A0 04     LDY #04     secteur 4 (premier secteur de catalogue)
DBA9- 8D 25 C0  STA C025    POSNMP piste du nom cherché dans le catalogue
DBAC- 8C 26 C0  STY C026    POSNMS secteur du nom cherché dans le catalogue
DBAF- 20 63 DA  JSR DA63    XPBUF3 charge dans BUF3 le secteur Y de la piste A
DBB2- AE 02 C3  LDX C302    octet n°3 contient POSNMX maximale c'est la position de la première
                            entrée libre du catalogue (#00 si le catalogue est plein)
DBB5- D0 08     BNE DBBF    position trouvée on retourne avec X = POSNMX
DBB7- AD 00 C3  LDA C300    si nul, il faut charger le secteur de catalogue suivant
DBBA- AC 01 C3  LDY C301    dont on prend les coordonnées piste = A, secteur = Y
DBBD- D0 EA     BNE DBA9    n° secteur étant jamais nul, reboucle pour suivant
DBBF- 60        RTS         sauf si c'était le dernier (Z = 1 si rien trouvé)
```

XWDESC écrit le ou les descripteurs du fichier à sauver

Reviens avec le nombre de secteurs à sauver dans NSSAV (C05A/5B), les coordonnées du premier secteur descripteur dans PSDESC (C05C/5D), le nombre de descripteurs utilisés dans NSDESC (C05E) et premier descripteur en place.

```
DBC0- 8D 58 C0  STA C058    AY -> NSRSAV
DBC3- 8C 59 C0  STY C059    (nombre de secteurs restant à sauver)
DBC6- 8D 5A C0  STA C05A    AY -> NSSAV
DBC9- 8C 5B C0  STY C05B    (nombre de secteurs à sauver)
```

Ecriture du descripteur n°1

```
DBCC- 20 CE DA  JSR DACE    XVBUF1 rempli BUF1 de 0 (futur descripteur)
DBCF- A2 01     LDX #01
DBD1- 8E 5E C0  STX C05E    1 -> NSDESC (nombre de descripteurs utilisés)
DBD4- 20 6C DC  JSR DC6C    XLIBSE cherche un secteur libre dans la bitmap dans BUF2, retourne avec
                            A = n° de piste et Y = n° de secteur (sinon "DISK_FULL_ERROR")
DBD7- 8D 5C C0  STA C05C    AY -> PSDESC (coordonnées premier secteur descripteur)
DBDA- 8C 5D C0  STY C05D    (n° piste -> C05C et n° secteur -> C05D)
DBDD- 8D 01 C0  STA C001    n° piste -> PISTE
DBE0- 8C 02 C0  STY C002    n° secteur -> SECTEUR
DBE3- A2 08     LDX #08
DBE5- BD 51 C0  LDA C051,X  copie 9 octets de C051 à C059 en C103 à C10B
DBE8- 9D 03 C1  STA C103,X  soit FTYPE, DESALO, FISALO, EXSALO et NSRSAV
DBEB- CA        DEX         (c'est à dire les 9 octets de STATUS)
```

```
DBEC-  10 F7      BPL DBE5
DBEE-  8E 02 C1   STX C102     soit #FF -> C102 marque du premier secteur descripteur
DBF1-  A2 0C      LDX #0C      X = #0C pointe dans le secteur descripteur au début de la liste coordonnées
                               piste/secteur des secteurs constituants le fichier
```

Boucle d'écriture, dans le descripteur, de la liste des coordonnées des NSRSAV secteurs constituants le fichier

```
DBF3-  8E 5F C0   STX C05F     X -> C05F (PTDESC = pointeur descripteur)
DBF6-  AD 58 C0   LDA C058     teste NSRSAV
DBF9-  0D 59 C0   ORA C059     reste t-il des secteurs à sauver?
DBFC-  F0 58      BEQ DC56     sinon branche en DC56 (fini)
DBFE-  AD 58 C0   LDA C058
DC01-  D0 03      BNE DC06
DC03-  CE 59 C0   DEC C059     décrémente le nombre de secteurs à sauver
DC06-  CE 58 C0   DEC C058
DC09-  20 6C DC   JSR DC6C     XLIBSE cherche un secteur libre dans la bitmap dans BUF2, retourne avec
                               A = n° de piste et Y = n° de secteur (sinon "DISK_FULL_ERROR")
DC0C-  AE 5F C0   LDX C05F     X pointe dans la liste des coordonnées du descripteur
DC0F-  9D 00 C1   STA C100,X   écrit dans la liste des coordonnées du
DC12-  E8         INX          descripteur une paire piste/secteur
DC13-  98         TYA          pointant sur chacun des secteurs
DC14-  9D 00 C1   STA C100,X   constituants le fichier
DC17-  E8         INX          vise octet suivant du descripteur
DC18-  D0 D9      BNE DBF3     reboucle si fin du descripteur pas atteinte
DC1A-  AD 58 C0   LDA C058     si fin du descripteur atteinte,
DC1D-  0D 59 C0   ORA C059     teste s'il reste des secteurs à sauver
DC20-  F0 34      BEQ DC56     s'il ne reste rien branche en DC56 (fini)
DC22-  AC 01 C1   LDY C101     s'il en reste, teste le n° de secteur du descripteur suivant
DC25-  D0 1C      BNE DC43     si déjà validé, branche en DC43
DC27-  20 6C DC   JSR DC6C     si pas de suivant, XLIBSE cherche un secteur libre dans la bitmap dans
                               BUF2, retourne avec A = n° de piste et Y = n° de secteur (sinon
                               "DISK_FULL_ERROR")
DC2A-  8D 00 C1   STA C100     écrit le n° de piste en C100
DC2D-  48         PHA          et l'empile
DC2E-  8C 01 C1   STY C101     écrit le n° de secteur en C101
DC31-  98         TYA          et l'empile
DC32-  48         PHA
DC33-  20 A8 DA   JSR DAA8     sauve BUF1 selon DRIVE, PISTE et SECTEUR
DC36-  68         PLA
DC37-  8D 02 C0   STA C002     récupère dans PISTE et SECTEUR les
DC3A-  68         PLA          coordonnées du descripteur suivant
DC3B-  8D 01 C0   STA C001
DC3E-  EE 5E C0   INC C05E     augmente NSDESC (nombre de descripteurs utilisés)
DC41-  D0 0C      BNE DC4F     branchement forcé en DC4F
```

Sauve le descripteur courant et charge le suivant

DC43-	20 A8 DA	JSR DAA8	sauve BUF1 selon DRIVE, PISTE et SECTEUR
DC46-	AD 00 C1	LDA C100	
DC49-	AC 01 C1	LDY C101	AY coordonnées du descripteur suivant
DC4C-	20 5D DA	JSR DA5D	XPBUF1 charge dans BUF1 le descripteur suivant

Ecriture d'un descripteur secondaire

DC4F-	20 CE DA	JSR DACE	XVBUF1 rempli BUF1 de 0
DC52-	A2 02	LDX #02	2 -> PTDESC = pointeur descripteur: pour les descripteurs "secondaires", la liste des coordonnées commence à l'octet n°2
DC54-	D0 9D	BNE DBF3	branchement forcé vers la boucle d'écriture de la liste des coordonnées des secteurs constituants le fichier

Sauve BUF1 et charge premier secteur descripteur

DC56-	A9 00	LDA #00	
DC58-	8D 00 C1	STA C100	mise à zéro des coordonnées du descripteur suivant
DC5B-	8D 01 C1	STA C101	
DC5E-	20 A8 DA	JSR DAA8	sauve BUF1 selon DRIVE, PISTE et SECTEUR
DC61-	AD 5C C0	LDA C05C	
DC64-	AC 5D C0	LDY C05D	coordonnées du prochain secteur libre
DC67-	4C 5D DA	JMP DA5D	XPBUF1 charge dans BUF1 le secteur Y de la piste A. Il aurait fallu tester NSDESC car dans la plupart des cas, il n'y a qu'un seul descripteur et on pourrait se passer de ce re-chargement!

XLIBSE cherche un secteur libre sur la bitmap dans BUF2

Entrée en DC6C. Retourne avec A = n° de la piste et Y = n° du secteur libre trouvé. Sinon "DISK_FULL_ERROR"

Curieusement cette entrée n'est pas utilisée !

DC6A-	18	CLC	entrée avec C = 0
DC6B-	24 38	BIT 38	continue en DC6D

Entrée proprement dite de XLIBSE

DC6C-	38	SEC	entrée avec C = 1

Vérifie qu'il y a encore de la place sur la disquette

DC6D-	AD 02 C2	LDA C202	
DC70-	AA	TAX	X = LL du nombre de secteurs libres
DC71-	0D 03 C2	ORA C203	teste si la disquette est pleine (c'est à dire si le nombre de secteurs libres C202/C203 est nul)
DC74-	D0 07	BNE DC7D	si pas pleine, continue en DC7D (routine "Cherche un secteur libre")
DC76-	90 5C	BCC DCD4	si pleine et vient de l'entrée DC6A, INY et RTS
DC78-	A2 07	LDX #07	si pleine et vient de l'entrée DC6C, prépare une

DC7A- 4C 7E D6 JMP D67E "DISK_FULL_ERROR"

Ancienne routine “Cherche un secteur libre”, déportée par Ray en E67F, pour tenir compte de la double bitmap

Chaque octet de la liste des secteurs de la disquette (bitmap proprement dite qui commence à l'octet n° #10 du secteur de bitmap) est formé des bits b7 à b0. Chacun de ces bits représente un secteur de la disquette. Ce secteur est libre si le bit est à 1 ou occupé si le bit est à 0.

Ainsi, le b0 du premier octet représente l'état du secteur n°#01 de la piste n°#00. L'état du N ième secteur est représenté par le bit b(r-1) de l'octet n° #10 + N/8, r étant le reste de la division N/8. Par exemple, sur une disquette formatée en 17 secteurs par piste, pour le deuxième secteur de catalogue (secteur 7 de la piste 20) N = (17 x 20) + 7 = 347 (en effet, on compte 17 secteurs pour les pistes n°0 à 19 et le secteur visé est le septième de la vingtième piste). Or 347 / 8 = 43 (#2B) et on a r = 3. Le deuxième secteur de catalogue est donc représenté par le b2 (troisième bit de poids faible) de l'octet n°#10 + #2B = #3B.

Le sous-programme suivant suit la même logique mise à l'envers: il cherche dans la bitmap un octet libre, inverse le bit correspondant et calcule à quelle piste A et à quel secteur Y il correspond.

De DC7D à DC8E, le code d'origine de la routine "Cherche un secteur libre" a été modifié par Ray afin de supporter la double bitmap. Le principe usuel a été utilisé: pour palier au manque local de place, une partie du code est remplacée par un saut à une nouvelle routine "Cherche un secteur libre" située en E67F. Cette nouvelle routine ne reprend le cours normal de l'ancienne qu'en DC90. Ray a donc utilisé le surplus de place ainsi libérée, de DC80 à DC8F pour créer un nouveau sous-programme. Ce sous-programme "Sauve la bitmap sur la disquette" est appelé par XSMAP en DA8A à l'aide d'un JMP DC80 et y retourne à la fin à l'aide d'un JMP DA8E. Au total, les 19 octets modifiés sont indiqués en gras.

DC7D **4C 7F E6** JMP E67F routine "Cherche un secteur libre" modifiée

XSMAP sauve la bitmap sur la disquette

DC80	**24 2F**	BIT 2F	teste le b7 de 2F (flag première ou deuxième page active)
DC82	**10 05**	BPL DC89	continue en DC89 si bitmap normale (première page active): écrit BUF2 dans le premier secteur de bitmap sur la disquette
DC84	**08**	PHP	sinon (deuxième page active), sauve les registres
DC85	**20 3A E6**	JSR E63A	empile les octets C202 à C208, sauve BUF2 sur la disquette dans le troisième secteur de la piste 20, charge le deuxième secteur de la piste 20, restaure les octets C202 à C208, force C = 1
DC88	**28**	PLP	récupère les registres sauvés ci-dessus

Ecrit BUF2 dans le premier secteur de bitmap sur la disquette

DC89 **A0 02** LDY #02 sauve la première page de bitmap: deuxième secteur

Ecrit BUF2 dans le second secteur de bitmap sur la disquette
(entrée secondaire avec Y = #03 pré-positionné)

DC8B **A9 14** LDA #14 de la piste 20

DC8D	**4C 8E DA**	JMP DA8E	reprend le cours normal de XSMAP: sauve BUF2 sur la disquette dans le Y ième secteur de la piste 20

Reprise du cours normal de l'ancienne routine "Cherche un secteur libre"

DC90-	A9 01	LDA #01	0000 0001 masque pour premier bit
DC92-	A0 00	LDY #00	n° du bit examiné (premier = b0)
DC94-	48	PHA	empile le masque
DC95-	3D 10 C2	AND C210,X	teste le secteur pointé par le masque et par X
DC98-	D0 05	BNE DC9F	branche si le bit est à 1 (secteur libre)
DC9A-	68	PLA	si occupé, récupère le masque
DC9B-	0A	ASL	pointe sur le secteur suivant (décalage à gauche)
DC9C-	C8	INY	incrémente le n° d'ordre du bit examiné
DC9D-	D0 F5	BNE DC94	rebouclage forcé (il existe au moins 1 bit à 1)

Inverse le bit correspondant et met à jour la bitmap

DC9F-	68	PLA	récupère le masque et en inverse tous les bits
DCA0-	49 FF	EOR #FF	(secteur libre est pointé par le 0 du masque)
DCA2-	3D 10 C2	AND C210,X	met à zéro le bit du secteur à réserver
DCA5-	9D 10 C2	STA C210,X	re-écrit le résultat dans la bitmap

Calcule le nombre de secteurs du début de la disquette jusqu'au secteur trouvé

Sachant que chaque octet gère l'état de 8 secteurs, il faut multiplier par 8 le nombre d'octets situés avant l'octet modifié dans la bitmap et ajouter le nombre de secteurs représentés dans l'octet modifié entre b0 et le bit modifié.

DCA8-	**A9** 00	LDA #00	
DCAA-	85 F3	STA F3	F3 = #00 (sera le HH du nombre de secteurs)
DCAC-	8A	TXA	nombre d'octets situés avant l'octet modifié
DCAD-	0A	ASL	pour multiplier A par 8,
DCAE-	26 F3	ROL F3	on pousse les 3 bits de poids fort de A,
DCB0-	0A	ASL	dans les 3 bits de poids faible de F3,
DCB1-	26 F3	ROL F3	qui à la fin contient le HH du résultat,
DCB3-	0A	ASL	le LL étant dans A
DCB4-	26 F3	ROL F3	F2/F3 contient le nombre N de secteurs
DCB6-	85 F2	STA F2	représentés par les X octets situés avant l'octet modifié dans la bitmap, octet qui lui indique l'état des 8 secteurs suivants (de b0 à b7) et dont le n° du bit modifié est dans Y
DCB8-	98	TYA	récupère n° du bit modifié (codé par b0 b1 b2)
DCB9-	05 F2	ORA F2	ajoute Y secteurs au LL du résultat précédent (dont b0 b1 b2 sont à 0 à la suite des 3 ASL) en effectuant simplement un OU logique. A/F3 contient maintenant le n° d'ordre (premier secteur = n°0).

Calcule le n° de piste A et le n° de secteur Y

DCBB-	A2 FF	LDX #FF	pour avoir X = #00 en début de boucle. Le n° de secteur Y (reste + 1) est

			calculé par soustractions successives: n° d'ordre du secteur libre - nombre de secteurs par piste. Le nombre de soustractions donne le n° de piste A.
DCBD-	38	SEC	prépare soustraction
DCBE-	E8	INX	nombre de cycles de soustraction
DCBF-	A8	TAY	LL du reste actuel
DCC0-	ED 07 C2	SBC C207	A = A - nombre de secteurs par piste
DCC3-	B0 F8	BCS DCBD	reboucle tant que pas dépassé
DCC5-	C6 F3	DEC F3	si dépassé, décrémente aussi HH
DCC7-	10 F4	BPL DCBD	reboucle tant que pas dépassé
DCC9-	8A	TXA	A = n° de piste (nombre de soustractions)
DCCA-	EC 06 C2	CPX C206	compare A avec le nombre de pistes par face
DCCD-	90 05	BCC DCD4	première face si A < nombre de pistes par face
DCCF-	ED 06 C2	SBC C206	si deuxième face A = A - nombre de pistes par face
DCD2-	09 80	ORA #80	et force à 1 le b7 du n° de piste
DCD4-	C8	INY	n° de secteur = reste + 1
DCD5-	60	RTS	

Sous-programme pour XDETSE (DD15) et XCREAY (DD2D)

Calcule à quel octet de la bitmap correspond le secteur YA à libérer

Retourne avec X pointant sur un octet de la bitmap proprement dite (qui commence au #10 ème octet du secteur de bitmap) et avec dans A un masque dont un seul bit est à 1 représentant le secteur YA à libérer.

Rappel: l'état du N ième secteur d'une disquette est représenté par le bit b(r-1) de l'octet n° N/8 de la bitmap, “r” étant le reste de la division N/8. Ainsi sur une disquette formatée en 17 secteurs par piste, pour le deuxième secteur de catalogue (secteur 7, piste 20), N = (17 x 20) + 7 = 347 (en effet, on compte 17 secteurs pour les pistes n°0 à 19 et le secteur visé est le septième de la vingtième piste). Or 347 / 8 = 43 (#2B) et on a r = 3. Le deuxième secteur de catalogue est donc représenté par le b2 (troisième bit de poids faible) de l'octet n° X = #2B de la bitmap soit l’octet n°#10 + #2B = #3B du premier secteur de bitmap. Afin de simplifier le calcul de r-1, on utilise un truc simple en décrémentant Y dès le départ.

DCD6-	88	DEY	nombre de secteurs précédant le secteur Y sur la piste A NB: Y passera à zéro s’il représente le premier secteur de la piste A
DCD7-	AA	TAX	n° de la piste où se trouve le secteur à libérer
DCD8-	10 06	BPL DCE0	suite en DCE0 si c'est une piste de la première face
DCDA-	29 7F	AND #7F	sinon, force à zéro le flag de deuxième face, c'est à dire A = n° de la piste dans la deuxième face (en partant de #00 au lieu de #80)
DCDC-	18	CLC	calcule nouveau n° de piste = nombre de pistes
DCDD-	6D 06 C2	ADC C206	de la première face + n° de la piste dans la deuxième face
DCE0-	AA	TAX	X = nouveau n° de piste (sans saut à #80)

Nombre de secteurs dans les pistes précédentes

Ce qui exclu pour l’instant la piste où se trouve le secteur à libérer

DCE1-	A9 00	LDA #00	force à zéro le LL d'un totalisateur pour calculer le nombre de secteurs qu'il y a dans les X pistes précédant la piste de n° X où se trouve le secteur à

			libérer
DCE3-	85 F3	STA F3	F3 = #00 = HH de ce totalisateur
DCE5-	E0 00	CPX #00	teste s'il y a un calcul à faire (si secteur Y de la piste #00, le résultat est #0000, valeur actuelle du totalisateur)
DCE7-	F0 0B	BEQ DCF4	si pas de calcul, continue en DCF4
DCE9-	18	CLC	prépare multiplication par additions successives
DCEA-	6D 07 C2	ADC C207	ajoute le nombre de secteurs par piste
DCED-	90 02	BCC DCF1	si résultat < #FF, saute l'instruction suivante
DCEF-	E6 F3	INC F3	sinon, incrémente le HH du totalisateur
DCF1-	CA	DEX	décrémente le "nouveau n° de piste" et
DCF2-	D0 F5	BNE DCE9	reboucle en DCE9 jusqu'à ce qu'il arrive à #00
DCF4-	85 F2	STA F2	F2 = LL du résultat

Nombre de secteurs précédant le secteur à libérer

Ajout des secteurs précédant le secteur à libérer dans la piste de celui-ci.

DCF6-	18	CLC	prépare une addition pour calculer le nombre total de secteurs qu'il y a avant le secteur à libérer = résultat précédant + nombre de secteurs qui précédent le secteur à libérer dans la piste où il est situé
DCF7-	98	TYA	nombre de secteurs qui précédent le secteur à libérer dans la piste où il est situé
DCF8-	65 F2	ADC F2	plus LL du résultat précédant
DCFA-	90 02	BCC DCFE	si résultat < #FF, saute l'instruction suivante
DCFC-	E6 F3	INC F3	sinon, incrémente le HH du totalisateur
DCFE-	48	PHA	empile le LL du totalisateur. Le résultat actuel (nombre total de secteurs qui précédent le secteur à libérer) est donc sur la pile (LL) et dans F3 (HH)

Remarque

Pour diviser par 8 un nombre entier codé sur 2 octets, il faut effectuer 3 décalages à droite sur ces deux octets: les 3 bits faibles de HH passent dans les 3 bits forts de LL et les 3 bits faibles de LL, qui sont éjectés, constituent le reste.

Calcul du reste de la division

A contient toujours le LL du nombre de secteurs précédant le secteur à libérer: on se contente d'en récupérer les 3 bits de poids faible.

DCFF-	29 07	AND #07	force à zéro tous les bits de A sauf b0, b1 et b2
DD01-	A8	TAY	Y contient le reste de la division (simple!)

Calcule à quel octet et à quel bit de la bitmap correspond le secteur YA à libérer

En DD0B, la fin de cette routine a été détournée vers un sous-programme complémentaire de Ray qui tient compte du fait que le nombre de secteurs présents sur la disquette avant le secteur à libérer peut être plus grand que prévu initialement et que la modification peut affecter le deuxième secteur de bitmap (voir plus loin ce sous-programme). Pour palier au manque local de place, le principe usuel a donc encore été utilisé:

une partie du code est remplacée par un saut à une routine insérée ailleurs et dans laquelle on peut à "loisir" reprendre le code supprimé et ajouter les opérations supplémentaires. Pour insérer son JMP E6C4, Ray a été obligé d'écraser les 2 derniers octets (ROR et TAX) de la routine "Calcule à quel octet et à quel bit de la bitmap correspond le secteur YA à libérer" et le premier octet (qui était SEC) de la routine suivante. Ce SEC manquant sera inséré ailleurs (en E6D8) et les appels à DD0D seront remplacé par des appels à DD0E. Les 3 octets modifiés sont indiqués en gras.

DD02-	68	PLA	récupère le LL du totalisateur dans A
DD03-	46 F3	LSR F3	
DD05-	6A	ROR	
DD06-	46 F3	LSR F3	b0 b1 b2 du HH sont récupérés
DD08-	6A	ROR	dans b5 b6 b7 du LL
DD09-	46 F3	LSR F3	
DD0B	**4C C4 E6**	JMP E6C4	détournement vers sous-programme complémentaire

Elabore un masque dont un bit représente le secteur à libérer

A l'origine cette routine commençait en DD0D par un SEC qui a été écrasé par le JMP E6C4 ci-dessus. Pour réparer ceci, Ray a inséré ailleurs (en E6D8) le SEC manquant et les appels à l'ancienne routine en DD0D ont été remplacé par des appels en DD0E.

DD0E-	A9 00	LDA #00	le masque est à zéro au départ
DD10-	2A	ROL	effectue un nombre de rotation à gauche égal à
DD11-	88	DEY	la retenue de la division, ce qui amène C à la
DD12-	10 FC	BPL DD10	position qui représente le secteur à libérer
DD14-	60	RTS	

XDETSE libère le secteur Y de la piste A sur le secteur de bitmap courant

Retourne avec C = 1 si ce secteur était déjà libre sinon avec C = 0.

DD15-	20 D6 DC	JSR DCD6	calcule à quel octet X de la bitmap correspond ce secteur. Au retour: un bit de A est à 1 et représente le secteur à libérer
DD18-	1D 10 C2	ORA C210,X	prend dans A l'octet situé à la X ème position après le début de la bitmap et force à 1 le bit correspondant au masque
DD1B-	DD 10 C2	CMP C210,X	teste si l'octet est inchangé
DD1E-	F0 0C	BEQ DD2C	si oui, simple RTS en DD2C avec C = 1
DD20-	9D 10 C2	STA C210,X	réécrit cet octet s'il a été modifié
DD23-	EE 02 C2	INC C202	et incrémente le nombre d'octets libres
DD26-	D0 04	BNE DD2C	
DD28-	EE 03 C2	INC C203	
DD2B-	18	CLC	C = 0 si libération effective
DD2C-	60	RTS	

XCREAY crée une table piste/secteur de AY secteurs

Drôle de titre qui ne correspond pas bien à ce que semble faire cette routine: marquer dans la bitmap que le secteur Y de la piste A est occupé! En sortie, C = 1 si ce secteur était déjà occupé sinon avec C = 0.

DD2D-	20 D6 DC	JSR DCD6	calcule à quel octet X de la bitmap correspond ce secteur. Au retour: un bit de A est à 1 et représente le secteur à libérer
DD30-	49 FF	EOR #FF	inverse tous les bits du masque A
DD32-	3D 10 C2	AND C210,X	prend dans A l'octet situé à la X ème position après le début de la bitmap et force à 0 le bit correspondant au masque
DD35-	DD 10 C2	CMP C210,X	teste si l'octet est inchangé
DD38-	F0 F2	BEQ DD2C	si oui, simple RTS en DD2C avec C = 1
DD3A-	9D 10 C2	STA C210,X	réécrit cet octet s'il a été modifié
DD3D-	AD 02 C2	LDA C202	et décrémente le nombre d'octets libres
DD40-	D0 03	BNE DD45	
DD42-	CE 03 C2	DEC C203	
DD45-	CE 02 C2	DEC C202	
DD48-	18	CLC	C = 0 si marquage d'occupation effectif
DD49-	60	RTS	

SUITE DES COMMANDES SEDORIC
(AVEC QUELQUES ROUTINES ASSOCIÉES, D'USAGE GÉNÉRAL)

EXÉCUTION DE LA COMMANDE SEDORIC SAVEM

Rappel de la syntaxe

SAVEM nom_de_fichier_non_ambigu(,AUTO)
SAVEM nom_de_fichier_non_ambigu(,A adresse_début)(,E adresse_fin)(,AUTO)
SAVEM nom_de_fichier_non_ambigu(,A adresse_début)(,E adresse_fin)(,T adresse_d'exécution)

Dans sa première forme, sauve sur disquette un programme BASIC et ceci en mode exécution automatique au chargement si ,AUTO est précisé. Dans les deux autres formes, sauve un bloc mémoire selon les paramètres ,A (début de fichier) et ,E (fin de fichier) et ceci en mode exécution automatique au chargement si ,AUTO est précisé (adresse de début du bloc) ou si ,T (adresse spéciale d'exécution) est indiqué.

Le fichier sauvegardé est ajouté à la suite du fichier existant. Le "M" de SAVEM signifie MERGE, ce qui n'est pas très heureux, car le fichier ainsi sauvé n'est pas "mélangé" au(x) précédent(s), mais ajouté à la suite.

Non documenté

Voir ci-dessous à la commande SAVE.

DD4A-	A9 40	LDA #40	code #40 pour SAVEM (c'est à dire b6 à 1 (0100 0000))
DD4C-	2C A9 C0	BIT C0A9	et continue en DD55...

EXÉCUTION DE LA COMMANDE SEDORIC SAVEU

Rappel de la syntaxe

SAVEU nom_de_fichier_non_ambigu(,AUTO)
SAVEU nom_de_fichier_non_ambigu(,A adresse_début)(,E adresse_fin)(,AUTO)
SAVEU nom_de_fichier_non_ambigu(,A adresse_début)(,E adresse_fin)(,T adresse_d'exécution)

Dans sa première forme, sauve sur disquette un programme BASIC et ceci en mode exécution automatique au chargement si ,AUTO est précisé. Dans les deux autres formes, sauve un bloc mémoire selon les paramètres ,A (début de fichier) et ,E (fin de fichier) et ceci en mode exécution automatique au chargement si ,AUTO est précisé (adresse de début du bloc) ou si ,T (adresse spéciale d'exécution) est indiqué.

Si un fichier de même nom existe déjà, il sera conservé avec l'extension ".BAK". Le "U" de SAVEU signifie UNDELETE: le fichier précédent de même nom est préservé. Remarque, compte tenu de la place disponible avec SEDORIC V3.0 sur les disquettes 3", cette commande pourrait être utilisée systématiquement à la place de SAVE.

Non documenté

Voir ci-dessous à la commande SAVE.

```
DD4D- A9 C0     LDA #C0     CODE #C0 pour SAVEU (c'est à dire b6 et b7 à 1 (1100 0000))
DD4F- 2C A9 80  BIT 80A9    et continue en DD55...
```

EXÉCUTION DE LA COMMANDE SEDORIC SAVE

Rappel de la syntaxe

SAVE nom_de_fichier_non_ambigu(,AUTO)
SAVE nom_de_fichier_non_ambigu(,A adresse_début)(,E adresse_fin)(,AUTO)
SAVE nom_de_fichier_non_ambigu(,A adresse_début)(,E adresse_fin)(,T adresse_d'exécution)

Dans sa première forme, sauve sur disquette un programme BASIC et ceci en mode exécution automatique au chargement si ,AUTO est précisé. Dans les deux autres formes, sauve un bloc mémoire selon les paramètres ,A (début de fichier) et ,E (fin de fichier) et ceci en mode exécution automatique au chargement si ,AUTO est précisé (adresse de début du bloc) ou si ,T (adresse spéciale d'exécution) est indiqué.

Si un fichier de même nom existe déjà, un message d'erreur sera généré. Remarque, compte tenu de la place disponible avec SEDORIC V3.0 sur les disquettes 3", cette commande pourrait être remplacée systématiquement par la commande SAVEU.

Non documenté

La gestion des paramètres est catastrophique. ATTENTION À CE QUE VOUS TAPEZ.

Voici quelques indications:

-Les paramètres ,A ,E et ,T peuvent être dans n'importe quel ordre, mais le paramètre ,AUTO (s'il est présent) doit obligatoirement être le dernier, sinon une SYNTAX_ERROR sera générée.

-Les options ,T et ,AUTO peuvent cohabiter (!), mais l'option ,AUTO (obligatoirement en dernier) l'emportera, en imposant obligatoirement l'adresse d'exécution #501 si le programme est de type BASIC ou l'adresse indiquée après le paramètre ,A si le programme est de type "bloc de mémoire" (langage machine).

-Si ni le paramètre ,A ni le paramètre ,E n'est présent, un programme BASIC sera sauvé. Si le paramètre ,T est malgré tout présent, ce programme BASIC sera sauvé en AUTO et l'adresse d'exécution sera #501, quelque soit l'adresse indiquée à la suite de ,T bien que celle-ci soit reportée comme adresse d'exécution.

-Si le paramètre ,A est présent, mais pas le paramètre ,E une SYNTAX_ERROR sera générée (logique). Mais si le paramètre ,E est présent, mais pas le paramètre ,A un bloc mémoire sera sauvé, allant de #501 à l'adresse indiquée à la suite de ,E.

-Le paramètre ,T peut être présent plusieurs fois, l'adresse d'exécution finalement retenue sera celle qui

suit le dernier ,T (sauf s'il y a encore un ,AUTO après!).

-La validité de l'adresse d'exécution indiquée après le paramètre ,T n'est pas vérifiée. Si vous faites une faute de frappe et quelle tombe en dehors du bloc sauvé, rien ne vous le dira... jusqu'au moment de l'exécution!

A titre d'exemple, voici quelques immondices:

SAVE"BOB1",E#5600	sera accepté et BOB1,V affichera	0501 5600 40 0000
SAVE"BOB2",E#5600,T#6000	sera accepté et BOB2,V affichera	0501 5600 41 6000
SAVE"BOB3",E#5600,T#6000,AUTO	sera accepté et BOB3,V affichera	0501 5600 41 0501
SAVE"BOB3",T#5600,A#5500,E#6000,T#5700,T#5800	donne	5500 6000 41 5800
SAVE"BOB4",T#5600,A#5500,E#6000,T#5700,T#5800,AUTO	donne	5500 6000 41 5500

Je suis sûr que, même si vous avez suivi mes explications, vous aurez tout oublié demain. Tenez-vous en donc strictement à la syntaxe de base, en vérifiant bien avant de presser sur la touche RETURN!

Enfin, toujours le même problème avec l'utilisation possible des minuscules pour entrer les commandes SEDORIC: le ",AUTO" doit être tapé en MAJUSCULES.

DD50-	A9 80	LDA #80	CODE #80 pour SAVE (c'est à dire b7 à 1 (1000 0000))
DD52-	2C A9 00	BIT 00A9	et continue en DD55...

EXÉCUTION DE LA COMMANDE SEDORIC SAVEO

Rappel de la syntaxe

SAVEO nom_de_fichier_non_ambigu(,AUTO)
SAVEO nom_de_fichier_non_ambigu(,A adresse_début)(,E adresse_fin)(,AUTO)
SAVEO nom_de_fichier_non_ambigu(,A adresse_début)(,E adresse_fin)(,T adresse_d'exécution)

Dans sa première forme, sauve sur disquette un programme BASIC et ceci en mode exécution automatique au chargement si ,AUTO est précisé. Dans les deux autres formes, sauve un bloc mémoire selon les paramètres ,A (début de fichier) et ,E (fin de fichier) et ceci en mode exécution automatique au chargement si ,AUTO est précisé (adresse de début du bloc) ou si ,T (adresse spéciale d'exécution) est indiqué.

Si un fichier de même nom existe déjà, il sera écrasé. S'il est protégé en écriture, un message d'erreur sera généré et rien ne se passera. Si aucun fichier de ce nom n'existe, la commande se comporte comme un simple SAVE. Le "O" de SAVEO signifie OVER, ce qui signifie que le nouveau fichier sera écrit par dessus l'ancien.

Non documenté

Attention: l'ancien fichier est d'abord effacé, puis le nouveau est sauvé. Si un problème se produit entre deux, vous avez tout perdu! Utilisez plutôt la commande SAVEU. Voir également ci-dessus le non documenté de la commande SAVE.

DD53-	A9 00	LDA #00	CODE #00 pour SAVEO (c'est à dire aucun bit à 1)

Suite commune SAVEM, SAVEU, SAVE et SAVEO: analyse des paramètres

DD55-	20 28 DE	JSR DE28	XDEFSA place A dans VSALO0, #80 dans FTYPE, #0000 dans EXSALO et valeurs de début et fin BASIC dans DESALO et FISALO
DD58-	20 4F D4	JSR D44F	XNF lit nom de fichier non-ambigu à TXTPTR et l'écrit dans BUFNOM
DD5B-	20 9E D7	JSR D79E	XNOWILD recherche "?" dans BUFNOM revient avec C = 1 si pas trouvé ou génère une "WILDCARD(S)_NOT_ALLOWED_ERROR" si trouvé

Sauve s'il n'y a plus de paramètre à analyser

Si aucun paramètre n'a été rencontré, sauve un programme BASIC, non AUTO, par défaut.
Sinon sauve en fonction des paramètres qui ont été rencontrés après rebouclage en ce point.

DD5E-	20 9E D3	JSR D39E	XCRGOT relit le caractère à TXTPTR (sans incrémenter TXTPTR = CHRGOT), puis le convertit en MAJUSCULE, les espaces sont sautés, Z = 1 si fin d'instruction (0 ou :), C = 0 si caractère chiffre 0 à 9 (soit #30 à #39), sinon C = 1, Y et X inchangés
DD61-	D0 03	BNE DD66	saute la ligne suivante s'il y a des paramètres
DD63-	4C 0B DE	JMP DE0B	sinon, continue pour sauver BASIC par défaut

Paramètre ",T" ?

C'est curieux 1) de commencer par ,T et 2) de pouvoir mettre plusieurs paramètres ,T à la suite, sans générer de SYNTAX_ERROR!

DD66-	20 2C D2	JSR D22C	D067/ROM exige une "," lit le caractère suivant et le convertit en MAJUSCULE
DD69-	C9 54	CMP #54	est-ce un "T" (adresse d'exécution spéciale)
DD6B-	D0 1C	BNE DD89	sinon continue en DD89 (suite analyse syntaxe)
DD6D-	20 98 D3	JSR D398	XCRGET incrémente TXTPTR, lit un caractère (CHRGET), les espaces sont sautés, le met dans A, le convertit en MAJUSCULE, Z = 1 si fin d'instruction (0 ou :), C = 0 si chiffre 0 à 9 (soit #30 à #39), sinon C =1. Y et X inchangés
DD70-	20 FA D2	JSR D2FA	E853/ROM évalue un nombre non signé à TXTPTR et revient avec ce nombre dans YA, 33/34 et D3/D4 et mise à jour de TXTPTR sur l'octet suivant ce nombre (C = 0, N et Z positionnés selon l'octet de poids faible)
DD73-	8C 56 C0	STY C056	
DD76-	8D 57 C0	STA C057	place cette adresse dans EXSALO (adresse d'exécution)
DD79-	4E 51 C0	LSR C051	
DD7C-	38	SEC	force à 1 le b0 de FTYPE (pour AUTO)
DD7D-	2E 51 C0	ROL C051	
DD80-	D0 DC	BNE DD5E	rebouclage forcé en DD5E pour sauver, sinon pour analyser le paramètre suivant
DD82-	A9 40	LDA #40	si ,T ,A ou ,E
DD84-	8D 51 C0	STA C051	FTYPE = #40 (bloc de données)
DD87-	D0 D5	BNE DD5E	rebouclage forcé en DD5E pour sauver, sinon pour analyser le paramètre suivant

Paramètre ,A ?

DD89-	C9 41	CMP #41	est-ce un "A" (adresse de début de fichier)
DD8B-	D0 0E	BNE DD9B	sinon continue en DD9B (suite analyse syntaxe)
DD8D-	20 98 D3	JSR D398	XCRGET incrémente TXTPTR, lit un caractère (CHRGET), les espaces sont sautés, le met dans A, le convertit en MAJUSCULE, Z = 1 si fin d'instruction (0 ou :), C = 0 si chiffre 0 à 9 (soit #30 à #39), sinon C =1. Y et X inchangés
DD90-	20 FA D2	JSR D2FA	E853/ROM évalue un nombre non signé à TXTPTR et revient avec ce nombre dans YA, 33/34 et D3/D4 et mise à jour de TXTPTR sur l'octet suivant ce nombre (C = 0, N et Z positionnés selon l'octet de poids faible)
DD93-	8C 52 C0	STY C052	
DD96-	8D 53 C0	STA C053	place cette adresse dans DESALO (adresse de début de fichier)
DD99-	90 E7	BCC DD82	rebouclage forcé (C = 0 car chiffre en DD8D)

Paramètre ,E ?

DD9B-	C9 45	CMP #45	est-ce un "E" (adresse de fin de fichier)
DD9D-	D0 0E	BNE DDAD	sinon continue en DDAD (suite analyse syntaxe)
DD9F-	20 98 D3	JSR D398	XCRGET incrémente TXTPTR, lit un caractère (CHRGET), les espaces sont sautés, le met dans A, le convertit en MAJUSCULE, Z = 1 si fin d'instruction (0 ou :), C = 0 si chiffre 0 à 9 (soit #30 à #39), sinon C =1. Y et X inchangés
DDA2-	20 FA D2	JSR D2FA	E853/ROM évalue un nombre non signé à TXTPTR et revient avec ce nombre dans YA, 33/34 et D3/D4 et mise à jour de TXTPTR sur l'octet suivant ce nombre (C = 0, N et Z positionnés selon l'octet de poids faible)
DDA5-	8C 54 C0	STY C054	
DDA8-	8D 55 C0	STA C055	place cette adresse dans FISALO (adresse de fin de fichier)
DDAB-	90 D5	BCC DD82	rebouclage forcé (C = 0 car chiffre en DD9F)

Paramètre ,AUTO ?

DDAD-	C9 C7	CMP #C7	est-ce le token "AUTO"? (dernière possibilité)
DDAF-	D0 72	BNE DE23	"SYNTAX_ERROR", car aucun autre paramètre autorisé
DDB1-	20 98 D3	JSR D398	XCRGET incrémente TXTPTR, lit un caractère (CHRGET), les espaces sont sautés, le met dans A, le convertit en MAJUSCULE, Z = 1 si fin d'instruction (0 ou :), C = 0 si chiffre 0 à 9 (soit #30 à #39), sinon C =1. Y et X inchangés
DDB4-	D0 6D	BNE DE23	"SYNTAX_ERROR", après AUTO il faut "fin de commande"
DDB6-	4E 51 C0	LSR C051	
DDB9-	38	SEC	force à 1 le b0 de FTYPE (pour AUTO)
DDBA-	2E 51 C0	ROL C051	
DDBD-	30 4C	BMI DE0B	continue en DE0B si BASIC
DDBF-	AD 52 C0	LDA C052	
DDC2-	AC 53 C0	LDY C053	copie DESALO dans EXSALO (adresse d'exécution)
DDC5-	8D 56 C0	STA C056	**NB:** ,AUTO annule ,T adresse spéciale d'exécution
DDC8-	8C 57 C0	STY C057	
DDCB-	90 3E	BCC DE0B	branchement forcé en DE0B (C = 0 en DDB6/DDBA)

EXÉCUTION DE LA COMMANDE SEDORIC KEYSAVE

Rappel de la syntaxe

KEYSAVE nom_de_fichier_non_ambigu

Cette commande, qui est équivalente à SAVE nom_de_fichier,A#C800,E#C9DD permet de sauvegarder la table "KEYDEF" (C800 à C87F), celle des 16 commandes re-definissables (C880 à C97F) et celle des 16 commandes pré-definies (C980 à C9DD), c'est à dire la configuration des touches de fonctions.

Voir un exemple de l'utilisation de cette commande dans le "Non documenté" de la commande ACCENT (en EB91). Voir en C880 pour les différents fichiers SEDORIC3*.KEY livrés avec SEDORIC V3.0.

Saisie du paramètre et exécution

En DDDB, j'ai allongé la zone mémoire sauvée par **KEYSAVE**, pour incorporer les commandes pré-definies. L'adresse de début reste C800, mais l'adresse de fin passe de C97F à C9DD. L'octet modifié (le LL de FISALO soit #7F devient #DD) est indiqué en gras. Les versions précédentes de SEDORIC ne sauvaient pas la table PREDEF. Les nouveaux fichiers générés ont la même taille, pourquoi se priver d'une possibilité d'édition de la table PREDEF?

DDCD-	20 4F D4	JSR D44F	XNF lit nom de fichier non-ambigu à TXTPTR et l'écrit dans BUFNOM
DDD0-	A9 00	LDA #00	
DDD2-	A0 C8	LDY #C8	
DDD4-	8D 52 C0	STA C052	adresse C800 -> DESALO (début de fichier)
DDD7-	8C 53 C0	STY C053	
DDDA-	A9 **DD**	LDA #DD	
DDDC-	A0 C9	LDY #C9	adresse C97F -> AY pour FISALO (fin de fichier)
DDDE-	D0 1E	BNE DDFE	branchement forcé en #DDFE car Y non nul

EXÉCUTION DE LA COMMANDE SEDORIC ESAVE

Rappel de la syntaxe

ESAVE nom_de_fichier_non_ambigu

Cette commande, qui est équivalante à SAVE nom_de_fichier,A#BB80,E#BFDF en mode TEXT ou à SAVE nom_de_fichier,A#A000,E#BF3F en mode HIRES, permet de sauvegarder l'écran courant qu'il sera possible de recharger à condition d'être dans le même mode.

Saisie du paramètre et exécution

DDE0-	20 4F D4	JSR D44F	XNF lit nom de fichier non-ambigu à TXTPTR et l'écrit dans BUFNOM
DDE3-	AD 1F 02	LDA 021F	0 si mode TEXT, 1 si mode HIRES
DDE6-	D0 08	BNE DDF0	branche si mode HIRES

DDE8-	A2 80	LDX #80	
DDEA-	A0 BB	LDY #BB	valeurs pour adresse BB80 et BFDF (écran TEXT)
DDEC-	A9 DF	LDA #DF	
DDEE-	D0 06	BNE DDF6	branchement forcé en #DDF6 car A non nul
DDF0-	A2 00	LDX #00	
DDF2-	A0 A0	LDY #A0	valeurs pour adresse A000 et BF3F (écran HIRES)
DDF4-	A9 3F	LDA #3F	
DDF6-	8E 52 C0	STX C052	mise à jour de DESALO (début de fichier)
DDF9-	8C 53 C0	STY C053	
DDFC-	A0 BF	LDY #BF	valeur pour adresse BFDF (TEXT) ou BF3F (HIRES)

Suite commune à KEYSAVE et à ESAVE

DDFE-	A2 40	LDX #40	X = # 40 pour FTYPE "bloc de données"
DE00-	20 3B DE	JSR DE3B	AY -> FISALO, X -> FTYPE et #0000 -> EXSALO
DE03-	A9 C0	LDA #C0	
DE05-	8D 4D C0	STA C04D	#C0 -> VSALO0 (code de type SAVEU)
DE08-	20 9E D7	JSR D79E	XNOWILD recherche "?" dans BUFNOM revient avec C = 1 si pas trouvé ou génère une "WILDCARD(S)_NOT_ALLOWED_ERROR" si trouvé

Suite commune pour SAVE*, KEYSAVE, ESAVE et CREATEW

DE0B-	38	SEC	prépare une soustraction
DE0C-	AD 54 C0	LDA C054	
DE0F-	ED 52 C0	SBC C052	
DE12-	8D 4F C0	STA C04F	calcule FISALO - DESALO (adresse fin - adresse début)
DE15-	AD 55 C0	LDA C055	et résultat dans LGSALO (longueur du fichier)
DE18-	ED 53 C0	SBC C053	
DE1B-	8D 50 C0	STA C050	
DE1E-	B0 7C	BCS DE9C	si adresse fin > adresse début, continue au sous-programme XSAVEB (sauve fichier selon BUFNOM, VSALO0, VSALO1, DESALO, FISALO, EXSALO)
DE20-	A2 08	LDX #08	sinon donnera une "ILLEGAL_QUANTITY_ERROR"
DE22-	2C A2 09	BIT 09A2	et continue en D67E
DE23-	A2 09	LDX #09	pour "SYNTAX_ERROR"
DE25-	4C 7E D6	JMP D67E	incrémente X et traite l'erreur n° X

XDEFSA Positionne les valeurs pour sauver le programme BASIC (XSAVEB)

DE28-	8D 4D C0	STA C04D	place dans VSALO0 le code "type de SAVE": #00 pour SAVEO, #40 pour SAVEM, #80 pour SAVE et #C0 pour SAVEU
DE2B-	A5 9A	LDA 9A	
DE2D-	A4 9B	LDY 9B	pointeur "début de BASIC"
DE2F-	8D 52 C0	STA C052	copié dans DESALO (début de fichier)
DE32-	8C 53 C0	STY C053	
DE35-	A5 9C	LDA 9C	
DE37-	A4 9D	LDY 9D	pointeur "fin de BASIC/début des variables"
DE39-	A2 80	LDX #80	code pour fichier BASIC non AUTO

DE3B-	8D 54 C0	STA C054	copié dans FISALO (fin de fichier)
DE3E-	8C 55 C0	STY C055	
DE41-	8E 51 C0	STX C051	et dans FTYPE (type de fichier)
DE44-	A9 00	LDA #00	
DE46-	8D 56 C0	STA C056	mise à zéro de EXSALO (adresse d'exécution)
DE49-	8D 57 C0	STA C057	
DE4C-	60	RTS	

EXÉCUTION DE LA COMMANDE SEDORIC CREATEW

Rappel de la syntaxe

CREATEW nom_de_fichier_non_ambigu

Permet de créer un masque de saisie de 25 lignes de 40 caractères utilisable avec la commande WINDOW. Cette commande, qui est interdite en mode HIRES est en fait un éditeur d'écran de type "pleine page" dans lequel tous les caractères peuvent être utilisés. **CTRL/S** permet de sauver l'écran et **CTRL/C** permet d'en sortir sans sauver. Il est possible de placer dans ce masque des "champs de données", matérialisés à l'écran comme un pavé plein, à l'aide de **CTRL/W**. Un écran partiel (de BBD0 à BFB7, c'est à dire sans la ligne service, ni la première ligne, ni la dernière) sera sauvegardé sous le nom de fichier spécifié de type "WINDOW" (FTYPE = #60).

Non documenté

Bogue dans le manuel concernant les lignes utilisées par CREATEW. La ligne "service" et la première ligne (n°0) ainsi que la dernière ligne de l'écran (n°26) ne sont pas utilisées. Voir aussi à la commande WINDOW les problèmes soulevés par la longueur des champs.

Saisie du paramètre et exécution

DE4D-	20 4F D4	JSR D44F	XNF lit nom de fichier non-ambigu à TXTPTR et l'écrit dans BUFNOM
DE50-	20 DE DF	JSR DFDE	vérifie que l'on est bien en mode TEXT, sinon génère une "DISP_TYPE_MISMATCH_ERROR", réinitialise la pile et retourne au "Ready"

Début de la saisie de touche

DE53-	20 02 D3	JSR D302	JSR EB78/ROM si touche frappée alors N = 1 et A = code ASCII correspondant, sinon N = 0
DE56-	10 FB	BPL DE53	reboucle jusqu'à obtenir une réponse
DE58-	C9 03	CMP #03	est-ce CTRL/C? (pour sortir = annulation)
DE5A-	F0 F0	BEQ DE4C	si oui, simple RTS (seule sortie de cette commande)
DE5C-	C9 13	CMP #13	est-ce CTRL/S? (pour sauver l'écran)
DE5E-	D0 1C	BNE DE7C	sinon continue en DE7C (suite analyse syntaxe)

Sauve le fichier

DE60-	20 40 D7	JSR D740	si oui XCUROFF cache le curseur (= vidéo normale)
DE63-	A9 D0	LDA #D0	
DE65-	A0 BB	LDY #BB	
DE67-	8D 52 C0	STA C052	adresse BBD0 -> DESALO (début de fichier)
DE6A-	8C 53 C0	STY C053	
DE6D-	A9 B7	LDA #B7	
DE6F-	A0 BF	LDY #BF	adresse BFB7 -> AY pour FISALO (fin de fichier)
DE71-	A2 60	LDX #60	X = # 60 pour FTYPE "WINDOW"
DE73-	20 00 DE	JSR DE00	AY -> FISALO, X -> FTYPE, #0000 -> EXSALO et #C0 -> VSALO0 (code de type SAVEU). vérifie absence de jocker dans BUFNOM. Calcule LGSALO (longueur du fichier) et enfin sauve fichier selon BUFNOM, VSALO0, VSALO1, DESALO, FISALO, EXSALO
DE76-	20 3E D7	JSR D73E	XCURON rend le curseur visible (= vidéo inverse)
DE79-	4C 53 DE	JMP DE53	reboucle en attente de touche

Champ de données ?

DE7C-	C9 17	CMP #17	est-ce CTRL/W? (pour champ de données)
DE7E-	D0 0E	BNE DE8E	sinon continue en DE8E (suite analyse touche)
DE80-	AC 69 02	LDY 0269	si oui Y = colonne curseur TEXT (0 à 39)
DE83-	A9 7F	LDA #7F	A = #7F (carré plein de couleur INK)
DE85-	AC 69 02	LDY 0269	Y = colonne curseur TEXT (0 à 39) (encore? ce code n'étant pas appelé d'ailleurs, il s'agit d'une bogue mineure due à la fatigue du programmeur)
DE88-	91 12	STA (12),Y	place A à l'adresse de la ligne du curseur TEXT
DE8A-	A9 09	LDA #09	A = #09 (CTRL/I = flèche droite)
DE8C-	D0 09	BNE DE97	branchement forcé en DE97

Affichage des autres touches

DE8E-	C9 0D	CMP #0D	est-ce RETURN? (code de CR)
DE90-	D0 05	BNE DE97	sinon continue en DE97
DE92-	20 2A D6	JSR D62A	si oui XAFCAR affiche ce CR suivi d'un LF:
DE95-	A9 0A	LDA #0A	A = LF (line feed)
DE97-	20 2A D6	JSR D62A	XAFCAR affiche le caractère ASCII contenu en A
DE9A-	D0 B7	BNE DE53	reboucle en attente de touche

XSAVEB Sauve le fichier selon BUFNOM, VSALO0/1, DESALO, FISALO, EXSALO

Routine d'intérêt général utilisée par SAVE*, KEYSAVE, ESAVE et CREATEW

DE9C-	78	SEI	interdit les interruptions
DE9D-	20 2D DB	JSR DB2D	vérifie que la disquette en place est bien une disquette SEDORIC, cherche le fichier dans le catalogue, revient avec le bon secteur de catalogue en place (de coordonnées POSNMP et POSNMS) et avec X = POSNMX, pointant sur "l'entrée" cherchée ou avec Z = 1 si le fichier n'a pas été trouvé
DEA0-	F0 6F	BEQ DF11	branche si pas encore de fichier de ce nom

Sous-programme "il existe déjà un fichier de ce nom dans le catalogue"

Analyse le type de SAVE présent dans VSALO0

DEA2-	AD 4D C0	LDA C04D	VSALO0 contient-il #00?
DEA5-	F0 16	BEQ DEBD	si oui, continue en DEBD (SAVEO)
DEA7-	C9 80	CMP #80	VSALO0 contient-il #80?
DEA9-	F0 0D	BEQ DEB8	si oui, continue en DEB8 (SAVE) "FILE_ALREADY_EXIST_ERROR"
DEAB-	C9 C0	CMP #C0	VSALO0 contient-il #C0?
DEAD-	F0 16	BEQ DEC5	si oui, continue en DEC5 (SAVEU ou KEYSAVE ou ESAVE ou CREATEW) sinon, il s'agit de SAVEM: le fichier à sauvegarder sera ajouté à la suite du fichier existant pour n'en former qu'un

Suite pour SAVEM

DEAF-	20 07 DB	JSR DB07	XCABU copie la ligne d'"entrée" de catalogue à POSNMX (BUF3) dans BUFNOM (cette mise à jour inclut PSDESP coordonnées du descripteur principal et NSTOTP nombre de secteurs totaux + PROT)
DEB2-	4C 1B DF	JMP DF1B	et continue en DF1B

Traitement des erreurs

DEB5-	A2 02	LDX #02	pour une "INVALID_FILE_NAME_ERROR"
DEB7-	2C A2 06	BIT 06A2	et continue en DEBA

Suite pour SAVE (erreur)

DEB8-	A2 06	LDX #06	pour une "FILE_ALREADY_EXIST_ERROR"
DEBA-	4C 7E D6	JMP D67E	incrémente X et traite l'erreur n° X

Suite pour SAVEO (DEL ancien fichier)

DEBD-	20 64 E2	JSR E264	délète le fichier indexé à POSNMX dans BUF3
DEC0-	B0 2D	BCS DEEF	si C = 1, affiche “nom + IS PROTECTED” et sort
DEC2-	4C 11 DF	JMP DF11	si C = 0 ,continue en DF11

Suite pour SAVEU, KEYSAVE, ESAVE et CREATEW

DEC5-	A0 02	LDY #02	
DEC7-	B9 32 C0	LDA C032,Y	lit dans BUFNOM les 3 caractères de
DECA-	48	PHA	l'extension et les empile (pour sauver
DECB-	88	DEY	ultérieurement le nouveau fichier)
DECC-	10 F9	BPL DEC7	
DECE-	A0 02	LDY #02	index pour lecture 3 caractères
DED0-	B9 32 C0	LDA C032,Y	lit dans BUFNOM les caractères de l'extension et
DED3-	D9 FA CC	CMP CCFA,Y	les compare avec BAK (table des extensions)
DED6-	D0 05	BNE DEDD	si différent, continue en DEDD
DED8-	88	DEY	caractère suivant
DED9-	10 F5	BPL DED0	reboucle tant que 3 caractères n'ont pas été lus
DEDB-	30 D8	BMI DEB5	si aucune différence trouvée, "INVALID_FILE_NAME_ERROR" car on

			ne peut sauver avec SAVEU un fichier "*.BAK"
DEDD-	A2 03	LDX #03	index pour lecture dans la table des extensions
DEDF-	20 4A D3	JSR D34A	lit "BAK" dans table CCF7, le copie dans BUFNOM à la place de l'extension du fichier à sauver
DEE2-	20 30 DB	JSR DB30	XTVNM cherche dans le catalogue un éventuel "nom.BAK" comme indiqué dans BUFNOM et revient avec POSNMX POSNMP et POSNMS ou Z = 1
DEE5-	F0 0A	BEQ DEF1	si pas trouvé, OK continue en DEF1
DEE7-	20 64 E2	JSR E264	si trouvé, efface ce "nom.BAK" (sera remplacé)
DEEA-	90 05	BCC DEF1	si délétion réussie (C = 0), continue en DEF1
DEEC-	68	PLA	sinon, affiche “nom + IS PROTECTED”
DEED-	68	PLA	dépile les 3 caractères de l'extension
DEEE-	68	PLA	devenus inutiles
DEEF-	58	CLI	autorise les interruptions et retourne
DEF0-	60	RTS	

Suite pour SAVEU, KEYSAVE, ESAVE et CREATEW

L’ancien fichier en .BAK a été effacé ou n’existait pas, renomme le fichier existant en .BAK

DEF1-	A0 00	LDY #00	
DEF3-	68	PLA	
DEF4-	99 32 C0	STA C032,Y	récupère les 3 caractères de l'extension sur la pile
DEF7-	C8	INY	et les remet dans BUFNOM pour la recherche de
DEF8-	C0 03	CPY #03	l'ancien fichier et la sauvegarde du nouveau
DEFA-	D0 F7	BNE DEF3	
DEFC-	20 30 DB	JSR DB30	XTVNM cherche dans le catalogue le fichier BUFNOM qui deviendra ".BAK" et revient avec POSNMX POSNMP et POSNMS ou Z = 1
DEFF-	AE 27 C0	LDX C027	X = POSNMX début de “l’entrée” dans le secteur de catalogue
DF02-	B9 FA CC	LDA CCFA,Y	lit BAK dans la table des extensions
DF05-	9D 09 C3	STA C309,X	et le copie dans “l’entrée” du secteur de catalogue
DF08-	E8	INX	caractère suivant dans "entrée" du secteur de catalogue
DF09-	C8	INY	caractère suivant dans table des extensions
DF0A-	C0 03	CPY #03	et reboucle tant que 3 caractères
DF0C-	D0 F4	BNE DF02	n'ont pas été copiés
DF0E-	20 82 DA	JSR DA82	XSCAT sauve le secteur de catalogue selon POSNMP et POSNMS

Mise à zéro des 2 derniers octets de BUFNOM

Suite pour tous s'il n'existe pas déjà de fichier à ce nom, suite pour SAVEO après DEL de l'ancien nom, suite pour SAVEU, KEYSAVE, ESAVE et CREATEW après DEL de l'ancien ".BAK" et mise de l'ancien fichier en ".BAK".

DF11-	A2 03	LDX #03	
DF13-	A9 00	LDA #00	mise à 0 de C035 à C038
DF15-	9D 35 C0	STA C035,X	C035/36 PSDESP coordonnées du secteur descripteur principal
DF18-	CA	DEX	C037/38 NSTOTP nombre secteur totaux + PROT
DF19-	10 FA	BPL DF15	

Suite pour tous les précédents et pour SAVEM

DF1B-	AE 50 C0	LDX C050	X = HH de LGSALO (nombre de secteurs pleins)
DF1E-	A0 00	LDY #00	Y = #00
DF20-	E8	INX	incrémente le nombre de secteurs nécessaires
DF21-	8A	TXA	et le passe dans A
DF22-	D0 01	BNE DF25	saut forcé de l'instruction suivante
DF24-	C8	INY	

Ecriture de tous les secteurs (sauf le dernier)

DF25-	20 C0 DB	JSR DBC0	écriture du ou des descripteurs du fichier à sauver. Revient avec le nombre de secteurs à sauver dans NSSAV (C05A/5B), les coordonnées du premier secteur descripteur dans PSDESC (C05C/5D), le nombre de descripteurs utilisés dans NSDESC (C05E) et premier descripteur en place
DF28-	AD 52 C0	LDA C052	
DF2B-	AC 53 C0	LDY C053	
DF2E-	88	DEY	DESALO - #100 -> RWBUF
DF2F-	8D 03 C0	STA C003	(sera re-ajusté au début de la boucle)
DF32-	8C 04 C0	STY C004	
DF35-	A0 0A	LDY #0A	Y pointe 2 octets avant début de la liste des coordonnées piste/secteur des secteurs à sauver (sera re-ajusté au début de la boucle)
DF37-	EE 04 C0	INC C004	augmente RWBUF de #100 (début zone à sauver)
DF3A-	AD 50 C0	LDA C050	A = HH de LGSALO (longueur du fichier)
DF3D-	F0 17	BEQ DF56	branche s'il ne reste plus de secteur complet
DF3F-	CE 50 C0	DEC C050	sinon décrémente HH de LGSALO
DF42-	20 28 E2	JSR E228	Y = Y + 2 (chargement éventuel du descripteur suivant si nécessaire). Revient soit avec C = 0 (descripteur en place et Y pointant sur les coordonnées du secteur suivant du fichier à sauver/charger), soit avec C = 1 s'il n'y a pas de suivant.
DF45-	B9 00 C1	LDA C100,Y	
DF48-	8D 01 C0	STA C001	lit n° piste -> PISTE
DF4B-	B9 01 C1	LDA C101,Y	
DF4E-	8D 02 C0	STA C002	lit n° secteur -> SECTEUR
DF51-	20 A4 DA	JSR DAA4	XSVSEC écrit un secteur selon DRIVE, PISTE, SECTEUR et RWBUF
DF54-	F0 E1	BEQ DF37	rebouclage forcé (n° secteur jamais nul)

Ecriture du dernier secteur (généralement incomplet)

DF56-	20 28 E2	JSR E228	Y = Y + 2 (chargement éventuel du descripteur suivant si nécessaire)
DF59-	B9 00 C1	LDA C100,Y	
DF5C-	48	PHA	
DF5D-	B9 01 C1	LDA C101,Y	empile les coordonnées du dernier secteur
DF60-	48	PHA	
DF61-	20 CE DA	JSR DACE	XVBUF1 rempli BUF1 de 0
DF64-	AD 03 C0	LDA C003	
DF67-	AC 04 C0	LDY C004	sauve RWBUF dans F2/F3 (pour lecture des
DF6A-	85 F2	STA F2	derniers octets significatifs du fichier)
DF6C-	84 F3	STY F3	
DF6E-	A0 FF	LDY #FF	pour Y = #00 en début de boucle

DF70-	C8	INY	vise octet à recopier
DF71-	B1 F2	LDA (F2),Y	lecture octet du dernier segment du fichier
DF73-	99 00 C1	STA C100,Y	écriture dans BUF1
DF76-	CC 4F C0	CPY C04F	compare index avec LL de LGSALO (longueur du fichier) qui représente en fait la longueur du dernier segment du fichier
DF79-	D0 F5	BNE DF70	reboucle tant que le dernier octet du fichier n'a pas été copié dans le buffer provisoire dont la fin restera vide (#00)
DF7B-	68	PLA	
DF7C-	A8	TAY	récupère les coordonnées du dernier secteur
DF7D-	68	PLA	
DF7E-	20 91 DA	JSR DA91	XSBUF1 sauve BUF1 à la piste A et le secteur Y

Mise à jour du nombre de secteurs totaux du fichier

DF81-	18	CLC	prépare addition
DF82-	AD 5A C0	LDA C05A	LL de NSSAV (nombre de secteurs sauvés)
DF85-	6D 5E C0	ADC C05E	+
DF88-	90 03	BCC DF8D	NSDESC (nombre de secteurs descripteurs)
DF8A-	EE 5B C0	INC C05B	+
DF8D-	6D 37 C0	ADC C037	LL de NSTOTP (nombre secteurs totaux + PROT)
DF90-	8D 37 C0	STA C037	-> LL de NSTOTP (inclus secteur précédent si SAVEM)
DF93-	AD 38 C0	LDA C038	HH de NSTOTP (mise à 0 des 4 bits poids fort)
DF96-	29 0F	AND #0F	+
DF98-	6D 5B C0	ADC C05B	HH de NSSAV (+ retenue éventuelle de l'addition précédente)
DF9B-	09 40	ORA #40	force b6 du résultat
DF9D-	8D 38 C0	STA C038	-> HH de NSTOTP (inclus secteur précédent si SAVEM)
DFA0-	AD 35 C0	LDA C035	AY coordonnées secteur descripteur principal
DFA3-	AC 36 C0	LDY C036	(a été mis à jour seulement si SAVEM)
DFA6-	F0 1D	BEQ DFC5	branche si pas valable (n° secteur jamais nul)

SAVEM: mise à jour du "lien"

DFA8-	20 5D DA	JSR DA5D	XPBUF1 charge dans BUF1 secteur descripteur principal
DFAB-	AD 00 C1	LDA C100	lit AY coordonnées secteur descripteur suivant
DFAE-	AC 01 C1	LDY C101	(cherche le dernier secteur descripteur)
DFB1-	D0 F5	BNE DFA8	reboucle si valable (n° secteur jamais nul)
DFB3-	AD 5C C0	LDA C05C	si dernier descripteur trouvé (Y = #00)
DFB6-	AC 5D C0	LDY C05D	lit et copie coordonnées du premier secteur descripteur du fichier
DFB9-	8D 00 C1	STA C100	sauvé dans le "lien" du dernier descripteur
DFBC-	8C 01 C1	STY C101	
DFBF-	20 A4 DA	JSR DAA4	XSVSEC sauve descripteur selon DRIVE, PISTE, SECTEUR et RWBUF
DFC2-	4C D4 DF	JMP DFD4	et continue en DFD4

Autres SAVE: mise à jour des coordonnées du descripteur principal et cherche une "entrée" libre dans le catalogue

DFC5-	AD 5C C0	LDA C05C	
DFC8-	AC 5D C0	LDY C05D	lit dans AY les coordonnées du premier secteur descripteur et les copie

DFCB-	8D 35 C0	STA C035	dans PSDESP coordonnées du secteur descripteur principal
DFCE-	8C 36 C0	STY C036	
DFD1-	20 59 DB	JSR DB59	XTRVCA cherche une "entrée" libre dans le catalogue revient avec POSNMX, POSNMP et POSNMS

Suite et fin pour tous les SAVE

DFD4-	20 8A DA	JSR DA8A	l'entrée de cette routine XSMAP sauve BUF2 (bitmap) sur la disquette a été déportée en DC80 pour adaptation à la double bitmap. Un JSR DC80 plus direct aurait fait gagner un peu de temps.
DFD7-	20 EE DA	JSR DAEE	XBUCA copie BUFNOM dans BUF3, à position POSNMX
DFDA-	58	CLI	autorise les interruptions
DFDB-	4C 82 DA	JMP DA82	sauve BUF3 (catalogue) selon POSNMP et POSNMS (RTS)

XVERTXT vérifie si bien en mode TEXT

DFDE-	AD 1F 02	LDA 021F	0 si mode TEXT, 1 si mode HIRES
DFE1-	F0 10	BEQ DFF3	si mode TEXT, simple RTS
DFE3-	4C 6F D1	JMP D16F	sinon JSR C47E/ROM affiche le message d'erreur "DISP_TYPE_MISMATCH" puis effectue un JSR C496/ROM qui réinitialise la pile, affiche " ERROR" et retourne au "Ready"

XDEFLO Positionne les valeurs par défaut pour XLOADA

DFE6-	A9 00	LDA #00	pour remise à zéro
DFE8-	A2 03	LDX #03	de 4 octets
DFEA-	9D 4D C0	STA C04D,X	remet à zéro VSALO0, VSALO1 et 2 octets suivants
DFED-	CA	DEX	c'est à dire les adresses C04D, C04E, C04F et C050
DFEE-	10 FA	BPL DFEA	et reboucle tant que X n'est pas négatif
DFF0-	8E 72 C0	STX C072	#FF -> C072 (flag AUTO si b7=1, STOP si b7=0)
DFF3-	60	RTS	et retourne

DFF4-	4C 23 DE	JMP DE23	"SYNTAX_ERROR"

EXÉCUTION DE LA COMMANDE SEDORIC LOAD

Rappel de la syntaxe

LOAD nom_de_fichier_non_ambigu(,A adresse_de_chargement)(,V)(,J)(,N)

Charge le fichier indiqué à son adresse normale ou à l'adresse spécifiée par l'option ",A" ou à la fin du programme BASIC si l'option ",J" est présente. Si l'option ",V" est utilisée, seul le "status" du fichier est affiché (sans chargement). Enfin l'option ",N" permet de charger en bloquant l'exécution automatique éventuelle du programme.

Cette commande est aussi la suite de l'interpréteur si aucun mot-clé n'a été reconnu, ce qui implique soit un LOAD direct, réduit au seul nom de fichier, soit un changement de drive par défaut, dans les 2 cas

l'entrée est en DFF9 et avait été précédée d'un LDA #00, c'est à dire flag N = 0 et d'un CMP / BEQ c'est à dire C = 1, ce qui indique un nom_de_fichier_non_ambigu.

Non documenté

Les options ",A" et ",J" s'excluent mutuellement, ce qui n'est pas indiqué dans le manuel. Il en est de même pour les options ",V" et ",N". Lorsque l'on utilise le paramètre ",A" avec un groupe de fichiers "joints", seul le fichier principal est relogé. L'option ",V" utilisée en combinaison avec ",A" ou ",J" affiche les nouvelles adresses de début et de fin (très pratique).

Analyse de syntaxe et saisie des paramètres

DFF7-	A9 80	LDA #80	flag N = 1 si commande LOAD nom_de_fichier
DFF9-	20 54 D4	JSR D454	évalue le nom_de_fichier présent à TXTPTR, le met dans BUFNOM, après avoir convertit les éventuels "*" en "?", revient avec le drive validé, avec TXTPTR sauvé sur la pile (pointant sur le caractère suivant le nom_de_fichier) et avec le caractère courant à TXTPTR dans A
DFFC-	20 9E D7	JSR D79E	XNOWILD recherche "?" dans BUFNOM revient avec C = 1 si pas trouvé ou génère une "WILDCARD(S)_NOT_ALLOWED_ERROR" si trouvé
E000-	E6 DF	INC DF	extension de ACC1 (pas trouvé la raison ?)
E002-	20 9E D3	JSR D39E	XCRGOT relit le caractère à TXTPTR (sans incrémenter TXTPTR = CHRGOT), puis le convertit en MAJUSCULE, les espaces sont sautés, Z = 1 si fin d'instruction (0 ou :), C = 0 si caractère chiffre 0 à 9 (soit #30 à #39), sinon C = 1, Y et X inchangés
E005-	F0 4B	BEQ E052	si fin de commande, charge le fichier
E007-	D0 05	BNE E00E	sinon, continue en E00E (recherche les éventuels paramètres)
E009-	20 98 D3	JSR D398	XCRGET incrémente TXTPTR, lit un caractère (CHRGET), les espaces sont sautés, le met dans A, le convertit en MAJUSCULE, Z = 1 si fin d'instruction (0 ou :), C = 0 si chiffre 0 à 9 (soit #30 à #39), sinon C =1. Y et X inchangés
E00C-	F0 44	BEQ E052	si fin de commande, charge le fichier
E00E-	20 2C D2	JSR D22C	D067/ROM exige une "," lit le caractère suivant et le convertit en MAJUSCULE
E011-	A0 40	LDY #40	Y = #40 (drapeau par défaut pour paramètre ",V")
E013-	C9 56	CMP #56	est-ce un "V"?
E015-	F0 06	BEQ E01D	si oui, continue en E01D avec Y = #40
E017-	C9 4E	CMP #4E	est-ce un "N"?
E019-	D0 0C	BNE E027	sinon, continue en E027 (cherche "J" ou "A")
E01B-	A0 80	LDY #80	si oui, Y = #80 (drapeau pour le paramètre ",N")
E01D-	AD 4D C0	LDA C04D	lit valeur actuelle de VSALO0
E020-	D0 D2	BNE DFF4	si pas nul, branche sur vecteur "SYNTAX_ERROR" (sert à exclure coexistence de ",V" et ",N" ou l'existence de plusieurs ",V" ou de plusieurs ",N"dans le même LOAD)
E022-	8C 4D C0	STY C04D	si nul, écrit Y dans VSALO0 qui vaut donc à la fin soit #00 (ni ",V" ni ",N"), soit #40 (",V") ou #80 (",N")
E025-	F0 E2	BEQ E009	branchement forcé en E009 (lecture du caractère suivant)
E027-	C9 4A	CMP #4A	est-ce un "J"?
E029-	D0 09	BNE E034	sinon, continue en E034

E02B-	AD 4E C0	LDA C04E	si oui, lit valeur actuelle de VSALO1
E02E-	D0 C4	BNE DFF4	si pas nul, branche sur vecteur "SYNTAX_ERROR" (sert à exclure coexistence de ",J" et ",A" dans le même LOAD)
E030-	A2 80	LDX #80	si nul, X = #80 (pour paramètre ",J")
E032-	30 17	BMI E04B	branchement forcé en E04B
E034-	C9 41	CMP #41	est-ce un "A"?
E036-	D0 BC	BNE DFF4	sinon, branche sur vecteur "SYNTAX_ERROR" car on a examiné ,V ,N ,J et ,A seuls paramètres possibles
E038-	AD 4E C0	LDA C04E	si oui, lit valeur actuelle de VSALO1
E03B-	D0 B7	BNE DFF4	si pas nul, branche sur vecteur "SYNTAX_ERROR" (sert à exclure coexistance de ",J" et ",A" dans le même LOAD)
E03D-	20 98 D3	JSR D398	si nul, XCRGET incrémente TXTPTR, lit un caractère (CHRGET), les espaces sont sautés, le met dans A, le convertit en MAJUSCULE, Z = 1 si fin d'instruction (0 ou :), C = 0 si chiffre 0 à 9 (soit #30 à #39), sinon C =1. Y et X inchangés
E040-	20 FA D2	JSR D2FA	E853/ROM évalue un nombre non signé à TXTPTR et revient avec ce nombre dans YA, 33/34 et D3/D4 et mise à jour de TXTPTR sur l'octet suivant ce nombre (C = 0, N et Z positionnés selon l'octet de poids faible)
E043-	8C 52 C0	STY C052	sauve ce nombre en C052/C053 (DESALO)
E046-	8D 53 C0	STA C053	(c'est l'adresse où il faudra charger le fichier)
E049-	A2 40	LDX #40	X = #40 (pour paramètre ",A")
E04B-	8E 4E C0	STX C04E	VSALO1 nul, écrit X dans VSALO1 qui vaut donc à la fin soit #00 (ni ",A" ni ",J"), soit #40 (",A") ou #80 (",J")
E04E-	30 B9	BMI E009	si X = #80 branche en E009 (caractère suivant)
E050-	10 B0	BPL E002	si X = #40 branche en E002 (caractère courant, car la routine D2FA a déjà effectué une mise à jour de TXTPTR sur l'octet suivant)

Fin de commande rencontrée, charge le fichier

E052-	20 E5 E0	JSR E0E5	XLOADA charge fichier (ou fichiers "joints") selon BUFNOM, VSALO0 (",V" ou ",N"), VSALO1 (",A" ou ",J") et DESALO (si ",A"). Affiche le STATUS du fichier ou des fichiers "joints" (si ",V").
E055-	2C 4D C0	BIT C04D	teste b6 de VSALO0 (à 1 si ",V")
E058-	50 2B	BVC E085	branche en E085 si ce n'est pas le cas

Traitement supplémentaire si paramètre ",V"
Mise à jour des variables système FT, ST, EX et ED

E05A-	AD 51 C0	LDA C051	A = FTYPE type du fichier (principal) chargé
E05D-	20 E1 D7	JSR D7E1	mise à jour de la variable FT (HH=#00 et LL=A)
E060-	AD 52 C0	LDA C052	AY = DESALO (adresse de début de fichier)
E063-	AC 53 C0	LDY C053	mise à jour de la variable ST (HH=Y et LL=A)
E066-	20 F8 D7	JSR D7F8	(inclus effet éventuel de ",A" ou ",J")
E069-	AD 56 C0	LDA C056	AY = EXSALO adresse exécution fichier (principal)
E06C-	AC 57 C0	LDY C057	mise à jour de la variable EX (HH=Y et LL=A)
E06F-	20 FE D7	JSR D7FE	(ne semble pas affectée par ",A" ou ",J")
E072-	18	CLC	prépare addition (calcule nouvelle fin si",A" ou ",J")
E073-	AD 52 C0	LDA C052	A = LL de DESALO adresse (nouveau) début de fichier

E076-	6D 4F C0	ADC C04F	ajoute LL de LGSALO longueur du fichier
E079-	48	PHA	et empile le résultat: LL de (nouvelle) fin
E07A-	AD 53 C0	LDA C053	A = HH de DESALO adresse (nouveau) début de fichier
E07D-	6D 50 C0	ADC C050	ajoute HH de LGSALO longueur du fichier
E080-	A8	TAY	sauve HH de (nouvelle) fin dans Y
E081-	68	PLA	récupère AY = adresse (nouvelle) fin du fichier
E082-	20 FB D7	JSR D7FB	mise à jour de la variable ED (HH=Y et LL=A)

Suite pour tous

E085-	AD 4D C0	LDA C04D	A = VSALO0 dont b6 = flag ",V" et b7 = flag ",N"
E088-	0A	ASL	ancien b7 -> C et ancien b6 -> b7 et N
E089-	30 50	BMI E0DB	si N = 1 (pour ",V"), branche (simple CLI et RTS, c'est fini)
E08B-	2A	ROL	si N = 0, C -> b0 (ancien flag ",N")
E08C-	49 01	EOR #01	inverse b0 (ancien b7 = flag "Non exécution") on a donc maintenant 0000 0001 si rien demandé et 0000 0000 si STOP demandé
E08E-	2D 51 C0	AND C051	FTYPE dont le b0 sert de flag "AUTO" si à 1. On a donc maintenant 0000 0001 si AUTO **ET** si STOP pas expressément demandé par ",N"
E091-	4A	LSR	b0 -> C (à 1 l'exécution du fichier sera lancée)
E092-	AD 51 C0	LDA C051	A = FTYPE dont le b7 sert de flag "BASIC"
E095-	10 0D	BPL E0A4	branche si pas "BASIC"
E097-	08	PHP	si "BASIC", sauvegarde les indicateurs 6502
E098-	20 B4 E0	JSR E0B4	restaure les liens de lignes et les pointeurs
E09B-	28	PLP	récupère les indicateurs
E09C-	90 03	BCC E0A1	saute la ligne suivante si C = 0 (pas de RUN)

Termine le chargement du programme BASIC en lançant l'exécution

E09E-	4C AC D1	JMP D1AC	JSR C73A/ROM (place TXTPTR au début du BASIC et RUN)

Termine le chargement du programme BASIC sans lancer l'exécution

E0A1-	4C 80 D1	JMP D180	JSR C4A8/ROM (retour simple au Ready)

Termine le chargement du fichier non BASIC sans lancer l'exécution

E0A4-	90 35	BCC E0DB	autorise les interruptions et simple RTS si C = 0

Termine le chargement du programme langage machine en lançant l'exécution

E0A6-	AD 56 C0	LDA C056	si C = 1 prend EXSALO (adresse exécution fichier)
E0A9-	AC 57 C0	LDY C057	dans AY et exécute
E0AC-	4C 6B 04	JMP 046B	

EXÉCUTION DE LA COMMANDE SEDORIC OLD

Rappel de la syntaxe

OLD tout court

Récupère un programme BASIC après un NEW, un BOOT ou un RESET. Cette récupération ne se passe bien que si rien n'a été fait fait entre le NEW, le BOOT ou le RESET.

Le pointeur 9A/9B "début BASIC" pointe normalement sur 0501 qui contient le premier lien de ligne (adresse début ligne suivante, LL en 0501 et HH en 0502). Lors d'un NEW ou RESET ce HH est mis à zéro. OLD retire ce zéro, qui seul est significatif, puis restaure les liens de lignes et les pointeurs.

Non documenté

Et la vérification de syntaxe? Il n'y en a pas. Si vous tapez OLD,LIST au lieu de OLD:LIST, la commande OLD sera exécutée et c'est seulement en passant à la suite que le système s'aperçoit qu'il y a un problème!

E0AF-	A0 01	LDY #01	Y = #01 pour indexer adresse suivant 0501 soit 0502
E0B1-	98	TYA	A = #01 doit seulement être <> #00
E0B2-	91 9A	STA (9A),Y	écrit #01 en 0502
E0B4-	08	PHP	sauvegarde les indicateurs 6502
E0B5-	78	SEI	interdit les interruptions
E0B6-	20 88 D1	JSR D188	JSR C563/ROM restaure les liens des lignes
E0B9-	A4 92	LDY 92	
E0BB-	A5 91	LDA 91	prend dans AY l'adresse de fin de BASIC -2
E0BD-	18	CLC	prépare une addition
E0BE-	69 02	ADC #02	ajoute 2 à LL
E0C0-	90 01	BCC E0C3	répercute éventuellement
E0C2-	C8	INY	la retenue sur HH
E0C3-	85 9C	STA 9C	
E0C5-	84 9D	STY 9D	met à jour pointeur "fin BASIC/début variables"
E0C7-	85 9E	STA 9E	
E0C9-	84 9F	STY 9F	met à jour pointeur "fin variables/début tableaux"
E0CB-	85 A0	STA A0	
E0CD-	84 A1	STY A1	met à jour pointeur "fin tableaux/début free RAM"
E0CF-	A5 A6	LDA A6	
E0D1-	A4 A7	LDY A7	met à jour HIMEM "fin chaînes/début zone LM"
E0D3-	85 A2	STA A2	
E0D5-	84 A3	STY A3	met à jour pointeur "fin free RAM/début chaînes"
E0D7-	28	PLP	récupère les indicateurs
E0D8-	4C CC D1	JMP D1CC	JSR C952/ROM commande "Restore" = mise à jour du pointeur DATA (B0/B1) sur le début du texte BASIC

Fin utilisée par la commande LOAD et la routine XLOADA

E0DB-	58	CLI	autorise les interruptions
E0DC-	60	RTS	

Gestion des erreurs pour XLOADA, LOAD, REN, DEL, DESTROY, DELBAK etc.

E0DD-	A2 00	LDX #00	pour "FILE_NOT_FOUND_ERROR"

E0DF-	2C A2 0C	BIT 0CA2	continue en D67E pour traiter l'erreur
E0E0-	A2 0C	LDX #0C	pour "FILE_TYPE_MISMATCH_ERROR"
E0E2-	4C 7E D6	JMP D67E	continue en D67E pour traiter l'erreur

XLOADA charge programme selon BUFNOM, VSALO0, VSALO1, DESALO

E0E5-	20 2D DB	JSR DB2D	vérifie que la disquette en place est bien une disquette SEDORIC, cherche le fichier dans le catalogue, revient avec le bon secteur de catalogue en place (de coordonnées POSNMP et POSNMS) et avec X = POSNMX, pointant sur "l'entrée" cherchée ou avec Z = 1 si le fichier n'a pas été trouvé
E0E8-	F0 F3	BEQ E0DD	sinon trouvé, "FILE_NOT_FOUND_ERROR"

Charge fichier selon X = POSNMX, VSALO0, VSALO1, DESALO

X = POSNMX pointe sur la première lettre du nom_de_fichier à charger ("entrée")

Chaque "entrée" de catalogue est structurée ainsi:

octets n°00 à 08	nom
octets n°09 à 0B	extension
octet n°0C	piste du descripteur
octet n°0D	secteur du descripteur
octet n°0E	nombre de secteurs du fichier **(y compris les descripteurs)**
octet n°0F	attribut de protection (b6=1, PROT si b7=1, UNPROT si b7=0)

E0EA-	78	SEI	interdit les interruptions
E0EB-	38	SEC	C = 1 pour flag AUTO
E0EC-	6E 72 C0	ROR C072	met à 1 le b7 de C072 (flag AUTO par défaut)
E0EF-	BD 0C C3	LDA C30C,X	lit A = piste du descripteur
E0F2-	BC 0D C3	LDY C30D,X	et Y = secteur du descripteur
E0F5-	20 5D DA	JSR DA5D	XPBUF1 charge dans BUF1 le secteur du descripteur

Le premier secteur de descripteur chargé dans BUF1 est structuré ainsi:

C100-	octets n°00/01	"lien" (coordonnées du descripteur suivant)
C102-	octet n°02 contient #FF	(seulement si premier secteur descripteur) X pointe sur ce #FF, la suite est exprimée par rapport à X
C103-	octet n°03 (C101+X)	type de fichier (voir manuel SEDORIC page 100)
C104-	octets n°04/05 (C102+X et C103+X)	adresse (normale) de début
C106-	octets n°06/07 (C104+X et C105+X)	adresse (normale) de fin
C108-	octets n°08/09 (C106+X et C107+X)	adresse d'exécution si AUTO
C10A-	octets n°0A/0B (C108+X et C109+X)	nombre de secteurs à charger
C10C-	octets n°0C/0D (C100+Y et C101+Y)	liste coordonnées piste/secteur des secteurs à charger. Lorsque Y atteint #00 (fin BUF1), il faut charger

Le descripteur suivant dont la structure est simplifiée:

C100-	octets n°00/0	"lien" (coordonnées du descripteur suivant)
C102-	octets n°02/03 (C100+Y et C101+Y)	liste des coordonnées piste/secteur des secteurs à charger (Y

de #02 à #EF au maximum).

Pour les fichiers "joints", le "lien" du dernier descripteur de chaque fichier indique les coordonnées du premier descripteur du fichier suivant (le "lien" du dernier descripteur du dernier fichier indique bien sûr #0000). Il semble possible de combiner les descripteurs pour gagner de la place. Dans ce cas un #FF sera placé après la dernière paire de coordonnées piste/secteur du dernier secteur à charger à la fin du dernier descripteur de chaque fichier et sera suivi des octets usuels: type de fichier, adresse (normale) de début, adresse (normale) de fin, adresse d'exécution, nombre de secteurs à charger, liste des coordonnées piste/secteur des secteurs à charger du fichier "joint". La présence du #FF valide la valeur de X pour la lecture des octets de STATUS, puis la valeur de Y pour la lecture des octets de coordonnées piste/secteur des secteurs à charger.

La valeur X = #02 en E0F8 n'est donc valable qu'au premier tour. E0FA représente un point de rebouclage après chargement du dernier secteur du fichier, afin d'explorer le reste du secteur descripteur à la recherche d'un éventuel #FF marquant le début d'un fichier "joint" (voir plus haut). Si ce #FF n'est pas trouvé, les coordonnées d'un éventuel premier descripteur suivant seront lues en C100/C101, ce descripteur chargé dans BUF1 et exploré à la recherche d'un éventuel #FF. S'il n'y a plus de #FF, ni de descripteur, c'est fini: CLI et RTS.

E0F8-	A2 02	LDX #02	index pour octet n°02 du secteur descripteur
E0FA-	BD 00 C1	LDA C100,X	charge octet visé par X dans secteur descripteur
E0FD-	C9 FF	CMP #FF	est-ce #FF?
E0FF-	F0 0D	BEQ E10E	si oui, continue en E10E avec C = 1 et X validé
E101-	E8	INX	sinon, vise l'octet suivant
E102-	D0 F6	BNE E0FA	branche en E0FA jusqu'à trouver #FF ou que X revienne à #00 (fin de descripteur)
E104-	AD 00 C1	LDA C100	charge alors dans AY les coordonnées d'un
E107-	AC 01 C1	LDY C101	éventuel descripteur suivant (A = piste Y = secteur)
E10A-	F0 CF	BEQ E0DB	si Y nul, il n'y en a pas, effectue CLI et RTS
E10C-	D0 E7	BNE E0F5	s'il y a une adresse, reboucle en E0F5 (charge secteur descripteur)

#FF trouvé

E10E-	BD 01 C1	LDA C101,X	vise premier octet après X (X = position de #FF)
E111-	85 F9	STA F9	lit le type de fichier et le sauve dans F9
E113-	29 C0	AND #C0	1100 0000 force tous les bits à 0 sauf b6 et b7
E115-	D0 05	BNE E11C	branche si BASIC ou BLOC_DE_DONNEES ou WINDOW
E117-	2C 4D C0	BIT C04D	sinon, type DIRECT ou SEQUENTIEL, interdit avec LOAD, sauf si paramètre ",V". Teste le b6 de C04D (flag pour ",V")
E11A-	50 C4	BVC E0E0	si pas ",V" branche en E0E0, "FILE_TYPE_MISMATCH_ERROR"
E11C-	2C 4E C0	BIT C04E	si autorisé, teste b6 et b7 de C04E (",A" ",J")
E11F-	70 19	BVS E13A	branche en E13A si ",A" (adresse de chargement)
E121-	10 0B	BPL E12E	branche en E12E si ni ",A" ni ",J"

Entrée secondaire ",J" (ajoute à fin BASIC)
NB: C a été mis à 1 en E0FD/E0FF donc correct pour SBC

E123-	A4 9D	LDY 9D	lit Y = HH et

```
E125-  A5 9C     LDA 9C      A = LL du pointeur "fin du programme BASIC/début des variables"
E127-  E9 02     SBC #02     calcule adresse "fin du programme BASIC" en diminuant LL du
E129-  B0 09     BCS E134    pointeur de 2. Fini si C = 1 (pas de retenue),
E12B-  88        DEY         mais répercussion sur HH si C = 0 (DEY ne touche
E12C-  90 06     BCC E134    pas C) et branchement final forcé en E134
```

Entrée secondaire ni ",J" ni ",A" (charge à la place normale)

```
E12E-  BD 02 C1  LDA C102,X  vise les deuxième  et troisième octets après X et prend dans AY
E131-  BC 03 C1  LDY C103,X  l'adresse (normale) de début de fichier
E134-  8D 52 C0  STA C052    AY = adresse où sera chargé le fichier
E137-  8C 53 C0  STY C053    et l'écrit dans DESALO (nouveau) début
```

Entrée secondaire ",A adresse_de_chargement", simple suite pour les autres

DESALO doit contenir en entrée l'adresse_de_chargement: celle qui suivait le ",A" si on vient de l'entrée de XLOADA ou celle de fin du BASIC en place si ",J" (mise à jour par le sous-programme E123) ou la valeur normale de début si ni ",A" ni ",J" (mise à jour par le sous-programme E12E).

```
E13A-  38        SEC         prépare une soustraction
E13B-  BD 04 C1  LDA C104,X  vise les deuxième, troisième, quatrième et cinquième octets après X
E13E-  FD 02 C1  SBC C102,X  calcule la longueur du fichier à charger
E141-  8D 4F C0  STA C04F    (adresse de fin - adresse de début)
E144-  BD 05 C1  LDA C105,X
E147-  FD 03 C1  SBC C103,X  et l'écrit dans LGSALO (C04F/C050)
E14A-  8D 50 C0  STA C050
E14D-  18        CLC         prépare une addition
E14E-  AD 52 C0  LDA C052    met LL de DESALO (adresse_de_chargement)
E151-  8D 03 C0  STA C003    dans LL de RWBUF (adresse écriture du prochain secteur)
E154-  6D 4F C0  ADC C04F    calcule A = LL de DESALO + LL de LGSALO
E157-  48        PHA         et empile LL de (nouvelle) fin
E158-  AD 53 C0  LDA C053    prend HH de DESALO (adresse de chargement),
E15B-  A8        TAY         le décrémente et place le résultat dans HH de
E15C-  88        DEY         RWBUF (donc RWBUF = adresse de chargement du fichier - #100)
E15D-  8C 04 C0  STY C004    (sera incrémenté de #100 en début de boucle)
E160-  6D 50 C0  ADC C050    A = HH DESALO + HH LGSALO = HH (nouvelle) fin
E163-  A8        TAY         le copie dans Y. L'adresse de (nouvelle) fin, formée de LL/pile et HH dans
                             Y servira pour ",V"
```

AUTO/STOP

```
E164-  2C 72 C0  BIT C072    teste le b7 de C072 (AUTO si b7=1 STOP si b7=0)
E167-  10 0C     BPL E175    si STOP branche en E175 (EXSALO pas mis à jour)
E169-  BD 06 C1  LDA C106,X  si AUTO, vise les sixième et septième octets après X
E16C-  8D 56 C0  STA C056    lit adresse d'exécution
E16F-  BD 07 C1  LDA C107,X  dans secteur descripteur
E172-  8D 57 C0  STA C057    et l'écrit dans EXSALO
```

F7/F8 nombre de secteurs à charger

E175-	BD 08 C1	LDA C108,X	vise les huitième et neuvième octets après X
E178-	85 F7	STA F7	lit dans secteur descripteur
E17A-	BD 09 C1	LDA C109,X	le nombre de secteurs à charger
E17D-	85 F8	STA F8	et l'écrit en F7/F8

",V" visualisation du STATUS du fichier

Très intéressant, on constate que lorsque ",V" est couplé à ",J" ou à ",A" (un seul possible à la fois) on obtient "en simulation", ce que seront les adresses de début et de fin du fichier à sa nouvelle place!

E17F-	2C 4D C0	BIT C04D	teste b6 de VSALO0 (à 1 si ",V")
E182-	50 35	BVC E1B9	branche en E1B9 si pas ",V"
E184-	AD 53 C0	LDA C053	si ",V" lit HH de DESALO
E187-	20 13 D6	JSR D613	XAFHEX affiche en hexadécimal le contenu de A
E18A-	AD 52 C0	LDA C052	lit LL de DESALO
E18D-	20 13 D6	JSR D613	XAFHEX affiche en hexadécimal le contenu de A
E190-	20 28 D6	JSR D628	affiche un espace
E193-	98	TYA	prend dans A le HH de fin de fichier
E194-	20 13 D6	JSR D613	XAFHEX affiche en hexadécimal le contenu de A
E197-	68	PLA	prend dans A le LL de fin de fichier
E198-	20 13 D6	JSR D613	XAFHEX affiche en hexadécimal le contenu de A
E19B-	20 28 D6	JSR D628	affiche un espace
E19E-	A5 F9	LDA F9	prend dans A le type de fichier
E1A0-	20 13 D6	JSR D613	XAFHEX affiche en hexadécimal le contenu de A
E1A3-	20 28 D6	JSR D628	affiche un espace
E1A6-	AD 57 C0	LDA C057	prend dans A le HH de l'adresse d'exécution
E1A9-	20 13 D6	JSR D613	XAFHEX affiche en hexadécimal le contenu de A
E1AC-	AD 56 C0	LDA C056	prend dans A le LL de l'adresse d'exécution
E1AF-	20 13 D6	JSR D613	XAFHEX affiche en hexadécimal le contenu de A
E1B2-	20 28 D6	JSR D628	affiche un espace
E1B5-	20 06 D2	JSR D206	CBF0/ROM va à la ligne
E1B8-	24 68	BIT 68	continue en E1BA (",V" continue, car il y a peut-être des fichiers "joints")

Suite si pas ",V"

E1B9-	68	PLA	élimine octet de la pile: LL de (nouvelle) fin (qui n'était là que pour ",V" le HH dans Y sera écrasé plus loin)

Suite pour tous

E1BA-	8A	TXA	A = position de #FF (début octets de STATUS)
E1BB-	18	CLC	prépare une addition
E1BC-	69 06	ADC #06	A = A + #06 (vise huitième octet après #FF)
E1BE-	A8	TAY	met résultat dans Y (pointeur lecture piste/secteur)

Actuellement, les pointeurs Y et RWBUF visent trop court et seront mis à jour à l'entrée de la boucle: Y

qui pointera sur chaque paire de coordonnées piste/secteur du secteur à charger, vise actuellement le huitième octet après #FF, puis visera le dixième en E1BF et finalement le douzième (première paire de coordonnées) au début du sous-programme E228; RWBUF (adresse de chargement du fichier) vise #100 octets trop bas et sera mis à jour en E1CA.

E1BF-	20 28 E2	JSR E228	sous-programme Y = Y + #02 avec le descripteur en place
E1C2-	A5 F7	LDA F7	début de boucle:
E1C4-	D0 02	BNE E1C8	décrémente le nombre de secteurs restant à
E1C6-	C6 F8	DEC F8	charger (anticipation du chargement en E1D6)
E1C8-	C6 F7	DEC F7	
E1CA-	EE 04 C0	INC C004	augmente RWBUF de #100 (prochaine adresse de chargement)
E1CD-	A5 F7	LDA F7	
E1CF-	05 F8	ORA F8	teste s'il reste des secteurs à charger
E1D1-	F0 09	BEQ E1DC	si plus rien à charger, branche en E1DC. En fait, il reste un secteur à charger (le dernier, qui est spécial) car le nombre de secteurs à charger (F7/F8) a été décrémenté à l'avance.
E1D3-	20 28 E2	JSR E228	Y = Y + 2 (chargement éventuel du descripteur suivant si nécessaire). Au premier tour, on a donc Y = #0C et là on charge!
E1D6-	20 50 E2	JSR E250	lit coordonnées du prochain secteur à charger et le charge selon DRIVE PISTE SECTEUR et RWBUF (pas de chargement si ",V")
E1D9-	4C C2 E1	JMP E1C2	et reboucle

Chargement du dernier secteur du fichier en cours
Ce secteur ne doit être mis en place que partiellement, selon LL de LGSALO

E1DC-	AD 03 C0	LDA C003	sauve adresse de chargement présente dans
E1DF-	AE 04 C0	LDX C004	RWBUF en F5/F6 pour usage ultérieur
E1E2-	85 F5	STA F5	(mise en place finale des seuls octets
E1E4-	86 F6	STX F6	significatifs du dernier secteur)
E1E6-	20 28 E2	JSR E228	Y = Y + #02 (chargement éventuel du descripteur suivant)
E1E9-	98	TYA	sauve Y (index des coordonnées du dernier secteur) sur la pile
E1EA-	48	PHA	(pour exploration ultérieure de la fin du secteur descripteur)
E1EB-	A9 00	LDA #00	attention au BIT qui suit, car A = 0000 0000
E1ED-	A2 C2	LDX #C2	met l'adresse de BUF2 dans RWBUF pour
E1EF-	8D 03 C0	STA C003	chargement transitoire du dernier secteur
E1F2-	8E 04 C0	STX C004	
E1F5-	2C 4D C0	BIT C04D	teste b6 et b7 de VSALO0, met Z à 1 car A = #00
E1F8-	70 0E	BVS E208	branche si b6 = 1 (",V" pas besoin de charger)
E1FA-	20 50 E2	JSR E250	charge dans BUF2 le dernier secteur à charger
E1FD-	A0 FF	LDY #FF	prépare index pour #00 en début de boucle
E1FF-	C8	INY	vise octet à lire
E200-	B9 00 C2	LDA C200,Y	lit octet du dernier secteur dans BUF2
E203-	91 F5	STA (F5),Y	et l'écrit à partir de l'adresse en F5/F6
E205-	CC 4F C0	CPY C04F	compare à LL de LGSALO (= longueur du fichier) qui représente en fait la longueur du dernier morceau (= secteur partiel)
E208-	D0 F5	BNE E1FF	reboucle tant que le dernier octet significatif du fichier n'a pas été mis en place finale (bogue évitée de justesse pour ",V" car Z = 1 par chance, il aurait été mieux en E1F8 de brancher en E20A)

E20A-	68	PLA	récupère Y (index de lecture des coordonnées
E20B-	A8	TAY	piste/secteur dans BUF1)
E20C-	20 28 E2	JSR E228	Y = Y + 2 teste si la fin du descripteur a été atteinte, revient avec C = 1 si fin de descripteur et pas de descripteur suivant
E20F-	B0 3D	BCS E24E	si pas de descripteur suivant, branche en E24E
E211-	98	TYA	passe Y dans X qui devient index de recherche
E212-	AA	TAX	de #FF (début du descripteur suivant)
E213-	AD 72 C0	LDA C072	flag AUTO si b7=1, STOP si b7=0
E216-	10 0D	BPL E225	si STOP, branche en E225 (signifie qu'on a déjà chargé le premier fichier d'un ensemble de fichiers "joints". L'adresse d'exécution des fichiers secondaires n'a pas besoin d'être initialisée.
E218-	4E 72 C0	LSR C072	si AUTO, remet b7 à 0 (STOP) les fichiers
E21B-	A9 00	LDA #00	secondaires seront d'office ni ",J" ni ",A":
E21D-	8D 4E C0	STA C04E	remet VSALO1 à 0 (ni ",J" ni ",A")
E220-	A5 F9	LDA F9	seul le FTYPE du premier fichier comptera pour le
E222-	8D 51 C0	STA C051	RUN: remet le type de fichier dans FTYPE
E225-	4C FA E0	JMP E0FA	et continue en E0FA (cherche octet #FF)

Sous-programme Y = Y + 2 avec chargement éventuel descripteur suivant

Ajuste Y pour viser les coordonnées du secteur suivant à charger, si Y n'a pas dépassé la fin du descripteur, retourne avec C = 0, sinon charge le descripteur suivant, ajuste Y et retourne avec C = 0. S'il n'y a plus de descripteur, retourne avec C = 1

E228-	C8	INY	Y = Y + 2
E229-	C8	INY	Y vise les coordonnées du prochain secteur à charger
E22A-	D0 1D	BNE E249	si Y valable branche en E249
E22C-	AD 03 C0	LDA C003	si Y = 0, (Y dépasse la fin du descripteur)
E22F-	48	PHA	lit adresse en RWBUF (C003/C004) et l'empile
E230-	AD 04 C0	LDA C004	(adresse où charger le secteur suivant du fichier)
E233-	48	PHA	
E234-	AD 00 C1	LDA C100	A = piste
E237-	AC 01 C1	LDY C101	Y = secteur du secteur descripteur suivant
E23A-	F0 0F	BEQ E24B	si Y = 0 (impossible sauf si pas de suivant)
E23C-	20 5D DA	JSR DA5D	XPBUF1 charge dans BUF1 le secteur descripteur suivant
E23F-	68	PLA	
E240-	8D 04 C0	STA C004	
E243-	68	PLA	récupère valeur initiale de l'adresse RWBUF
E244-	8D 03 C0	STA C003	
E247-	A0 02	LDY #02	Y = 2 (vise première paire de coordonnées du secteur suivant)
E249-	18	CLC	et C = 0 (descripteur en place et Y pointant
E24A-	60	RTS	sur coordonnées du secteur suivant à charger)

"Il n'y a pas de descripteur suivant"

E24B-	38	SEC	C = 1 (pas de descripteur suivant)
E24C-	68	PLA	élimine 2 octets de la pile
E24D-	68	PLA	(adresse qui était en RWBUF, devenue inutile)

```
E24E- 58       CLI       autorise les interruptions
E24F- 60       RTS       NB: Y = 0 dans ce cas
```

Lit les coordonnées du prochain secteur à charger et le charge selon DRIVE PISTE SECTEUR et RWBUF (sauf si ",V")

```
E250- B9 00 C1 LDA C100,Y lit dans le descripteur les coordonnées
E253- 8D 01 C0 STA C001   piste/secteur du prochain secteur à charger
E256- B9 01 C1 LDA C101,Y et les écrit respectivement dans PISTE
E259- 8D 02 C0 STA C002   et dans SECTEUR
E25C- 2C 4D C0 BIT C04D   teste b6 de VSALO (à 1 si ",V")
E25F- 70 E9    BVS E24A   si ",V", branche en E24A (simple RTS)
E261- 4C 73 DA JMP DA73   XPRSEC lit un secteur selon DRIVE, PISTE, SECTEUR et RWBUF Le
                          retour se fera au point d'appel du sous-programme E250 (économie d'un
                          RTS!)
```

Efface un fichier du catalogue indexé par POSNMX et présent dans BUF3 et libère tous ses secteurs

A partir du descripteur principal, les sous-programmes E264 et E266 libèrent sur la bitmap tous les secteurs du fichier (secteurs constituant le fichier et les éventuels fichiers "joints"), mais aussi tous les secteurs descripteurs du fichier et des éventuels fichiers "joints". Mis à part le secteur de bitmap qui est modifié, ni les descripteurs, ni les secteurs du ou des fichiers ne sont affectés: seule “l’entrée” correspondant au fichier à supprimer dans le catalogue est effacée et par conséquent, les coordonnées piste/secteur du descripteur principal sont perdues. Avec un bon éditeur de disquette il est possible de le retrouver et de rétablir la situation après un DEL malheureux! Il serait assez facile d'écrire un programme UNDEL ou un programme UNMERGE.

Ce sous-programme ne modifie qu'un seul secteur de catalogue, celui où était “l’entrée” supprimée. Si “l’entrée” à effacer est la dernière "entrée" valide du secteur de catalogue, elle est simplement surchargée de #00. Sinon, elle est écrasée par les suivantes qui sont remontées d'un cran (16 octets), la dernière "entrée", devenue inutile, est alors surchargée de #00. Cette "entrée" libérée est souvent réutilisée peu après (par exemple lors d'un SAVEO). Dans d'autres cas (par exemple lors d'un DESTROY), l'ensemble des secteurs de catalogue sera restructuré afin d'éliminer les "entrées" vides (qui sont toujours situées à la fin des secteurs de catalogue). Cette restructuration est effectuée par un autre sous-programme (E4A7) et permet de réduire le nombre de secteurs de catalogue utilisés et, éventuellement, de récupérer des secteurs.

Voir en ANNEXE le détail de ce qui se passe lors d’un DEL

A l'entrée C = 0 si nom_de_fichier_non_ambigu et C = 1 si nom_de_fichier_ambigu (dans ce cas le nom de fichier avec jockers ne sera pas affiché devant le message "IS PROTECTED" si erreur).

En sortie, C = 0 si tout s'est bien passé et C = 1 si erreur (laquelle est traitée de manière spécifique).

Entrée appelée de SAVEO, SAVEU et DEL nom_de_fichier_non_ambigu

```
E264- 18       CLC       entrée avec C = 0 (flag nom_de_fichier_non_ambigu)
E265- 24 38    BIT 38    continue en E267
```

XNOMDE supprime nom_de_fichier_ambigu

E266-	38	SEC	entrée avec C = 1 (flag nom_de_fichier_ambigu) continue en E267
E267-	AE 27 C0	LDX C027	X = POSNMX position de “l’entrée” dans BUF3
E26A-	BC 0F C3	LDY C30F,X	teste b7 de l'octet PROT/UNPROT
E26D-	30 61	BMI E2D0	si b7 = 1, branche en E2D0 qui affiche [le nom du fichier si SAVEO] IS PROTECTED et retourne à SAVEO ou DEL avec C = 1
E26F-	AD 04 C2	LDA C204	
E272-	D0 03	BNE E277	décrémente le nombre de fichier
E274-	CE 05 C2	DEC C205	dans le secteur de bitmap
E277-	CE 04 C2	DEC C204	
E27A-	BD 0C C3	LDA C30C,X	lit les coordonnées du descripteur principal
E27D-	48	PHA	(treizième et quatorzième octets de “l’entrée”
E27E-	BD 0D C3	LDA C30D,X	dans BUF3) et les empile
E281-	48	PHA	

Efface “l’entrée” dans le secteur de catalogue

E282-	38	SEC	
E283-	AD 02 C3	LDA C302	calcule la position de la prochaine entrée
E286-	E9 10	SBC #10	libre (n° du premier octet vacant dans secteur de
E288-	8D 02 C3	STA C302	catalogue) et la copie dans Y
E28B-	A8	TAY	
E28C-	A9 10	LDA #10	
E28E-	85 F2	STA F2	F2 = 16 octets à déplacer ou à effacer
E290-	B9 00 C3	LDA C300,Y	lit octet de la dernière entrée actuelle
E293-	86 F3	STX F3	F3 = POSNMX = n° premier octet de l'entrée à supprimer
E295-	C4 F3	CPY F3	l'entrée à supprimer est-elle la dernière entrée?
E297-	F0 03	BEQ E29C	si oui, saute l'instruction de copie
E299-	9D 00 C3	STA C300,X	sinon, remplace l'octet de la dernière entrée
E29C-	A9 00	LDA #00	actuelle par #00 et copie l'ancienne valeur
E29E-	99 00 C3	STA C300,Y	à la place correspondante dans l'entrée à supprimer
E2A1-	E8	INX	pour copie de l'octet suivant
E2A2-	C8	INY	pour lecture et effacement de l'octet suivant
E2A3-	C6 F2	DEC F2	décrémente le nombre d'octets à déplacer ou à effacer
E2A5-	D0 E9	BNE E290	et reboucle tant qu'il en reste
E2A7-	68	PLA	
E2A8-	A8	TAY	récupère dans AY les coordonnées du descripteur principal
E2A9-	68	PLA	
E2AA-	20 5D DA	JSR DA5D	XPBUF1 charge ce descripteur dans BUF1
E2AD-	AD 01 C0	LDA C001	
E2B0-	AC 02 C0	LDY C002	lit dans AY les coordonnées du dernier secteur chargé
E2B3-	20 15 DD	JSR DD15	XDETSE libère ce secteur sur la bitmap courant
E2B6-	A2 02	LDX #02	
E2B8-	BD 00 C1	LDA C100,X	lit l'octet visé par X et teste si #FF
E2BB-	C9 FF	CMP #FF	
E2BD-	F0 1D	BEQ E2DC	si oui, branche avec X positionné sur cet octet

```
E2BF-  E8          INX          sinon, indexe l'octet suivant
E2C0-  D0 F6       BNE E2B8     et relit tout le secteur jusqu'au dernier octet à la recherche d'un éventuel #FF
E2C2-  AD 00 C1    LDA C100     si rien trouvé, lit dans AY les coordonnées
E2C5-  AC 01 C1    LDY C101     du descripteur suivant
E2C8-  D0 E0       BNE E2AA     s'il y a un descripteur suivant, reboucle en E2AA
```

Sauve les secteurs de bitmap et de catalogue

```
E2CA-  20 8A DA    JSR DA8A     si pas de suivant, sauve le secteur de bitmap. L'entrée de cette routine
                                XSMAP sauve BUF2 (bitmap) sur la disquette a été déportée en DC80 pour
                                adaptation à la double bitmap. Un JSR DC80 plus direct aurait fait gagner
                                un peu de temps.
E2CD-  4C 82 DA    JMP DA82     sauve le secteur catalogue à POSNMP et POSNMS et retourne
```

Erreur: le fichier était PROTégé

```
E2D0-  B0 03       BCS E2D5     si nom_de_fichier_ambigu, saute l'instruction suivante
E2D2-  20 B4 DA    JSR DAB4     affiche le nom de fichier présent à POSNMX dans BUF3
E2D5-  A2 09       LDX #09      indexe le message "IS PROTECTED"
E2D7-  20 6C D3    JSR D36C     affiche (X+1) ème message de zone CEE7 terminé par "caractère + 128"
E2DA-  38          SEC          retourne avec C = 1 (erreur)
E2DB-  60          RTS
```

#FF trouvé, saute les octets de STATUS
X pointe sur le #FF identifiant le début d'un groupe d'octets STATUS

```
E2DC-  BD 08 C1    LDA C108,X
E2DF-  85 F5       STA F5       copie le nombre de secteurs de ce fichier
E2E1-  BD 09 C1    LDA C109,X   (nombre d'octets à libérer) dans F5/F6
E2E4-  85 F6       STA F6
E2E6-  8A          TXA          X = X + 10
E2E7-  18          CLC          (pour passer les octets de STATUS et viser
E2E8-  69 0A       ADC #0A      la première paire piste/secteur de coordonnées
E2EA-  AA          TAX          des secteurs constituant le fichier)
```

Libère les secteurs constituant le fichier

```
E2EB-  8A          TXA
E2EC-  48          PHA          empile X
E2ED-  BD 00 C1    LDA C100,X   lit dans AY les coordonnées d'un
E2F0-  BC 01 C1    LDY C101,X   secteur du fichier
E2F3-  20 15 DD    JSR DD15     XDETSE libère ce secteur sur la bitmap courant
E2F6-  68          PLA
E2F7-  AA          TAX          récupère X et l'augmente de 2
E2F8-  E8          INX          pour viser la paire suivante
E2F9-  E8          INX
E2FA-  D0 16       BNE E312     branche si fin du descripteur pas atteinte
```

Charge et libère un descripteur secondaire

E2FC-	AD 00 C1	LDA C100	si la fin du descripteur est atteinte, lit
E2FF-	AC 01 C1	LDY C101	dans AY les coordonnées du descripteur suivant
E302-	F0 C6	BEQ E2CA	s'il n'y en a plus, branche en E2CA
E304-	20 5D DA	JSR DA5D	XPBUF1 charge dans BUF1 ce descripteur
E307-	AD 01 C0	LDA C001	lit dans AY les coordonnées
E30A-	AC 02 C0	LDY C002	du secteur chargé
E30D-	20 15 DD	JSR DD15	XDETSE libère ce secteur sur la bitmap courant
E310-	A2 02	LDX #02	prépare X pour viser première paire piste/secteur de coordonnées des secteurs constituant le fichier

Pour chaque secteur libéré, décrémente le nombre restant

E312-	A4 F5	LDY F5	
E314-	D0 02	BNE E318	
E316-	C6 F6	DEC F6	décrémente le nombre de secteurs à libérer
E318-	C6 F5	DEC F5	
E31A-	A5 F5	LDA F5	
E31C-	05 F6	ORA F6	reste t-il des secteurs à libérer?
E31E-	D0 CB	BNE E2EB	si oui, branche en E2EB pour libérer le suivant
E320-	F0 96	BEQ E2B8	sinon, branche en E2B8 pour chercher #FF suivant

Affiche nom_de_fichier et taille du fichier à POSNMX

E322-	20 B4 DA	JSR DAB4	affiche le nom de fichier présent à POSNMX dans BUF3
E325-	A9 20	LDA #20	place un espace dans DEFAFF
E327-	8D 4C C0	STA C04C	pour l'afficher devant la taille du fichier
E32A-	AE 27 C0	LDX C027	X = POSNMX = début de “l’entrée” du catalogue
E32D-	BD 0F C3	LDA C30F,X	lit dernier octet d'"entrée" (HH taille + PROT)
E330-	08	PHP	sauvegarde les indicateurs dont N (à 1 si PROT)
E331-	29 0F	AND #0F	0000 1111 efface les 4 bits de poids fort (PROT)
E333-	A8	TAY	et sauve HH de la taille du fichier dans Y
E334-	BD 0E C3	LDA C30E,X	lit octet précédent (LL de taille du fichier)
E337-	20 56 D7	JSR D756	affichage en décimal du nombre AY
E33A-	A9 20	LDA #20	pour afficher un espace si pas PROT
E33C-	28	PLP	récupère les indicateurs dont N
E33D-	10 02	BPL E341	saute l'instruction suivante si pas PROTégé
E33F-	A9 50	LDA #50	pour afficher un "P"
E341-	4C 2A D6	JMP D62A	XAFCAR affiche le caractère ASCII contenu dans A

EXÉCUTION DE LA COMMANDE SEDORIC DIR

Rappel de la syntaxe

La syntaxe de la commande DIR est très simple: DIR nom_de_fichier_ambigu, le nom_de_fichier_ambigu pouvant non seulement comporter des jokers, mais aussi être omis ou réduit à la lettre qui désigne le lecteur

à traiter. On obtient alors un affichage du genre:

```
Drive A V3 (Mst) NNNNNNNNNNNNNNNNNNNN
MASTER   .SYS 105
1213 sectors free (D/42/17)  1 Files


Drive A V3 (Slv) NNNNNNNNNNNNNNNNNNNN
SLAVE    .SYS  17
1395 sectors free (D/42/17)  1 Files


Drive A (Type=G) NNNNNNNNNNNNNNNNNNNN
GAMES    .SYS  26
1377 sectors free (D/42/17)  1 Files
```

Dans ces 3 exemples le nom de la disquette est NNNNNNNNNNNNNNNNNNNN (DNAME, 21 caractères au maximum). Ces disquettes comportent un seul fichier qui contient la copie des secteurs systèmes d'une disquette MASTER, SLAVE ou GAMEINIT. Il y a respectivement 107, 16 et 25 secteurs occupés ou réservés pour le système, selon le type de disquette. Sur la disquette MASTER par exemple, il y a 42 pistes x 17 secteurs x 2 faces = 1428 secteurs formatés, moins 107 secteurs systèmes, moins le fichier (107 secteurs plus un secteur descripteur) soit 1428 - 107 - 108 = 1213 secteurs libres).

Dans ce qui suit, il est parfois difficile de se représenter ce que fait la routine DIR, notamment lors de l'affichage de la dernière ligne dont voici quelques exemples où les espaces ont été soulignés:

```
***0_sectors_free_(D/42/17)123_Files__
*123_sectors_free_(S/42/17)_14_Files__
1326_sectors_free_(D/42/17)__0_Files__
3731_sectors_free_(D/101/19)__0 Files_
```
(soit 38 caractères à chaque fois)

Notez en passant, que l'affichage des versions précédentes a été modifié (la virgule avant le nombre de files a été supprimée) ceci afin de permettre d'afficher plus de 99 pistes (extension BIGDISK). Comme le montre le quatrième exemple ci-dessus, une disquette Master formatée sous EUPHORIC en 101 pistes de 19 secteurs (valeurs maximales) offre 3731 secteurs à l'utilisateur. N'y rangez pas de petits fichiers de 2 ou 3 secteurs, car votre directory pourrait afficher plus d'un millier de fichiers, de quoi s'y perdre!

E344-	20 51 D4	JSR D451	XNFA lit un nom de fichier ambigu à TXTPTR et l'écrit dans BUFNOM
E347-	08	PHP	sauvegarde les indicateurs 6502
E348-	78	SEI	interdit les interruptions

Affichage de l'en-tête

E349-	A9 14	LDA #14	piste 20
E34B-	A0 01	LDY #01	secteur 1
E34D-	20 5D DA	JSR DA5D	XPBUF1 prend dans BUF1 le secteur Y de la piste A
E350-	20 4C DA	JSR DA4C	XPMAP prend le premier secteur de bitmap dans BUF2, vérifie le format, met à zéro le b7 de 2F (flag"première bitmap chargée").
E353-	A2 05	LDX #05	indexe "CRLFDrive_"
E355-	20 6C D3	JSR D36C	affiche (X+1) ème message de zone CEE7 terminé par "caractère + 128"

E358-	AD 28 C0	LDA C028	préfixe de BUFNOM (n° du drive demandé)
E35B-	20 0E D6	JSR D60E	convertit n° lecteur en lettre et l'affiche
E35E-	A2 06	LDX #06	pour septième message ("_V3_(Mst)_")
E360-	AC 0A C2	LDY C20A	Y = octet n°#0A de la bitmap (type de disquette)
E363-	F0 12	BEQ E377	si #00 (Master) continue en E377
E365-	A2 11	LDX #11	pour dix huitième message ("_V3_(Slv)_")
E367-	88	DEY	décrémente Y (type de disquette)
E368-	F0 0D	BEQ E377	si #00, y valait #01 (Slave) continue en E377
E36A-	A2 12	LDX #12	indexe "_(Type="
E36C-	20 6C D3	JSR D36C	affiche (X+1) ème message de zone CEE7 terminé par "caractère + 128"
E36F-	AD 0A C2	LDA C20A	A = octet n°#0A de la bitmap (type de disquette)
E372-	20 2A D6	JSR D62A	XAFCAR affiche le caractère ASCII contenu dans A
E375-	A2 13	LDX #13	indexe ")"
E377-	20 6C D3	JSR D36C	affiche (X+1) ème message de zone CEE7 terminé par "caractère + 128"
E37A-	A0 EB	LDY #EB	index pour afficher les 21 caractères de DNAME, le nom de la disquette. Y varie donc de #EB à #FF
E37C-	B9 1E C0	LDA C01E,Y	lu dans BUF1 à partir de C109 (nom disquette)
E37F-	20 2A D6	JSR D62A	XAFCAR affiche le caractère ASCII contenu dans A
E382-	C8	INY	caractère suivant
E383-	D0 F7	BNE E37C	reboucle en E37C tant que Y n'a pas dépassé #FF. En effet le passage de #FF à #00 entraîne Z = 1
E385-	20 1F E4	JSR E41F	JSR conditionnel à CBF0/ROM va à la ligne
E388-	20 06 D2	JSR D206	CBF0/ROM va à la ligne

Cherche un nom_de_fichier dans le catalogue

E38B-	20 30 DB	JSR DB30	cherche dans le catalogue à partir de POSNMX, le fichier indiqué dans BUFNOM et revient avec POSNMX, POSNMP, POSNMS ou Z = 1
E38E-	D0 09	BNE E399	si trouvé, continue en E399 (affichage nom, taille et éventuellement protection) (bogue bénigne: il aurait fallu un BNE E39B)
E390-	F0 33	BEQ E3C5	sinon, continue en E3C5 (affiche ligne finale)

Examine “l’entrée” suivante

E392-	78	SEI	interdit les interruptions
E393-	20 1F E4	JSR E41F	JSR conditionnel à CBF0/ROM (Aller à la ligne)
E396-	20 41 DB	JSR DB41	ajuste POSNMX sur entrée suivante du catalogue et reprend la recherche, dans le catalogue, du fichier indiqué dans BUFNOM (Z = 1 si fini)

Affiche le nom_du_fichier, sa taille et, P s'il est protégé puis cherche et affiche le fichier suivant

E399-	F0 27	BEQ E3C2	branche en E3C2 si fin de catalogue atteinte
E39B-	20 22 E3	JSR E322	affiche le nom_du_fichier à POSNMX, sa taille la lettre P, s'il est protégé. Puis ajuste POSNMX sur l'entrée suivante du
E39E-	20 41 DB	JSR DB41	catalogue et reprend la recherche dans le catalogue du fichier indiqué dans BUFNOM (avec au retour Z = 1 si fini)
E3A1-	F0 1C	BEQ E3BF	branche si la fin du catalogue est atteinte (CRLF pour finir la ligne commencée puis affichage ligne finale de bilan)

```
E3A3-  20 28 D6  JSR D628
E3A6-  20 28 D6  JSR D628      affiche deux espaces, puis le nom_du_fichier
E3A9-  20 22 E3  JSR E322      à POSNMX, sa taille et la lettre P s'il est protégé
```

Gestion pause de l'affichage

```
E3AC-  58        CLI           autorise les interruptions
E3AD-  20 02 D3  JSR D302      JSR EB78/ROM si touche frappée alors N = 1 et A = code ASCII
                               correspondant, sinon N = 0
E3B0-  10 E0     BPL E392      si pas touche, reboucle (continue d'afficher)
E3B2-  20 02 D3  JSR D302      si touche, pause l'affichage et re-teste touche
E3B5-  10 FB     BPL E3B2      poursuit pause jusqu'à obtenir une deuxième touche
E3B7-  C9 20     CMP #20       deuxième touche obtenue, est-ce un espace?
E3B9-  F0 D7     BEQ E392      si oui, reboucle reprend l'affichage
E3BB-  C9 1B     CMP #1B       sinon, est-ce un ESC?
E3BD-  D0 F3     BNE E3B2      si pas ESC, reprend scrutation clavier en E3B2
E3BF-  20 06 D2  JSR D206      JSR CBF0/ROM va à la ligne.
```

La pause est obtenue avec n'importe quelle touche, y compris ESC ou CTRL/C. La reprise n'est obtenue qu'avec la touche ESPACE. La touche ESC permet d'abandonner l'affichage du catalogue, mais la dernière ligne est quand même affichée avant de terminer.

Le test de touche ne marche qu'à la fin de chaque ligne affichée. Il semble intéressant de pauser avec une autre touche que la touche ESPACE afin d'éviter les rebonds dûs aux ratés de saisie. Il faut reconnaître que cette routine ne marche pas bien: on ne gagne pas à tous les coups lorsqu'on veut arrêter le défilement à l'endroit souhaité! Il serait possible de faire beaucoup mieux (voir la routine utilisée pour le commande CHKSUM).

Affiche la ligne de bilan de catalogue

De E3F1 à E408, j'ai été obligé de modifier le code d'origine à cause de mon extension "BIGDISK", qui permet de formater jusqu'à 101 pistes au lieu de 99 au maximum. Un lecteur de disquette ne peut guère formater plus de 82 pistes. Mais avec l'émulateur EUPHORIC de Fabrice Francès, pourquoi se priver de 3838 secteurs par disquette, même virtuelle, la seule limite étant celle de la bitmap. Le code modifié (24 octets en gras) permet d'afficher le nombre de pistes sur 3 digits au lieu de deux au détriment de l'affichage d'une virgule pas très utile qui a été supprimé (d'où la présence des 3 NOPs). Voir ci-dessus, au début de la commande DIR, quelques exemples de la nouvelle ligne de bilan de catalogue.

```
E3C2-  20 06 D2  JSR D206      CBF0/ROM va à la ligne
E3C5-  A9 2A     LDA #2A       place un "*" dans DEFAFF pour afficher dans les
E3C7-  8D 4C C0  STA C04C      digits libres devant le nombre de "sectors free"
E3CA-  AD 02 C2  LDA C202
E3CD-  AC 03 C2  LDY C203      lit dans AY le nombre de secteurs libres
E3D0-  20 56 D7  JSR D756      et l'affiche en décimal sur 4 digits
E3D3-  A2 07     LDX #07       indexe "_sectors_free_("
E3D5-  20 6C D3  JSR D36C      affiche (X+1) ème message de zone CEE7 terminé par "caractère + 128"
E3D8-  A9 00     LDA #00       place un "#00" dans DEFAFF pour afficher
E3DA-  8D 4C C0  STA C04C
```

```
E3DD-  A9 44        LDA #44     A = "D"
E3DF-  2C 09 C2     BIT C209    teste b7 de C209 (à 1 si double face)
E3E2-  30 02        BMI E3E6    si double face, saute l'instruction suivante
E3E4-  A9 53        LDA #53     A = "S"
E3E6-  20 2A D6     JSR D62A    XAFCAR affiche le caractère ASCII contenu dans A
E3E9-  A9 2F        LDA #2F     A = "/"
E3EB-  20 2A D6     JSR D62A    XAFCAR affiche le caractère ASCII contenu dans A
E3EE-  AD 06 C2     LDA C206    A = nombre de pistes par face
E3F1   A2 01        LDX #01     variable à utiliser pour obtenir 3 digits
E3F3   20 50 D7     JSR D750    routine d'affichage en décimal (ici sur 3 digits maximum)
E3F6   A9 2F        LDA #2F     "/"
E3F8   20 2A D6     JSR D62A    XAFCAR affiche le caractère ASCII contenu dans A
E3FB   AD 07 C2     LDA C207    nombre de secteurs / piste
E3FE   20 4E D7     JSR D74E    routine d'affichage en décimal sur 2 digits
E401   A9 29        LDA #29     ")"
E403   20 2A D6     JSR D62A    XAFCAR affiche le caractère ASCII contenu dans A
E406   EA EA EA     NOP NOP NOP 3 octets gagnés, mais difficilement récupérables!
E409-  A9 20        LDA #20     place un espace dans DEFAFF pour afficher
E40B-  8D 4C C0     STA C04C    devant le nombre de fichiers
E40E-  AD 04 C2     LDA C204
E411-  AC 05 C2     LDY C205    lit dans AY le nombre de fichiers
E414-  A2 01        LDX #01     nombre de 0 à 999
E416-  20 58 D7     JSR D758    affiche en décimal le nombre AY sur 3 digits
E419-  28           PLP         récupère les indicateurs
E41A-  A2 08        LDX #08     indexe le message " Files "
E41C-  20 6C D3     JSR D36C    affiche (X+1) ème message de zone CEE7 terminé par "caractère + 128"
```

CRLF conditionnel

Effectue un CRLF si imprimante ON, ou si mode 40 colonnes ou si ROM 1.1, sinon simple RTS sans CRLF

```
E41F-  2C F1 02     BIT 02F1    teste b7 de 02F1 (à 1 si imprimante en service)
E422-  30 0C        BMI E430    branche si imprimante en service
E424-  AD 6A 02     LDA 026A    sinon, teste b5 de 026A (à 1 si 40 colonnes)
E427-  29 20        AND #20     masque 0010 0000: ne garde que le b5
E429-  D0 05        BNE E430    branche si mode 40 colonnes
E42B-  AD 24 C0     LDA C024    sinon teste b7 de ATMORI (à 0 si ROM 1.0)
E42E-  10 03        BPL E433    saute l'instruction suivante si ROM ORIC-1
E430-  4C 06 D2     JMP D206    CBF0/ROM va à la ligne

E433-  60           RTS

E434-  4C 23 DE     JMP DE23    "SYNTAX_ERROR"
```

EXÉCUTION DE LA COMMANDE SEDORIC DELBAK

Rappel de la syntaxe

DELBAK (lecteur)

Effectue un DESTROY"*.BAK" sur la disquette présente dans le lecteur indiqué (ou le lecteur actif), c'est à dire efface sans demande de confirmation tous les fichiers "*.BAK". Les fichiers “*.BAK” protégés par PROT seront épargnés.

Saisie du paramètre et exécution

E437-	20 0D E6	JSR E60D	valide drive si indiqué, sinon valide DRVDEF
E43A-	D0 F8	BNE E434	"SYNTAX_ERROR" (DELBAK ne peut être suivi que d'une lettre de A à D ou de rien du tout, "fin d'instruction" est exigée)
E43C-	A2 09	LDX #09	pour lecture de ?????????BAK dans la table
E43E-	20 4D D3	JSR D34D	CCF7 + 9 et copie dans BUFNOM
E441-	38	SEC	Flag C = 1 pour DELBAK et DESTROY
E442-	B0 08	BCS E44C	suite forcée en E44C

EXÉCUTION DE LA COMMANDE SEDORIC DESTROY

Rappel de la syntaxe

DESTROY (nom_de_fichier_ambigu)

Efface, sans demande de confirmation, tous les fichiers correspondant au nom de fichier ambigu spécifié sur la disquette présente dans le lecteur indiqué (ou le lecteur actif). Le nom de fichier ambigu peut être réduit à une indication de lecteur, ou même être absent! Seuls les fichiers protégés par PROT seront épargnés.

Saisie du paramètre et exécution

E444-	38	SEC	Flag C = 1 pour DELBAK et DESTROY
E445-	24 18	BIT 18	continue en #E447

EXÉCUTION DE LA COMMANDE SEDORIC DEL

Rappel de la syntaxe

DEL (nom_de_fichier_ambigu)

Efface après demande de confirmation, tous les fichiers correspondant au nom de fichier ambigu spécifié sur la disquette présente dans le lecteur indiqué (ou le lecteur actif). Le nom de fichier ambigu peut être réduit à une indication de lecteur, ou même être absent! Si un nom de fichier non-ambigu est utilisé, le fichier correspondant sera détruit sans demande de confirmation. Les fichiers protégés par PROT ne peuvent être effacés.

Documentation supplémentaire

Voir en ANNEXE, ce qui se passe au niveau de la disquette, lorsqu'on supprime des fichiers.

Saisie du paramètre et exécution

E446-	18	CLC	Flag C = 0 pour DEL et continue en #E447

Suite commune DESTROY et DEL

E447-	08	PHP	sauvegarde les indicateurs dont C
E448-	20 51 D4	JSR D451	XNFA lit un nom de fichier ambigu à TXTPTR et l'écrit dans BUFNOM
E44B-	28	PLP	récupère les indicateurs dont C

Suite commune DELBAK, DESTROY et DEL

E44C-	6E 72 C0	ROR C072	C->b7 de C072 (0 = DEL, 1 = DELBAK ou DESTROY)
E44F-	20 2D DB	JSR DB2D	vérifie que la disquette en place est bien une disquette SEDORIC, cherche le fichier dans le catalogue, revient avec le bon secteur de catalogue en place (de coordonnées POSNMP et POSNMS) et avec X = POSNMX, pointant sur "l'entrée" cherchée ou avec Z = 1 si le fichier n'a pas été trouvé
E452-	D0 03	BNE E457	si trouvé, saute l'instruction suivante
E454-	4C DD E0	JMP E0DD	si pas trouvé, "FILE_NOT_FOUND_ERROR"
E457-	20 A0 D7	JSR D7A0	cherche "?" dans BUFNOM (C = 1 si pas trouvé)
E45A-	90 17	BCC E473	branche en E473 (il y a des "?" dans BUFNOM)
E45C-	20 64 E2	JSR E264	sinon, efface le fichier et libère ses secteurs
E45F-	90 46	BCC E4A7	si OK restructure le catalogue en E4A7, sinon...
E461-	60	RTS	simple RTS
E462-	20 2A D6	JSR D62A	XAFCAR affiche le caractère ASCII contenu dans A
E465-	20 06 D2	JSR D206	CBF0/ROM va à la ligne
E468-	20 41 DB	JSR DB41	ajuste POSNMX sur entrée suivante du catalogue et reprend la recherche, dans le catalogue, du fichier indiqué dans BUFNOM (Z = 1 si fini)
E46B-	AE 27 C0	LDX C027	X = POSNMX
E46E-	20 48 DB	JSR DB48	vérifie que le secteur de catalogue est en place, le met éventuellement
E471-	F0 34	BEQ E4A7	branche en E4A7 si fini
E473-	20 B4 DA	JSR DAB4	affiche le nom de fichier présent à POSNMX dans BUF3
E476-	2C 72 C0	BIT C072	teste flag b7
E479-	30 20	BMI E49B	branche en E49B si DELBAK ou DESTROY
E47B-	A2 0A	LDX #0A	si DEL, indexe message " (Y)es or (N)o:"
E47D-	20 6C D3	JSR D36C	affiche (X+1) ème message de zone CEE7 terminé par "caractère + 128"
E480-	20 02 D3	JSR D302	JSR EB78/ROM si touche frappée alors N = 1 et A = code ASCII correspondant, sinon N = 0
E483-	20 A1 D3	JSR D3A1	XminMAJ convertit (si nécessaire) en MAJUSCULE le caractère dans A
E486-	C9 4E	CMP #4E	est-ce "N"?
E488-	F0 D8	BEQ E462	si oui, reboucle en E462 (affiche N et passe au suivant)
E48A-	C9 1B	CMP #1B	est-ce "ESC"?
E48C-	F0 D3	BEQ E461	si oui, simple RTS
E48E-	C9 59	CMP #59	est-ce "Y"?

E490-	D0 EE	BNE E480	si pas "Y", reprend la saisie (seul N et Y autorisés)
E492-	20 2A D6	JSR D62A	XAFCAR affiche le "Y" contenu dans A
E495-	20 06 D2	JSR D206	CBF0/ROM va à la ligne
E498-	20 B4 DA	JSR DAB4	affiche le nom de fichier présent à POSNMX dans BUF3
E49B-	20 66 E2	JSR E266	XNOMDE DELete ce fichier
E49E-	B0 C5	BCS E465	si erreur, branche en E465 (au suivant)
E4A0-	A2 0B	LDX #0B	si OK, indexe le message "_DELETED CR**LF**"
E4A2-	20 6C D3	JSR D36C	affiche (X+1) ème message de zone CEE7 terminé par "caractère + 128"
E4A5-	30 C4	BMI E46B	et reboucle en E46B (au suivant)

Restructuration des secteurs de catalogue

Calcul du nombre de secteurs nécessaires selon le nombre de fichiers

Pour ce faire, il faut simplement diviser le nombre AX de fichiers présents sur la disquette par 15 (il y a 15 "entrées" par secteur de catalogue). Cette division est réalisée par soustractions successives AX = AX - 15, le résultat est donné par le nombre de soustractions effectuées (dans F5).

E4A7-	A9 00	LDA #00	remise à 0 de F5
E4A9-	85 F5	STA F5	(nombre de secteurs de catalogue nécessaires)
E4AB-	AD 04 C2	LDA C204	AX = nombre de fichiers présents
E4AE-	AE 05 C2	LDX C205	sur la disquette
E4B1-	18	CLC	C = 0, retenue pour la première soustraction
E4B2-	24 38	BIT 38	et continue en E4B4
E4B3-	38	SEC	C = 1, pas de retenue pour les soustractions suivantes
E4B4-	E9 0F	SBC #0F	A = A - 15 et C = 1 si A >= #0F (pas de retenue) cette soustraction porte sur le LL du nombre de fichiers présents, s'il devient négatif, il faut décrémenter HH et ce jusqu'à ce que HH devienne lui-même négatif. On a alors effectué une soutraction de trop, mais tout secteur de catalogue commencé doit être compté, donc le nombre de secteurs de catalogue nécessaire est arrondi par excès.
E4B6-	E6 F5	INC F5	augmente le nombre de soustractions effectuées
E4B8-	B0 F9	BCS E4B3	reboucle si pas dépassé
E4BA-	CA	DEX	décrémente HH du nombre de fichiers présents
E4BB-	10 F6	BPL E4B3	et reboucle si pas négatif
E4BD-	AE 08 C2	LDX C208	si négatif, c'est fini, lit le nombre actuel
E4C0-	E4 F5	CPX F5	de secteurs de catalogue et compare au nombre
E4C2-	F0 9D	BEQ E461	nécessaire, simple RTS si identique

Copie des "entrées" du dernier secteur dans les "entrées" libres

S'il y a au moins un secteur de catalogue de plus que nécessaire, on commence une boucle de compactage, pour récupérer le secteur utilisé en trop. Pour ce faire, on va chercher le dernier secteur de catalogue et en recopier les "entrées" valides dans les "trous" des secteurs de catalogue précédents. Rappel: dans chaque secteur de catalogue, la ou les "entrées" libres ont été groupées à la fin du secteur de catalogue (sous-programme E264/E266).

Recherche de l'avant-dernier et du dernier secteur de catalogue

E4C4-	CA	DEX	décrémente le nombre actuel de secteurs de catalogue afin de trouver l'avant-dernier secteur de catalogue (ce qui est moins simple que de trouver le dernier). Cet avant-dernier secteur deviendra le dernier et il faudra mettre à 0 les coordonnées du suivant, car il n'y en aura plus).
E4C5-	A9 14	LDA #14	
E4C7-	A0 04	LDY #04	AY = piste/secteur du premier secteur de catalogue
E4C9-	86 F5	STX F5	F5 = nombre de secteurs de catalogue restant à lire avant de trouver l'avant-dernier
E4CB-	C6 F5	DEC F5	décrémente F5, car c'est l'avant avant-dernier (c'est à dire l'antépénultième!) qui porte les coordonnées de l'avant-dernier
E4CD-	D0 06	BNE E4D5	si pas encore nul, continue en E4D5
E4CF-	8D 5C C0	STA C05C	si nul, AY contient les coordonnées piste/secteur de l'avant- dernier secteur de catalogue
E4D2-	8C 5D C0	STY C05D	on sauve ces coordonnées dans C05C/C05D
E4D5-	20 5D DA	JSR DA5D	XPBUF1 charge dans BUF1 ce secteur de catalogue
E4D8-	AD 00 C1	LDA C100	
E4DB-	AC 01 C1	LDY C101	AY = piste/secteur du secteur de catalogue suivant
E4DE-	D0 EB	BNE E4CB	reboucle si le n° de secteur suivant est valide
E4E0-	A0 10	LDY #10	sinon, le dernier secteur de catalogue est présent dans BUF1 c'est celui dont il faudra copier ailleurs les "entrées" valides
E4E2-	84 F5	STY F5	F5=#10 n° de l'octet de première "entrée" de BUF1
E4E4-	20 A5 DB	JSR DBA5	cherche POSNMX de première place libre dans directory en chargeant le catalogue dans BUF3 depuis le secteur #04 de la piste #14
E4E7-	A4 F5	LDY F5	Y vise premier caractère de la première "entrée" dans BUF1
E4E9-	CC 02 C1	CPY C102	le début de première "entrée" libre est-il atteint?
E4EC-	F0 14	BEQ E502	si oui, branche en E502 (fini pour ce secteur)
E4EE-	B9 00 C1	LDA C100,Y	sinon, lit octet à recopier
E4F1-	9D 00 C3	STA C300,X	et l'écrit à "l'entrée" libre trouvée dans BUF3
E4F4-	C8	INY	pour lecture suivante
E4F5-	E8	INX	pour écriture suivante
E4F6-	8E 02 C3	STX C302	prochain octet libre dans BUF3, la ou les place(s) vacante(s)
E4F9-	D0 EE	BNE E4E9	étant en fin de secteur, reboucle tant que X<>0
E4FB-	84 F5	STY F5	fin du secteur à compléter atteinte, garde Y dans F5 pour reprise de recopie s'il reste des "entrées" valides dans BUF1
E4FD-	20 82 DA	JSR DA82	XSCAT sauve BUF3 (secteur de catalogue rempli)
E500-	F0 E2	BEQ E4E4	reboucle pour chercher une autre place vacante à boucher

Dernier secteur de catalogue recopié, le libère et sauve BUF3

E502-	20 82 DA	JSR DA82	XSCAT sauve BUF3 (secteur de catalogue rempli)
E505-	CE 08 C2	DEC C208	décrémente nombre de secteurs de catalogue utilisés
E508-	AD 5C C0	LDA C05C	récupère les coordonnées de l'ancien avant-
E50B-	AC 5D C0	LDY C05D	dernier secteur de catalogue (= nouveau dernier)
E50E-	20 63 DA	JSR DA63	et le charge dans BUF3
E511-	AD 00 C3	LDA C300	
E514-	48	PHA	empile les coordonnées de l'ancien dernier
E515-	AD 01 C3	LDA C301	secteur de catalogue pour libération ultérieure

```
E518-  48          PHA
E519-  A9 00       LDA #00
E51B-  8D 00 C3    STA C300      met le "lien" à 0 (pas de suivant)
E51E-  8D 01 C3    STA C301
E521-  20 A4 DA    JSR DAA4      XSVSEC sauve le nouveau dernier secteur de catalogue selon DRIVE,
                                 PISTE, SECTEUR et RWBUF
E524-  68          PLA
E525-  A8          TAY           récupère dans AY les coordonnées de l'ancien
E526-  68          PLA           dernier secteur de catalogue pour libération
E527-  AE 08 C2    LDX C208
E52A-  E0 05       CPX #05       saute l'instruction suivante si le nombre de
E52C-  90 03       BCC E531      secteurs de catalogue est inférieur à 5
E52E-  20 15 DD    JSR DD15      XDETSE libère le secteur AY sur la bitmap
E531-  20 8A DA    JSR DA8A      l'entrée de cette routine XSMAP sauve BUF2 (bitmap) sur la disquette a été
                                 déportée en DC80 pour adaptation à la double bitmap. Un JSR DC80 plus
                                 direct aurait fait gagner un peu de temps.
E534-  4C A7 E4    JMP E4A7      et reboucle pour voir s'il y a encore un secteur de catalogue à récupérer
```

EXÉCUTION DE LA COMMANDE SEDORIC REN

Rappel de la syntaxe

REN ancien_nom_de_fichier_ambigu TO nouveau_nom_de_fichier_ambigu

Renomme les fichiers correspondants à l'ancien nom ambigu (NFAa dans ce qui suit) indiqué selon le modèle proposé avec le nouveau nom de fichier ambigu (NFAn dans ce qui suit). On peut utiliser deux noms de fichiers non ambigus, mais si des noms de fichiers ambigus sont utilisés, c'est à dire si des jokers sont présents (? ou *), leur place doit correspondre dans l'ancien et le nouveau nom.

Non documenté

Toujours le même problème avec l'utilisation possible des minuscules pour entrer les commandes SEDORIC: ça ne marche pas à tous les coups (bogue). Ici, le "TO" doit être tapé en MAJUSCULES.

Saisie des paramètres

```
E537-  20 51 D4    JSR D451      XNFA lit l'ancien nom (NFAa) à TXTPTR et l'écrit dans BUFNOM
E53A-  A2 0B       LDX #0B       pour copier 12 caractères de BUFNOM dans BUF1
E53C-  BD 29 C0    LDA C029,X    lecture dans BUFNOM (de C029 à C034)
E53F-  9D 00 C1    STA C100,X    écriture dans BUF1 (de C100 à C10B)
E542-  CA          DEX           caractère suivant
E543-  10 F7       BPL E53C      reboucle en E53C tant que l'index n'est pas négatif
```

Teste si un seul drive pour ancien nom (NFAa) et nouveau nom (NFAn)

```
E545-  AD 28 C0    LDA C028
E548-  48          PHA           empile préfixe de BUFNOM (n° du drive)
```

E549-	A9 C3	LDA #C3	demande le token "TO" (attention, bogue habituelle: "to" écrit en lettres minuscules ne marche pas: on obtient une "SYNTAX_ERROR")
E54B-	20 2E D2	JSR D22E	JSR D067/ROM puis D3A1/RAM overlay demande "TO" à TXTPTR, lit le caractère suivant à TXTPTR et le convertit en MAJUSCULE
E54E-	20 51 D4	JSR D451	XNFA lit le nouveau nom (NFAn) à TXTPTR et l'écrit dans BUFNOM
E551-	68	PLA	
E552-	CD 28 C0	CMP C028	si les drives demandés ne sont pas identiques
E555-	D0 13	BNE E56A	branche en D5AC ("INVALID_FILE_NAME_ERROR")

Comparaison des "?" de l'ancien nom (NFAa) et du nouveau nom (NFAn)
Cette comparaison est effectuée par la fin et inclut un octet de trop

E557-	A2 0C	LDX #0C	pour comparer 13 caractères (bogue?)
E559-	BC 29 C0	LDY C029,X	Y = caractère de NFAn dans BUFNOM (par la fin)
E55C-	BD 00 C1	LDA C100,X	A = caractère correspondant de NFAa dans BUF1
E55F-	9D 29 C0	STA C029,X	écrit ce caractère dans BUFNOM à la place de Y, ce qui revient à remettre le NFAa dans BUFNOM
E562-	C9 3F	CMP #3F	A est-il un "?"
E564-	F0 07	BEQ E56D	si oui, branche en E56D
E566-	C0 3F	CPY #3F	si A n'est pas un "?", Y est-il un "?"
E568-	D0 07	BNE E571	si ni A ni Y ne sont des "?", continue en E571
E56A-	4C AC D5	JMP D5AC	si l'un, mais pas l'autre, "INVALID_FILE_NAME_ERROR"
E56D-	C0 3F	CPY #3F	Y est-il aussi un "?"
E56F-	D0 F9	BNE E56A	si ce n'est pas le cas, "INVALID_FILE_NAME_ERROR"
			Les jokers doivent être à la même place dans l'ancien et le nouveau nom
E571-	98	TYA	Y, qui est précaire, est sauvé dans BUF1
E572-	9D 10 C1	STA C110,X	(dans la zone C110 à C11C qui reçoit NFAn)
E575-	CA	DEX	indexe le caractère précédent dans NFAa et NFAn
E576-	10 E1	BPL E559	et reboucle tant que X n'est pas négatif
			Continue si les "?" sont absents ou situés à la même place dans les 2 noms

Recherche du NFAa

E578-	20 2D DB	JSR DB2D	vérifie que la disquette en place est bien une disquette SEDORIC, cherche le fichier dans le catalogue, revient avec le bon secteur de catalogue en place (de coordonnées POSNMP et POSNMS) et avec X = POSNMX, pointant sur "l'entrée" cherchée ou avec Z = 1 si le fichier n'a pas été trouvé
E57B-	D0 08	BNE E585	continue en E585 si NFAa existe
E57D-	4C DD E0	JMP E0DD	si pas trouvé, "FILE_NOT_FOUND_ERROR"
E580-	20 41 DB	JSR DB41	ajuste POSNMX sur entrée suivante du catalogue et reprend recherche dans catalogue du fichier indiqué dans BUFNOM (Z = 1 si fini)
E583-	F0 76	BEQ E5FB	simple RTS si terminé

Sauve les coordonnées de “l’entrée” correspondant à l'ancien nom

E585-	AD 25 C0	LDA C025	

E588-	AC 26 C0	LDY C026	F5 reçoit POSNMP
E58B-	85 F5	STA F5	F6 reçoit POSNMS
E58D-	84 F6	STY F6	F7 reçoit POSNMX
E58F-	86 F7	STX F7	

Construit le nouveau nom et l'écrit dans BUFNOM

E591-	A0 00	LDY #00	Y indexe les caractères de BUFNOM (NFAa) et de la zone C110/C11C de BUF1 (NFAn) et X ceux de “l’entrée” de catalogue qui a été trouvée ci-dessus dans BUF3 (à renommer). Lorsqu'un caractère de NFAa contient un "?", il ne faut pas le changer, on relit ce caractère ancien dans BUFNOM et on le garde dans A. Si ce n'est pas un "?", on le remplace dans A par le caractère correspondant du NFAn pris dans BUF1. Puis on écrit le caractère de A dans BUFNOM, qui est donc mis à jour.
E593-	B9 29 C0	LDA C029,Y	lit caractère de NFAa (ancien) dans BUFNOM
E596-	C9 3F	CMP #3F	est-ce un "?" (C = 1 si c'est un "?")
E598-	D0 05	BNE E59F	sinon, continue en E59F pour saisir le nouveau
E59A-	BD 00 C3	LDA C300,X	si oui, lit caractère valide correspondant de “l’entrée”
E59D-	B0 03	BCS E5A2	et saut forcé de l'instruction suivante
E59F-	B9 10 C1	LDA C110,Y	lit le nouveau caractère correspondant de NFAn
E5A2-	99 29 C0	STA C029,Y	écrit le caractère dans BUFNOM qui est donc mis à jour
E5A5-	E8	INX	indexe caractère suivant de “l’entrée” dans BUF3
E5A6-	C8	INY	indexe caractère suivant dans BUFNOM et BUF1
E5A7-	C0 0C	CPY #0C	reboucle tant que 12 caractères
E5A9-	D0 E8	BNE E593	n'ont pas été mis à jour
E5AB-	BD 00 C3	LDA C300,X	complète BUFNOM avec les 4 octets suivants
E5AE-	99 29 C0	STA C029,Y	lus dans “l’entrée” de catalogue dans BUF3
E5B1-	E8	INX	(coordonnées piste/secteur du descripteur
E5B2-	C8	INY	principal et taille totale du fichier)
E5B3-	C0 10	CPY #10	rebouclage jusqu'à ce que le seizième octet
E5B5-	D0 F4	BNE E5AB	de “l’entrée” soit recopié dans BUFNOM

Cherche si le nouveau nom existe déjà

E5B7-	20 30 DB	JSR DB30	cherche fichier BUFNOM -> POSNMX/P/S ou Z = 1
E5BA-	08	PHP	sauvegarde les indicateurs 6502
E5BB-	F0 08	BEQ E5C5	s'il n'existe pas encore, OK, continue en E5C5
E5BD-	20 B4 DA	JSR DAB4	s'il existe déjà, affiche nom présent à POSNMX dans BUF3
E5C0-	A2 0E	LDX #0E	indexe "_ALREADY_EXISTSLFCR" (le nouveau nom ne doit pas déjà exister)
E5C2-	20 6C D3	JSR D36C	affiche (X+1) ème message de zone CEE7 terminé par "caractère + 128"

Remet le nouveau nom à la place de l'ancien et sauve catalogue

E5C5-	A5 F5	LDA F5	
E5C7-	A4 F6	LDY F6	
E5C9-	8D 25 C0	STA C025	restaure A = POSNMP de l'ancien nom
E5CC-	8C 26 C0	STY C026	restaure Y = POSNMS de l'ancien nom

```
E5CF-  A6 F7     LDX F7      restaure X = POSNMX de l'ancien nom
E5D1-  8E 27 C0  STX C027
E5D4-  28        PLP         récupère les indicateurs
E5D5-  D0 17     BNE E5EE    si fichier existe déjà, continue en E5EE, sinon
E5D7-  20 63 DA  JSR DA63    XPBUF3 charge dans BUF3 le secteur Y de la piste A
E5DA-  20 B4 DA  JSR DAB4    affiche le nom de fichier présent à POSNMX dans BUF3
E5DD-  20 EE DA  JSR DAEE    XBUCA copie BUFNOM dans BUF3 à la position POSNMX
E5E0-  A2 0F     LDX #0F     indexe le message "_-->_"
E5E2-  20 6C D3  JSR D36C    affiche (X+1) ème message de zone CEE7 terminé par "caractère + 128"
E5E5-  20 82 DA  JSR DA82    XSCAT sauve le secteur de catalogue BUF3 selon POSNMP et POSNMS
E5E8-  20 B4 DA  JSR DAB4    affiche le nom de fichier présent à POSNMX dans BUF3
E5EB-  20 06 D2  JSR D206    CBF0/ROM va à la ligne
```

Récupère le NFAa dans BUF1 (de C100 à C10B) et le copie dans BUFNOM

```
E5EE-  A0 0B     LDY #0B     pour copier 12 caractères
E5F0-  B9 00 C1  LDA C100,Y  lecture dans BUF1 (de C100 à C10B)
E5F3-  99 29 C0  STA C029,Y  écriture dans BUFNOM (de C029 à C034)
E5F6-  88        DEY         caractère suivant
E5F7-  10 F7     BPL E5F0    reboucle en E5F0 tant que l'index n'est pas négatif
E5F9-  30 85     BMI E580    reboucle en E580 lorsque 12 caractères copiés
E5FB-  60        RTS
```

EXÉCUTION DE LA COMMANDE SEDORIC SEARCH

Rappel de la syntaxe

SEARCH nom_de_fichier_ambigu

Met la variable EF (existing file) à 1 s'il existe au moins un fichier correspondant au nom spécifié sur la disquette présente dans le drive indiqué (ou dans le drive actif). Sinon EF prend la valeur zéro.

C’est carrément vicieux, car une valeur logique n'est TRUE que si elle vaut -1 elle est FALSE pour toute autre valeur, y compris 1. Donc quel que soit le résultat de SEARCH la variable est FALSE! Pour tester le résultat il faut utiliser IF EF=1 ... ou IF -EF ... ces deux conditions étant réalisées lorsqu’un fichier a été trouvé. Pourquoi faire simple, si on peut faire compliqué!

Saisie du paramètre et exécution

```
E5FC-  20 51 D4  JSR D451    XNFA lit un nom de fichier ambigu à TXTPTR et l'écrit dans BUFNOM
E5FF-  20 2D DB  JSR DB2D    vérifie que la disquette en place est bien une disquette SEDORIC, cherche
                             le fichier dans le catalogue, revient avec le bon secteur de catalogue en place
                             (de coordonnées POSNMP et POSNMS) et avec X = POSNMX, pointant
                             sur "l'entrée" cherchée ou avec Z = 1 si le fichier n'a pas été trouvé
E602-  08        PHP         sauvegarde les indicateurs dont Z
E603-  A9 00     LDA #00     A = #00 (par défaut pour "non trouvé")
E605-  28        PLP         récupère les indicateurs dont Z
```

E606-	F0 02	BEQ E60A	si pas trouvé, saute l'instruction suivante
E608-	A9 01	LDA #01	A = #01 (pour "trouvé")
E60A-	4C D5 D7	JMP D7D5	et dans les deux cas, continue en D7D5 où la variable EF (Existing File) reçoit A (0 si "non trouvé",1 si "trouvé"). Pour le traitement logique du résultat de la recherche, il faudra donc demander IF -EF THEN... et non IF EF THEN...

Valide drive si indiqué à TXTPTR, sinon valide DRVDEF

E60D-	AC 09 C0	LDY C009	Y = DRVDEF (n° drive actif par défaut)
E610-	20 9E D3	JSR D39E	XCRGOT relit le caractère à TXTPTR (sans incrémenter TXTPTR = CHRGOT), puis le convertit en MAJUSCULE, les espaces sont sautés, Z = 1 si fin d'instruction (0 ou :), C = 0 si caractère chiffre 0 à 9 (soit #30 à #39), sinon C = 1, Y et X inchangés
E613-	F0 0D	BEQ E622	branche en E622 si fin d'instruction
E615-	E9 41	SBC #41	rappel: C = 1 si lettre = OK pour soustraction
E617-	C9 04	CMP #04	teste s'il s'agit d'une lettre de A à D
E619-	B0 07	BCS E622	branche en E622 si ce n'est pas le cas
E61B-	A8	TAY	A (n° de drive) -> Y (écrase DRVDEF)
E61C-	20 C0 D7	JSR D7C0	vérifie si drive Y est "on line", si oui le valide, si non génère une erreur
E61F-	4C 98 D3	JMP D398	XCRGET incrémente TXTPTR, lit un caractère (CHRGET), les espaces sont sautés, le met dans A, le convertit en MAJUSCULE, Z = 1 si fin d'instruction (0 ou :), C = 0 si chiffre 0 à 9 (soit #30 à #39), sinon C =1. Y et X inchangés
E622-	20 C0 D7	JSR D7C0	vérifie si drive Y est "on line", si oui le valide, si non génère une erreur
E625-	4C 9E D3	JMP D39E	XCRGOT relit le caractère à TXTPTR (sans incrémenter TXTPTR = CHRGOT), puis le convertit en MAJUSCULE, les espaces sont sautés, Z = 1 si fin d'instruction (0 ou :), C = 0 si caractère chiffre 0 à 9 (soit #30 à #39), sinon C = 1, Y et X inchangés
E628-	4C DD E0	JMP E0DD	"FILE_NOT_FOUND_ERROR"
E62B-	4C D2 E2	JMP E2D2	"nom IS PROTECTED ERROR"

ROUTINES PRINCIPALES DE RAY MCLAUGHLIN

Les modifications apportées à SEDORIC par Ray sont vraiment géniales, notamment la double bitmap qui permet d'exploiter au mieux les lecteurs 3"1/2. Toutefois pour implanter ses nouvelles routines, Ray a sacrifié la table des vecteurs système située de FF43 à FFF9, soit 183 octets. Il faut dire que cette table est parfaitement inutile. Malheureusement, elle était documentée dans le manuel SEDORIC et certains programmateurs l'ont utilisée afin de se prémunir des éventuels changements d'adresses accompagnant une putative nouvelle version de SEDORIC. Ils avaient raison, car c'est la seule utilité de ce genre de table. Nous avons tous suffisamment regretté qu'une telle table n'existe pas dans la ROM V1.1 (ATMOS) notoirement incompatible avec la ROM V1.0 (ORIC-1).

Bilan final, certains programmes, parmi les meilleurs, étaient incompatibles avec les SEDORIC V2.x de Ray, car ils faisaient appel à des vecteurs qui n'existaient plus. La version 3.0 de SEDORIC est donc compatible avec tous les programmes écrits en langage machine, utilisant les routines de SEDORIC des versions 1.0 et 2.x et avec tous les programmes BASIC utilisant les commandes SEDORIC.

Je me suis attaché à remettre la table des vecteurs en place. La RAM overlay étant pleine comme un oeuf, il me fallait dégager de la place. J'ai donc crée une septième BANQUE dans laquelle j'ai placé certaines commandes dites "en mode immédiat", peut utilisées. Il s'agit de STATUS, PROT, UNPROT et SYSTEM, commandes qui étaient d'ailleurs groupée dans la RAM overlay de E628 à E62D. Cela ne pose aucun problème, étant donné la taille des disquettes 3"1/2. En effet, il est possible d'avoir en permanence une disquette master dans le drive système. Ceci à encore été amélioré par la suite avec le patch.001 qui permet de changer de BANQUE de manière transparente (voir en ANNEXE).

Les 6 derniers octets décrits précédemment de E628 à E62D appartiennent en fait au groupe de commandes STATUS, PROT, UNPROT, SYSTEM (E628 à E70A). Je les avait laissés par souci de compatibilité, mais en fait ils sont potentiellement disponibles. Il n'existe aucun appel au vecteur E628 ou au vecteur E62B autre qu'à partir du bloc E62E à E70A qui a été déplacé en BANQUE n°7. De même, de nulle part dans SEDORIC, on ne fait appel à des routines appartenant au bloc déplacé (routines E628, E62B, E62E, E6D0, E6D3, E702, E6E6, E6FF et E73D). Dans le bloc par contre, il est fait appel à E73D (SYNTAX_ERROR) que j'ai donc ajouté dans la BANQUE n°7 à la fin du bloc qui comprend donc finalement 230 octets (de C5F9 à C6DE).

En résumé, de E62E à E70A, se trouvaient les commandes suivantes:

STATUS ancienne entrée en E62E déplacée en E9F3.
PROT ancienne entrée en E6D0 déplacée en E9F6.
UNPROT ancienne entrée en E6D3 déplacée en E9F9.
SYSTEM ancienne entrée en E702 déplacée en E9FC.

A la place se trouve maintenant le code principal de Ray permettant de formater les disquettes avec le double de secteurs. Il n'est donc pas étonnant de trouver que dans cette zone 218 octets diffèrents sur 221, puisque le code d'origine a été complètement remplacé.

<u>De E62E à E634, (ex FF43 à FF49) Adaptation de XPMAP (en DA4C, prend la bitmap dans BUF2)</u>

E62E-	**A9 14**	LDA #14	piste où se trouve la bitmap
E630-	**A0 02**	LDY #02	secteur de la première page de bitmap

```
E632-  84 2F     STY 2F      b7 = 0, flag "première page de bitmap active"
E634-  60        RTS
```

Ecrit BUF2 dans le second secteur de bitmap sur la disquette

De E635 à E639, (ex FF4A à FF4E), nouveau code qui permet de sauver le deuxième secteur de bitmap.

```
E635-  A0 03     LDY #03     pour troisième secteur de la piste 20 (deuxième page de la bitmap)
E637-  4C 8B DC  JMP DC8B    écrit BUF2 dans le second secteur de bitmap sur la disquette
```

Nouvelle entrée de la routine XSMAP qui écrit BUF2 dans la deuxième page de bitmap sur la disquette et charge ensuite la première page dans BUF2

De E63A à E67E, (ex FF4F à FF93), adaptation de la routine XSMAP (situé en DA8A: sauve la bitmap sur la disquette).

Un JMP E63A placé au début de l'ancienne routine XSMAP entraîne le remplacement de celle-ci par la routine suivante, qui écrit la deuxième page de bitmap sur la disquette et charge ensuite la première page dans BUF2:

```
E63A-  18        CLC         flag entrée de "Sauve la seconde page de bitmap et charge la première"
E63B-  24 38     BIT 38      continue en E63D
```

Nouvelle entrée de la routine XSMAP qui écrit BUF2 dans la première page de bitmap sur la disquette et charge ensuite la deuxième page dans BUF2

En E63C, entrée secondaire (ex FF51 de Ray)

```
E63C-  38        SEC         flag entrée de "Sauve la première page et charge la deuxième"
E63D-  48        PHA
E63E-  98        TYA
E63F-  48        PHA         empile A et Y (sauvegarde)
E640-  AD 01 C0  LDA C001
E643-  48        PHA
E644-  AD 02 C0  LDA C002
E647-  48        PHA         empile PISTE et SECTEUR (sauvegarde)
E648-  A2 06     LDX #06
E64A-  BD 02 C2  LDA C202,X  empile les 7 octets du BUF2 de C202 à C208
E64D-  48        PHA         (nombre de secteurs libres, nombre de fichiers,
E64E-  CA        DEX         nombre de pistes par face, nombre de secteurs par
E64F-  10 F9     BPL E64A    piste et nombre de secteurs de directory).
```

Les informations correctes de la page active (première ou deuxième page) contenues dans ces 7 octets seront toujours copiées sur l'autre page de la bitmap (deuxième ou première). En effet, ces informations sont mises à jour continuellement.

```
E651-  B0 08     BCS E65B    si entré en E63C, continue en E65B pour écrire la première page de bitmap
```

sur la disquette, charger la deuxième, forcer 2F à 1 (flag deuxième page de bitmap active), mettre à jour les 7 octets d'information dans la deuxième page de bitmap, restaurer SECTEUR, PISTE Y et A et forcer C = 1.

Ecrit la deuxième page de bitmap sur la disquette et charge la première, met à zéro le b7 de 2F (flag"première bitmap chargée").

E653-	20 **35 E6**	JSR E635	sinon sauve BUF2 dans le troisième secteur de la piste 20
E656	20 **4C DA**	JSR DA4C	XPMAP prend le premier secteur de bitmap dans BUF2, vérifie le format, met à zéro le b7 de 2F (flag"première bitmap chargée").
E659	**F0 0C**	BEQ E667	suite forcée en E667 car Z = 1 après XPMAP

Ecrit la première page de bitmap sur la disquette et charge la deuxième, force 2F à 1 (flag deuxième page de bitmap active), met à jour les 7 octets d'information dans la deuxième page de bitmap, restaure SECTEUR, PISTE Y et A, force C = 1.

E65B	**86 2F**	STX 2F	force le b7 de 2F à 1 car X = #FF (deuxième page active)
E65D	**20 89 DC**	JSR DC89	écrit BUF2 dans le premier secteur de bitmap sur la disquette
E660	**A9 14**	LDA #14	indexe la piste n°20
E662	**A0 03**	LDY #03	et le secteur n°3, c'est à dire le deuxième secteur de la bitmap
E664	**20 50 DA**	JSR DA50	charge le secteur de bitmap de coordonnées AY dans BUF2 et vérifie le format

Met à jour les 7 octets d'information

Les informations correctes (mises à jour continuellement) provenant de la page précédente sont copiées dans la page qui vient d'être chargée.

E667	**A2 00**	LDX #00	
E669	**68**	PLA	récupère les 7 octets précédemment empilés
E66A	**9D 02 C2**	STA C202,X	et les copie dans BUF2 de C202 à C208
E66D	**E8**	INX	(nombre de secteurs libres, nombre de fichiers,
E66E	**E0 07**	CPX #07	nombre de pistes par face, nombre de secteurs par
E670	**90 F7**	BCC E669	piste et nombre de secteurs de directory)
E672	**68**	PLA	
E673	**8D 02 C0**	STA C002	
E676	**68**	PLA	
E677	**8D 01 C0**	STA C001	restaure SECTEUR et PISTE
E67A	**68**	PLA	
E67B	**A8**	TAY	
E67C	**68**	PLA	restaure Y et A
E67D	**38**	SEC	force C à 1
E67E	**60**	RTS	et retourne

Nouvelle entrée de la routine "Cherche un secteur libre"

De E67F à E6C3, (ex FF94 à FFD8), nouveau code modifiant le début de la routine DC7D

E67F	**A2 00**	LDX #00	X pointe au début de la liste des secteurs
E681	**BD 10 C2**	LDA C210,X	teste l'octet visé par X dans la liste des secteurs
E684	**D0 11**	BNE E697	si présence de secteurs libres, branche en E697
E686	**E8**	INX	sinon, vise l'octet suivant
E687	**E0 F0**	CPX #F0	la valeur maximale de X est-elle atteinte?
E689	**D0 F6**	BNE E681	sinon, reboucle en E681 (même page de bitmap)

La fin de la page courante est atteinte

E68B	**24 2F**	BIT 2F	si oui, teste si premier secteur de bitmap est présent
E68D	**10 03**	BPL E692	si présent, saute l'instruction suivante
E68F	**4C 78 DC**	JMP DC78	sinon (les 2 pages ont été examinées), "DISK_FULL_ERROR"

La première page était active, on la sauve, on charge la deuxième et on l'examine

E692	**20 3C E6**	JSR E63C	sauve le premier secteur de bitmap et charge le deuxième
E695	**B0 E8**	BCS E67F	reprise forcée en E67F car revient de E63C avec C = 1

Un secteur libre a été trouvé, on le réserve

E697	**AD 02 C2**	LDA C202	un octet "mappant" un secteur libre a été trouvé
E69A	**D0 03**	BNE E69F	
E69C	**CE 03 C2**	DEC C203	
E69F	**CE 02 C2**	DEC C202	décrémente le nombre de secteurs libres
E6A2	**24 2F**	BIT 2F	teste si on est sur le deuxième secteur de bitmap
E6A4	**30 03**	BMI E6A9	si oui, saute l'instruction suivante
E6A6	**4C 90 DC**	JMP DC90	si le premier secteur de bitmap est présent dans BUF2, reprend le cours normal de la routine "Cherche un secteur libre" en DC90

Sous-programme spécial pour réserver un secteur dans la deuxième page de bitmap

E6A9	**A9 60**	LDA #60	sinon, place un RTS en DCA8 à la fin du sous-programme "Inverse
E6AB	**8D A8 DC**	STA DCA8	le bit correspondant et met à jour la bitmap", c'est à dire inhibe le passage au sous-programme "Calcule le nombre de secteurs du début de la disquette jusqu'au secteur trouvé"
E6AE	**20 90 DC**	JSR DC90	reprise temporaire du cours normal de la routine "Cherche un secteur libre" en DC90 pour mettre à jour la bitmap en BUF2
E6B1	**A9 A9**	LDA #A9	restaure la valeur originale de l'octet DCA8
E6B3	**8D A8 DC**	STA DCA8	qui avait été écrasée par un RTS
E6B6	**8A**	TXA	nombre d'octets situés avant l'octet modifié
E6B7	**A2 00**	LDX #00	HH de ce nombre d'octets
E6B9	**18**	CLC	on ajoute les #F0 octets déjà utilisés
E6BA	**69 F0**	ADC #F0	dans le premier secteur de la bitmap
E6BC	**90** 01	BCC E6BF	si pas de retenue, continue en E6BF
E6BE	**E8**	INX	sinon reporte cette retenue
E6BF	**86 F3**	STX F3	dans F3 (HH du nombre d'octets)
E6C1	**4C AD DC**	JMP DCAD	reprise en DCAD du cours normal de la routine "Calcule le nombre de secteurs du début de la disquette jusqu'au secteur trouvé"

De E6C4 à E6E4, (ex FFD9 à FFF9), adaptation du sous-programme DCD6: "A quel bit de la bitmap correspond le secteur AY à libérer".

Trois instructions (ROR TAX et SEC, situées de DD0B à DD0D) ont été écrasées par un JMP E6C4 pour permettre le passage à la routine complémentaire suivante:

E6C4	**6A**	ROR	remplace le ROR écrasé en DD0B (A est l'octet de la bitmap qu'il faut modifier)
E6C5	**A6 F3**	LDX F3	HH du résultat de la division, est à 1 si le nombre d'octets présents avant l'octet à libérer est supérieur à #7FF soit 2047, ce qui est facilement atteint avec une disquette formatée en 80 pistes par face
E6C7	**D0 04**	BNE E6CD	si c'est le cas, saute les 2 instructions suivantes
E6C9	**C9 F0**	CMP #F0	teste si A < #F0 c'est à dire si l'octet à modifier est sur la première bitmap
E6CB	**90 0F**	BCC E6DC	si oui, continue en E6DC

Le secteur libre est dans la deuxième page de bitmap

E6CD	**24 2F**	BIT 2F	teste si le b7 de 2F est à 1, c'est à dire si le deuxième secteur de bitmap est présent dans BUF2
E6CF	**30 03**	BMI E6D4	si oui, saute l'instruction suivante
E6D1	**20 3C E6**	JSR E63C	sauve le premier en place dans BUF2 et charge le deuxième
E6D4	**38**	SEC	pour faire une soustraction
E6D5	**E9 F0**	SBC #F0	calcule l'octet du deuxième qui sera modifié
E6D7	**AA**	TAX	remplace le TAX écrasé en DD0C. A est l'octet du deuxième secteur de bitmap qu'il faut modifier
E6D8	**38**	SEC	remplace le SEC écrasé en DD0D
E6D9	**4C 0E DD**	JMP DD0E	et reprend le cours normal du sous-programme "Elabore un masque dont un bit représente le secteur à libérer" en DD0E

Le secteur libre est dans la première page de bitmap

E6DC	**24 2F**	BIT 2F	teste si le premier secteur de bitmap est présent dans BUF2
E6DE	**10 F7**	BPL E6D7	si oui, reprend en E6D7
E6E0	**20 3A E6**	JSR E63A	sinon, sauve le deuxième secteur de bitmap et charge le premier
E6E3	**B0 F2**	BCS E6D7	et reprend en E6D7 (C mis à 1 par le sous-programme E63A)

De E6E5 à E70A, série de **38 NOP**s, disponibles pour une future implémentation.

EXÉCUTION DE LA COMMANDE SEDORIC KEY

Rappel de la syntaxe

KEY SET ou **KEY OFF**

Inhibe (OFF, ce qui accélère les programmes) et rétablit (SET) la scrutation du clavier. Attention: n'utilisez pas KEY SET en mode direct sous peine de perdre la main. En effet, le clavier devient alors inutilisable,

seul un reset pourra vous la rendre. N'utilisez donc KEY SET qu'en mode programme sauf si vous êtes conscient de ce que vous faites!

Saisie du paramètre et exécution

E70B-	20 4D E9	JSR E94D	XSETOFF teste si le paramètre à TXTPTR est SET ou OFF, retourne avec C = 1 si SET ou C = 0 si OFF sinon génère une "SYNTAX_ERROR"
E70E-	90 09	BCC E719	continue en E719 si OFF
E710-	AD 07 03	LDA 0307	si SET, lit HH du latch de T1
E713-	8D 05 03	STA 0305	et le copie dans HH du compteur de T1
E716-	A9 40	LDA #40	pour autoriser les interruptions T1
E718-	2C A9 00	BIT 00A9	continue en E71B
E719-	A9 00	LDA #00	pour interdire les interruptions T1
E71B-	8D 0B 03	STA 030B	place A dans le registre des interruptions du 6522
E71E-	60	RTS	

EXÉCUTION DE LA COMMANDE SEDORIC OUT

Rappel de la syntaxe

OUT code_ASCII

Envoie le code ASCII indiqué sur le port parallèle. Cette commande équivaut à LPRINT CHR$(code_ASCII), en plus efficace et sans les bogues de l'ORIC 1. Elle permet entre autres de configurer une imprimante en lui envoyant des codes de contrôle.

E71F-	20 7F D2	JSR D27F	CF17/ROM et CF09/ROM et D8CB/ROM évalue un nombre entier à TXTPTR et le retourne dans X (code ASCII de 0 à 255)
E722-	08	PHP	sauvegarde les indicateurs 6502 dont I
E723-	78	SEI	interdit les interruptions
E724-	8E 01 03	STX 0301	place le code ASCII dans VIADRA (DATA port A)
E727-	AD 00 03	LDA 0300	lit la valeur présente dans VIADRB (DATA port B)
E72A-	29 EF	AND #EF	masque 1110 1111, force à 0 le b4 (strobe low)
E72C-	8D 00 03	STA 0300	et la remet en place dans VIADRB
E72F-	09 10	ORA #10	masque 0001 0000, force à 1 le b4 (strobe high)
E731-	8D 00 03	STA 0300	et la remet en place dans VIADRB
E734-	28	PLP	récupère les indicateurs 6502 dont I
E735-	A9 02	LDA #02	masque 0000 0010 pour tester le b1 de VIAIFR (ACK)
E737-	2C 0D 03	BIT 030D	ce BIT effectue "masque" AND "contenu de 030D"
E73A-	F0 FB	BEQ E737	reboucle en attente tant que b2 est nul
E73C-	60	RTS	retourne lorsque transition CA1 reçue (ACK reçu)
E73D-	4C 23 DE	JMP DE23	"SYNTAX_ERROR"

EXÉCUTION DE LA COMMANDE SEDORIC WIDTH

Rappel de la syntaxe

WIDTH ou
WIDTH nombre_de_caractères_à_l'écran ou
WIDTH LPRINT nombre_de_caractères_à_l'imprimante

Permet de fixer la largeur de l'affichage ou de l'impression, c'est à dire le nombre de caractères au bout duquel un CR + LF seront automatiquement ajoutés. Les valeurs par défaut sont respectivement de 40 et 80 pour l'ATMOS et de 53 et 93 pour l'ORIC 1 (bogue classique). De plus l'ORIC 1 ne peut pas faire de différence entre WIDTH et WIDTH LPRINT: que l'un ou l'autre soit utilisé, les affichages écran et imprimante seront affectés tous les deux et de manière identique.

Non documenté

Utilisée sans paramètre, la commande WIDTH remet les valeurs par défaut.

Enfin, toujours le même problème avec l'utilisation possible des minuscules pour entrer les commandes SEDORIC: ça ne marche pas à tous les coups (bogue). Ici, il faut utiliser "LPRINT" et non "lprint".

Saisie du paramètre

E740-	08	PHP	sauvegarde les indicateurs 6502, notamment Z
E741-	A2 00	LDX #00	mise à zéro de X sans affecter Z d'origine
E743-	28	PLP	récupère les indicateurs 6502, notamment Z
E744-	F0 08	BEQ E74E	continue en E74E si fin d'instruction (il faudra utiliser le nombre de caractères par défaut)
E746-	C9 8F	CMP #8F	le paramètre est-il le token de LPRINT?
E748-	D0 04	BNE E74E	sinon, continue en E74E avec X = 0 (pas de LPRINT)
E74A-	20 98 D3	JSR D398	XCRGET incrémente TXTPTR, lit un caractère (CHRGET), les espaces sont sautés, le met dans A, le convertit en MAJUSCULE, Z = 1 si fin d'instruction (0 ou :), C = 0 si chiffre 0 à 9 (soit #30 à #39), sinon C =1. Y et X inchangés
E74D-	E8	INX	compteur de paramètres: X = 1 (LPRINT présent)
E74E-	8A	TXA	
E74F-	48	PHA	empile X (à 1 si largeur imprimante, sinon à 0)
E750-	2C 24 C0	BIT C024	teste b7 de ATMORI (à 1 si ATMOS)
E753-	30 02	BMI E757	continue en E757 si ROM V 1.1
E755-	E8	INX	X = X + 2 si ROM V 1.0
E756-	E8	INX	(donc 4 cas de 0 à 3 selon écran/imprimante et ROM)
E757-	BD 0C CD	LDA CD0C,X	lit un octet dans la table CD0C/CD0F (valeur par défaut, c'est à dire #28 (40), #50 (80), #35 (53) et #5D (93))
E75A-	AA	TAX	cette valeur prend la place de l'index dans X
E75B-	20 9E D3	JSR D39E	XCRGOT relit le caractère à TXTPTR (sans incrémenter TXTPTR = CHRGOT), puis le convertit en MAJUSCULE, les espaces sont sautés, Z = 1 si fin d'instruction (0 ou :), C = 0 si caractère chiffre 0 à 9 (soit #30 à #39), sinon C = 1, Y et X inchangés
E75E-	F0 03	BEQ E763	suite en E763 avec valeur par défaut si fin d'instruction
E760-	20 7F D2	JSR D27F	CF17/ROM et CF09/ROM et D8CB/ROM évalue un nombre entier à

			TXTPTR et le retourne dans X (nombre de caractères de 0 à 255)
E763-	2C 24 C0	BIT C024	teste b7 de ATMORI
E766-	30 13	BMI E77B	continue en E77B si ROM V 1.1

WIDTH pour ROM V 1.0

E768-	68	PLA	élimine le flag LPRINT (pas de différence)
E769-	86 31	STX 31	met à jour le nombre de caractères par ligne pour le périphérique de sortie courant (écran ou imprimante)
E76B-	8A	TXA	et garde une copie de cette valeur dans A

Calcule la position de la dernière tabulation

E76C-	38	SEC	prépare une soustraction
E76D-	E9 08	SBC #08	retire 8 et reboucle tant que le nombre de
E76F-	B0 FC	BCS E76D	caractères par ligne est positif ou nul
E771-	49 FF	EOR #FF	masque 1111 1111, inverse les bits du résultat
E773-	E9 06	SBC #06	et retire 6
E775-	18	CLC	prépare une addition
E776-	65 31	ADC 31	résultat précédent + nombre de caractères par ligne
E778-	85 32	STA 32	donne position maximale pour tabulation par ","
E77A-	60	RTS	

WIDTH pour ROM V 1.1

E77B-	2C F1 02	BIT 02F1	teste le flag imprimante (b7 à 1 si périphérique courant)
E77E-	10 0A	BPL E78A	continue en E78A si l'écran est le périphérique courant

L'imprimante est le périphérique courant

E780-	68	PLA	teste si le flag "LPRINT était présent dans la commande WIDTH"

"WIDTH imprimante" lorsque l'imprimante est le périphérique courant

E781-	D0 E6	BNE E769	si oui, continue en E769 (met à jour le nombre de caractères par ligne pour le périphérique de sortie courant (écran ou imprimante, puis calcule la position de la dernière tabulation et retourne)

"WIDTH écran" lorsque l'imprimante est le périphérique courant

E783-	8E 57 02	STX 0257	sinon met à jour la longueur d'une ligne écran
E786-	8D 59 02	STA 0259	force à zéro la position horizontale écran
E789-	60	RTS	

L'écran est le périphérique courant

E78A-	68	PLA	teste flag LPRINT présent dans la commande WIDTH"

"WIDTH écran" lorsque l'écran est le périphérique courant

E78B-	F0 DC	BEQ E769	sinon, continue en E769 (met à jour le nombre de caractères par ligne pour le périphérique de sortie courant (écran ou imprimante, puis calcule la position de la dernière tabulation et retourne)

"WIDTH imprimante" lorsque l'écran est le périphérique courant

E78D-	8E 56 02	STX 0256	sinon met à jour la longueur d'une ligne imprimante
E790-	A9 00	LDA #00	
E792-	8D 58 02	STA 0258	force à zéro la position horizontale imprimante
E795-	60	RTS	

EXÉCUTION DE LA COMMANDE SEDORIC RANDOM

Rappel de la syntaxe

RANDOM ou **RANDOM expression_numérique**

Génère un nombre aléatoire au hasard (si pas d'argument) ou en utilisant comme germe la valeur indiquée (dans ce cas la même séquence sera toujours obtenue).

Non documenté

Les indications du manuel sont plutôt spartiates. Il n'est pas indiqué quelles sont les limites possibles de l'expression numérique (il n'y en a pas, ce sont celles du BASIC), ni où et comment on récupère le nombre aléatoire (RANDOM remplace tout simplement la commande BASIC RND).

Saisie éventuelle du paramètre

E796-	F0 06	BEQ E79E	continue en E79E s'il n'y a pas de paramètre

Utilise la valeur indiquée comme germe

E798-	20 16 D2	JSR D216	JSR CF17/ROM et CF09/ROM évalue une expression numérique à TXTPTR, retourne avec cette valeur qui servira de germe dans ACC1
E79B-	4C E2 D2	JMP D2E2	JSR E37D/ROM génère un nombre entre 0 et 1

Utilise les timers du VIA comme germe

E79E-	AD 04 03	LDA 0304	LL du compteur T1
E7A1-	AC 05 03	LDY 0305	HH du compteur T1
E7A4-	85 D0	STA D0	copiés dans les octets de poids le plus
E7A6-	84 D1	STY D1	fort de ACC1
E7A8-	AD 08 03	LDA 0308	LL du compteur T2
E7AB-	AC 09 03	LDY 0309	HH du compteur T2
E7AE-	85 D2	STA D2	copiés dans les octets

```
E7B0-  84 D3      STY D3       suivants de ACC1
E7B2-  4C 9B E7   JMP E79B     un JMP D2E2 eût été plus direct!

E7B5-  4C 23 DE   JMP DE23     "SYNTAX_ERROR"
```

EXÉCUTION DE LA COMMANDE SEDORIC RESET

Rappel de la syntaxe

RESET tout court

Comme son nom l'indique, cette commande simule un reset système (appui sur le bouton reset du drive master).

```
E7B8-  D0 FB      BNE E7B5     "SYNTAX_ERROR" s'il y a un paramètre
E7BA-  78         SEI          interdit les interruptions
E7BB-  A9 00      LDA #00      masque pour le registre 0314 (ROM/RAM overlay)
E7BD-  4C AD 04   JMP 04AD     et effectue un coldstart FFFC/ROM
```

EXÉCUTION DE LA COMMANDE SEDORIC PR

Rappel de la syntaxe

PR SET ou **PR OFF**

Redirige les sorties vers l'imprimante (SET) ou vers l'écran (OFF). C'est seulement dans ce dernier cas qu'une différence est faite entre PRINT et LPRINT. L'affichage du "Ready" entraîne automatiquement un PR OFF.

Le manuel précise qu'en raison d'une bogue de l'ATMOS, il faut insérer un PR OFF dans le programme avant qu'il ne s'arrête (STOP, END, CTRL/C ou simple fin de programme). Sinon, il faut réinitialiser la longueur de la ligne écran avec une commande WIDTH.

Utilisation en "Langage Machine"

Pour PR SET, passer sur la RAM overlay, effectuer un JSR E7C5 et revenir sur la ROM. Pour PR OFF, rester sur la ROM et effectuer simplement un JSR C82F.

Saisie du paramètre

```
E7C0-  20 4D E9   JSR E94D     XSETOFF teste si le paramètre à TXTPTR est SET ou OFF, retourne avec
                               C = 1 si SET ou C = 0 si OFF sinon génère une "SYNTAX_ERROR"
```

PR OFF

E7C3-	90 11	BCC E7D6	continue en D1C4 si OFF (on veut mettre l'imprimante hors service) pour exécuter un JSR C82F/ROM (mettre l'imprimante hors service)

PR SET

E7C5-	AC 24 C0	LDY C024	si SET (C = 1), teste ATMORI (#80 si ROM V1.1)
E7C8-	D0 03	BNE E7CD	c'est le cas, continue en E7CD
E7CA-	6E F1 02	ROR 02F1	pour ORIC-1, force à 1 le b7 du flag imprimante
E7CD-	4C BC D1	JMP D1BC	JSR C816/ROM mettre l'imprimante en service

EXÉCUTION DE LA COMMANDE SEDORIC LDIR

Rappel de la syntaxe

LDIR nom_de_fichier_ambigu

DIR avec sortie sur l'imprimante (ou plutôt sur le port parallèle).

E7D0-	20 C5 E7	JSR E7C5	effectue un PR SET
E7D3-	20 44 E3	JSR E344	effectue un DIR
E7D6-	4C C4 D1	JMP D1C4	JSR C82F/ROM mettre l'imprimante hors service

EXÉCUTION DE LA COMMANDE SEDORIC RESTORE

Rappel de la syntaxe

!RESTORE (n°_de_ligne) **REJETEZ TOUTE AUTRE SYNTAXE**

Place le pointeur de DATA au début de la ligne indiquée ou à défaut au début du programme BASIC. Attention RESTORE tout court est une commande BASIC. Pour appeler le RESTORE de SEDORIC, qui seul peut être re-numéroté (RENUM), il faut utiliser impérativement la syntaxe indiquée ci-dessus.

Mal documenté

restore (n°_de_ligne) marche toujours, mais SEDORIC V3.0 n'assure plus l'usage des minuscules, qui souffre de beaucoup trop d'exceptions. Attention, tout l'intérêt du RESTORE SEDORIC par rapport au RESTORE BASIC est de pouvoir indiquer un n° de ligne. Donc, si l'on ne veut pas courir de risque avec RENUM qui ne met pas à jour la forme "restore" en minuscules, mieux vaut ne jamais l'utiliser!

Entrée de la commande !RESTORE

E7D9-	08	PHP	sauvegarde les indicateurs 6502 dont Z
E7DA-	A6 9A	LDX 9A	XY adresse de début du programme BASIC

E7DC-	A4 9B	LDY 9B	pour valeur par défaut (premier lien de ligne)
E7DE-	28	PLP	récupère les indicateurs DONT Z
E7DF-	F0 0A	BEQ E7EB	continue en E7EB s'il n'y a pas de paramètre
E7E1-	20 FA D2	JSR D2FA	E853/ROM évalue un nombre non signé à TXTPTR et revient avec ce nombre dans YA, 33/34 et D3/D4 et TXTPTR est mis à jour sur l'octet suivant ce nombre
E7E4-	20 9C D1	JSR D19C	JSR C6B3/ROM cherche l'adresse de cette ligne BASIC
E7E7-	A6 CE	LDX CE	XY pointe sur LL du lien de cette ligne
E7E9-	A4 CF	LDY CF	il faut pointer sur le #00 qui précède
E7EB-	8A	TXA	teste si LL est nul
E7EC-	D0 01	BNE E7EF	sinon, se contente de décrémenter LL
E7EE-	88	DEY	si oui, décrémente HH puis
E7EF-	CA	DEX	décrémente LL
E7F0-	86 B0	STX B0	place l'adresse du #00 précédant le début de ligne
E7F2-	84 B1	STY B1	dans le pointeur de DATA
E7F4-	60	RTS	

EXÉCUTION DE LA COMMANDE SEDORIC QUIT

Rappel de la syntaxe

QUIT tout court

Cette commande permet de "quitter" SEDORIC et de retrouver un système compatible ORIC 1/ATMOS. Seul le vecteur ! reste branché sur SEDORIC, ce qui permet d'en utiliser encore les commandes. QUIT restore les pointeurs qui avaient été détournés par SEDORIC (IRQ, NMI) et inhibe les touches de fonction. Cette instruction a été créée afin de pouvoir exécuter certains programmes incompatibles avec SEDORIC.

Non documenté

Comment revenir à l'état initial sous SEDORIC? Il n'existe pas de commande appropriée à par le brutal !RESET et encore, selon le programme exécuté, il ne marche pas toujours. Il serait intéressant de programmer une commande qui ferait le contraire de QUIT et qui pourrait s'appeler RETRIEVE par exemple.

E7F5-	D0 BE	BNE E7B5	"SYNTAX_ERROR" (tout paramètre est interdit avec QUIT)
E7F7-	AD 3E 04	LDA 043E	
E7FA-	AC 3F 04	LDY 043F	redirige le vecteur de l'interpréteur F0/F1 sur ECB9
E7FD-	85 F0	STA F0	au lieu de 0400 (la page 4 n'est plus utilisée)
E7FF-	84 F1	STY F1	
E801-	08	PHP	sauve les indicateurs 6502 dont I
E802-	78	SEI	interdit les interruptions
E803-	2C 24 C0	BIT C024	teste ATMORI (#00 si V1.0 et #80 si V1.1)
E806-	10 20	BPL E828	branche en E828 si ORIC-1

ATMOS

```
E808-  A9 22     LDA #22
E80A-  A0 EE     LDY #EE     redirige le vecteur IRQ 0245/0246
E80C-  8D 45 02  STA 0245    sur EE22 au lieu de 0488
E80F-  8C 46 02  STY 0246
E812-  A9 78     LDA #78
E814-  A0 EB     LDY #EB     redirige le sous-programme "Gérer le tampon touche" 023C/023D
E816-  8D 3C 02  STA 023C    sur EB78 au lieu de 045B (les touches de
E819-  8C 3D 02  STY 023D    fonctions ne sont plus accessibles)
E81C-  A9 B2     LDA #B2
E81E-  A0 F8     LDY #F8     redirige le vecteur NMI 0248/0249
E820-  8D 48 02  STA 0248    sur F8B2 au lieu de 04C4
E823-  8C 49 02  STY 0249
E826-  28        PLP         récupère les indicateurs dont I
E827-  60        RTS         et retourne sur ATMOS
```

ORIC-1

```
E828-  A9 03     LDA #03
E82A-  A0 EC     LDY #EC     redirige le vecteur IRQ 0229/022A
E82C-  8D 29 02  STA 0229    sur EC03 au lieu de 0488
E82F-  8C 2A 02  STY 022A
E832-  A9 30     LDA #30
E834-  A0 F4     LDY #F4     redirige le vecteur NMI 022C/022D
E836-  8D 2C 02  STA 022C    sur F430 au lieu de 04C4
E839-  8C 2D 02  STY 022D
E83C-  28        PLP         récupère les indicateurs
E83D-  60        RTS         et retourne sur ORIC-1
```

Remet à jour TXTPTR et n° de ligne après un STRUN

Le dernier code de la table des mots-clés du DOS (en C9DE/CBB9) est FF (255). A ce code correspond l'adresse d'exécution E83E dans la table des mots-clés SEDORIC (en CC27/CCF6). L'ANNEXE 6 du manuel SEDORIC (codes des touches de fonctions) indique que le code 255 correspond à la "Génération des numéros de lignes". Le sous-programme E83E n'étant mentionné nulle part ailleurs dans la RAM overlay, il faudrait en conclure qu'il s'agit bien de la commande de génération des n° de lignes. En fait il s'agit d'un sous-programme assurant la restauration du TXTPTR et du n° de ligne après exécution d'un STRUN (voir commande STRUN).

```
E83E-  AD 13 C0  LDA C013    remise à jour de TXTPTR avec les 2 octets qui
E841-  AC 14 C0  LDY C014    se trouvent en C013/C014 et qui contiennent
E844-  85 E9     STA E9      l'ancienne valeur de TXTPTR
E846-  84 EA     STY EA
E848-  AD 11 C0  LDA C011    remise à jour du n° de ligne courante avec les
E84B-  AC 12 C0  LDY C012    2 octets C011/C012 qui contiennent
E84E-  85 A8     STA A8      l'ancienne valeur du n° de ligne
E850-  84 A9     STY A9
E852-  60        RTS
```

EXÉCUTION DE LA COMMANDE SEDORIC STRUN

Rappel de la syntaxe

STRUN expression_alphanumérique

C'est une des commandes les plus géniales de SEDORIC. Elle exécute, en mode programme uniquement, l'expression alphanumérique indiquée qui doit correspondre à une ligne de commande BASIC et/ou SEDORIC. Tout ce passe comme si la ligne correspondante était exécutée en mode direct. Il est possible d'utiliser non seulement les mots-clés BASIC et SEDORIC, mais aussi d'exécuter en mode programme des commandes autorisées uniquement en mode direct, telles que STRUN "B-" pour changer de lecteur actif. Seule contrainte: si la chaîne contient des mots-clés BASIC, elle doit être “tokenisée” au préalable avec la commande TKEN. Il est possible de constituer des commandes pré-definies stockées dans un tableau ou dans des DATA ou de combiner des éléments pour exécuter une commande selon le contexte. Avis aux petits bricoleurs: c'est le moment de sévir!

Non documenté

Les commandes STRUN et TKEN sont les deux seules commandes SEDORIC pour lesquelles le mode direct est interdit.

La chaîne “tokenisée” ne doit pas comporter plus de #44 (67) octets. Rappel un mot-clé BASIC compte pour un octet (exemple #8F pour le token LPRINT) et non pour plusieurs caractères (6 dans le cas de cet exemple).

Comme indiqué ci-dessus, il est possible de changer de lecteur actif à l'intérieur d'un programme BASIC grâce à STRUN, alors qu'aucune commande SEDORIC n'est prévue pour cela.

```
60000 REM Sous-programme changement de drive actif
60010 PRINT"Drive (A, B, C ou D)?";:GET DR$:PRINT DR$
60020 IF DR$<"A" OR DR$>"D" THEN PING:GOTO 60010
60030 DR$=DR$+"-":TKEN DR$:STRUN DR$:RETURN
```

NB: dans un programme en "Langage Machine" il faudrait faire:

```
JSR 0472      (passage sur la RAM overlay)
LDA #00       (00 pour A, 01 pour B, 02 pour C et 03 pour D)
STA C000      (indique le n° de drive actif)
JSR 0472      (retour sur la ROM)
```

Utilisation en "Langage Machine"

Impossible puisque cette commande retourne à l'exécution de la commande suivante dans un programme BASIC. On peut cependant s'inspirer de l'utilisation du TIB (Terminal Input Buffer) (voir en ANNEXE).

Saisie et vérification du paramètre

E853-	20 5C D2	JSR D25C	JSR D4D2/ROM interdit le mode direct
E856-	20 24 D2	JSR D224	JSR CF17/ROM évalue une expression à TXTPTR, retourne avec une valeur numérique dans ACC1 (et #00 dans 28) ou l'adresse d'une chaîne dans

			D3/D4 (et #FF dans 28) et A, N et Z positionnés selon l'exposant (résultat numérique) ou la longueur de chaîne
E859-	20 74 D2	JSR D274	JSR D7D0 et CF09/ROM vérifie si l'expression évaluée à TXTPTR est bien alphanumérique, retourne l'adresse de la chaîne dans XY et 91/92 ainsi que sa longueur dans A
E85C-	C9 44	CMP #44	teste si la chaîne comporte plus de 67 octets
E85E-	B0 3A	BCS E89A	si oui, "STRING_TOO_LONG_ERROR"

Exécution de la commande proprement dite

E860-	AA	TAX	longueur de la chaîne “tokenisée”
E861-	A8	TAY	idem
E862-	88	DEY	pour copier les A octets de la chaîne
E863-	B1 91	LDA (91),Y	lit un octet de la chaîne
E865-	99 35 00	STA 0035,Y	et le copie dans le TIB (tampon clavier)
E868-	88	DEY	décrémente le compteur
E869-	10 F8	BPL E863	reboucle en E863 s'il en reste à copier
E86B-	C8	INY	qui passe à zéro au premier tour
E86C-	B9 10 CD	LDA CD10,Y	copie à la suite de la chaîne “tokenisée”, les 11 octets situés en CD10/CD1A: soit 00 00 01 01 FA BF 23 34 36 37 FF c'est à dire #00 de fin de ligne BASIC, 0100 adresse de la ligne suivante, FA01 n°de la ligne (ici 64001, normalement limité à 64000), CALL#467 et enfin #FF
E86F-	95 35	STA 35,X	et qui constitue une fausse ligne de commande:
E871-	E8	INX	CALL#467 avec #FF comme argument
E872-	C0 0A	CPY #0A	teste si 11 octets sont copiés
E874-	D0 F5	BNE E86B	sinon reboucle en E86B
E876-	A5 E9	LDA E9	
E878-	A4 EA	LDY EA	sauve la valeur actuelle de TXTPTR en C013/C014
E87A-	8D 13 C0	STA C013	
E87D-	8C 14 C0	STY C014	
E880-	A5 A8	LDA A8	
E882-	A4 A9	LDY A9	et celle du n° de ligne courante en C011/C012
E884-	8D 11 C0	STA C011	
E887-	8C 12 C0	STY C012	
E88A-	A9 34	LDA #34	
E88C-	A0 00	LDY #00	réinitialise TXTPTR avec l'adresse du TIB (0034)
E88E-	A2 3A	LDX #3A	
E890-	85 E9	STA E9	
E892-	84 EA	STY EA	
E894-	86 34	STX 34	place ":" en avant du TIB
E896-	88	DEY	
E897-	84 A9	STY A9	place #FF (mode immédiat) dans le n° de ligne
E899-	60	RTS	

L'interpréteur continue à TXTPTR, c'est à dire dans le TIB, exécute la chaîne, puis le CALL#467. Le sous-programme 0467 exécute la commande ! avec son argument #FF. Ce #FF est interprété comme le code d'une fonction: le sous-programme E83E qui restore les anciens TXTPTR et n° de ligne afin de reprendre le cours normal du programme, c'est à dire après la commande STRUN qui vient d'être exécutée.

E89A- 4C 77 E9 JMP E977 "STRING_TOO_LONG_ERROR"

EXÉCUTION DE LA COMMANDE SEDORIC TKEN

Rappel de la syntaxe

TKEN variable_alphanumérique

Cette commande “tokenise”, en mode programme uniquement, la variable alphanumérique indiquée qui doit correspondre à une ligne de commande BASIC et/ou SEDORIC. La chaîne résultante, qui peut être utilisée par STRUN, est identique à la chaîne source à ceci près que chaque mot-clé BASIC a été remplacé par le token correspondant. La longueur de la chaîne résultante est donc inférieure à celle de la chaîne de départ, qui doit compter au maximum 78 caractères. Il y a là une bogue car comme l'indique le manuel la limite devrait être 79 (taille du TIB). Il faudrait mettre un #50 en E8A7.

Non documenté

Les commandes STRUN et TKEN sont les deux seules commandes SEDORIC pour lesquelles le mode direct est interdit.

Attention, ce que le manuel ne dit pas, c'est que la chaîne résultante doit avoir moins de 68 caractères (taille du TIB moins les 11 octets de restauration de TXTPTR et de n° de ligne courante utilisés par STRUN). Plus ou moins par hasard, ces 11 octets sont gagnés par la “tokenisation”, avec des ratés si la chaîne ne comporte pas assez de mots-clés BASIC!

Pas de chance, le manuel donne en exemple pour TKEN une utilisation interdite: le mode immédiat!

Re-pas de chance, le manuel indique qu'il faut utiliser une variable alphanumérique et donne ensuite un exemple avec une constante alphanumérique, ce qui ne rend pas très claire l'utilisation de TKEN et donc de STRUN.

E89D-	20 5C D2	JSR D25C	JSR D4D2/ROM interdit le mode direct
E8A0-	20 38 D2	JSR D238	JSR D188/ROM décode le nom de la variable à TXTPTR (c'est à dire juste après la commande TKEN) et place "l'adresse" de cette variable dans AY, B6/B7 et D3/D4, cette adresse pointe en fait sur les data (longueur/adresse dans le cas d'une chaîne) de la variable dans la zone des variables BASIC
E8A3-	20 74 D2	JSR D274	JSR D7D0 et CF09/ROM vérifie si l'expression évaluée à TXTPTR est bien alphanumérique, retourne l'adresse de la chaîne dans XY et 91/92 ainsi que sa longueur dans A
E8A6-	C9 4F	CMP #4F	vérifie si cette longueur est supérieure à 78
E8A8-	B0 F0	BCS E89A	si oui, "STRING_TOO_LONG_ERROR"
E8AA-	AA	TAX	X pointera sur le caractère suivant la chaîne
E8AB-	A8	TAY	longueur de la chaîne (index de 0 à longueur + 1)
E8AC-	B1 91	LDA (91),Y	lit un octet de la chaîne
E8AE-	99 35 00	STA 0035,Y	et le copie dans le TIB (tampon clavier)
E8B1-	88	DEY	précédent (opère par la fin)

E8B2-	10 F8	BPL E8AC	reboucle en E8AC tant qu'il en reste à copier. En fait l'octet qui est copié en trop sera écrasé par le 0 de fin d'instruction
E8B4-	C8	INY	passe à zéro
E8B5-	94 35	STY 35,X	écrit ce #00 à la fin de la chaîne
E8B7-	A5 E9	LDA E9	
E8B9-	48	PHA	empile LL de TXTPTR
E8BA-	A9 35	LDA #35	
E8BC-	85 E9	STA E9	et le remplace par celui du TIB
E8BE-	20 94 D1	JSR D194	JSR C5FA/ROM encode les mots-clés BASIC de la chaîne présente à TXTPTR. La chaîne produite comporte à la fin 5 octets supplémentaires (0 final, lien et n° de ligne), qu'il faudra retirer. Ces octets sont en réserve pour amorcer la structure de la ligne BASIC suivante.
E8C1-	68	PLA	on récupère TXTPTR, car la commande TKEN ne fonctionne pas en mode direct (puisque le tampon clavier est entièrement modifié par l'action de TKEN). Il est donc primordial de savoir où on se trouve dans le texte BASIC...
E8C2-	85 E9	STA E9	restaure le LL de l'ancien TXTPTR. On se demande comment le HH de TXTPTR est géré et pourtant ça marche!
E8C4-	98	TYA	longueur totale de la chaîne “tokenisée” produite
E8C5-	38	SEC	prépare une soustraction
E8C6-	E9 05	SBC #05	longueur utile de la chaîne “tokenisée”
E8C8-	20 64 D2	JSR D264	JSR D5AB/ROM réserve une place en mémoire pour la chaîne de longueur A, retourne avec cette longueur en D0 et l'adresse de la chaîne en D1/D2
E8CB-	A4 D0	LDY D0	longueur de la chaîne
E8CD-	88	DEY	pour copier les caractères correspondants
E8CE-	B9 35 00	LDA 0035,Y	lit un octet de la chaîne dans le TIB
E8D1-	91 D1	STA (D1),Y	et le copie à l'emplacement réserve en mémoire
E8D3-	88	DEY	précédent (opère par la fin)
E8D4-	10 F8	BPL E8CE	reboucle en E8CE tant qu'il en reste

Il y a ici une bogue potentielle, car au moins un caractère est écrit, même si la longueur de la chaîne set nulle. L'octet lu en 0035 + FF = 0134 sera écrit dans la zone des chaînes et écrasera un octet d'une autre chaîne.

XMAJSTR copie l'adresse et la longueur d’une chaîne alphanumérique (les 3 octets D0, D1 et D2) en B6, B7 et B8

E8D6-	A0 02	LDY #02	
E8D8-	B9 D0 00	LDA 00D0,Y	copie longueur et adresse (3 octets)
E8DB-	91 B6	STA (B6),Y	dans la table des variables
E8DD-	88	DEY	
E8DE-	10 F8	BPL E8D8	
E8E0-	60	RTS	

EXÉCUTION DE LA COMMANDE SEDORIC UNTKEN

Rappel de la syntaxe

UNTKEN variable_alphanumérique

La chaîne indiquée, qui en principe contient un ou des token(s), est convertie en clair, c'est à dire que chaque token est remplacé par le mot-clé correspondant. La longueur de la chaîne produite est donc probablement supérieure à celle de la chaîne traitée. Toutefois, pour obtenir une "STRING_TOO_LONG_ERROR" (chaîne > 255 octets), il faudra le faire exprès!

Non ou mal documenté

L'exemple donné dans le manuel n'est pas particulièrement clair. En particulier, l'utilité de la commande UNTKEN n'est pas évidente. En voici pourtant un exemple: faire un petit éditeur de programme BASIC plein écran, capable de décoder et de re-coder (TKEN) les mots-clés des lignes de commandes (à condition d'intégrer UNTKEN et TKEN dans un programme en LM). Mais les bricoleurs de génie, lui trouveront d'autres applications!

Analyse de syntaxe et saisie du paramètre

E8E1-	20 38 D2	JSR D238	JSR D188/ROM décode le nom de la variable à TXTPTR et place "l'adresse" de cette variable dans AY, B6/B7 et D3/D4, cette adresse pointe en fait sur les data (longueur/adresse dans le cas d'une chaîne) de la variable dans la zone des variables BASIC
E8E4-	20 74 D2	JSR D274	JSR D7D0 et CF09/ROM vérifie si l'expression évaluée à TXTPTR est bien alphanumérique, retourne l'adresse de la chaîne dans XY et 91/92 ainsi que sa longueur dans A
E8E7-	AA	TAX	teste si la chaîne est vide (longueur nulle)
E8E8-	F0 F6	BEQ E8E0	si oui simple RTS en E8E0
E8EA-	85 F3	STA F3	sinon, sauve la longueur de la chaîne
E8EC-	A2 00	LDX #00	initialise la longueur de la nouvelle chaîne
E8EE-	A0 00	LDY #00	index de lecture

Parcourt la chaîne source à la recherche des tokens

E8F0-	A9 E9	LDA #E9	
E8F2-	85 16	STA 16	16/17 = début de la table des mots-clés
E8F4-	A9 C0	LDA #C0	(en C0EA/ROM)
E8F6-	85 17	STA 17	
E8F8-	84 F2	STY F2	sauve l'index de lecture
E8FA-	B1 91	LDA (91),Y	lit un octet de la chaîne source
E8FC-	10 2B	BPL E929	ce n'est pas un token, suite en E929

Un token a été trouvé

E8FE-	29 7F	AND #7F	en force le b7 à 0 ce qui donne son n° d'ordre
E900-	F0 13	BEQ E915	continue en E915 si c'était #80 (n° d'ordre #00)
E902-	85 26	STA 26	sinon sauve le n° d'ordre du mot-clé
E904-	A0 00	LDY #00	initialise Y pour le sous-programme 0453 ci-après

Parcourt la table des mots-clés jusqu'au n° d'ordre voulu

E906-	E6 16	INC 16	
E908-	D0 02	BNE E90C	incrémente le pointeur dans la table des mots-clés
E90A-	E6 17	INC 17	
E90C-	20 53 04	JSR 0453	lit le caractère du mot-clé dans la table en ROM
E90F-	10 F5	BPL E906	ce n'est pas la fin du mot-clé, reboucle en E906
E911-	C6 26	DEC 26	fin du mot-clé atteinte, décrémente le compteur de mots-clés (n° d'ordre du mot-clé dans la table)
E913-	D0 F1	BNE E906	ce n'est pas le bon, reboucle en E906

Le bon mot-clé a été trouvé dans la table

E915-	A0 01	LDY #01	c'est le bon, indexe le début du mot-clé
E917-	E8	INX	longueur de la nouvelle chaîne (index d'écriture)
E918-	F0 1E	BEQ E938	"STRING_TOO_LONG_ERROR" si longueur > # 100 caractères
E91A-	20 53 04	JSR 0453	lit un caractère du mot-clé dans la table en ROM
E91D-	08	PHP	sauvegarde les indicateurs 6502 dont N
E91E-	29 7F	AND #7F	en force le b7 à 0, même si ce n'est pas le dernier
E920-	9D FF C0	STA C0FF,X	et le copie dans BUF1
E923-	C8	INY	vise le caractère suivant du mot-clé
E924-	28	PLP	récupère les indicateurs dont N
E925-	10 F0	BPL E917	reboucle en E917 tant que la fin n'est pas atteinte
E927-	30 06	BMI E92F	continue en E92F si le dernier caractère a été traité

Ce n'était pas un token

E929-	E8	INX	longueur de la nouvelle chaîne
E92A-	F0 0C	BEQ E938	"STRING_TOO_LONG_ERROR" si longueur > # 100 caractères
E92C-	9D FF C0	STA C0FF,X	copie cet octet "non token" dans BUF1

Octet suivant de la chaîne source

E92F-	C6 F3	DEC F3	décrémente la longueur de chaîne source
E931-	F0 08	BEQ E93B	suite en E93B si tous les octets ont été traités
E933-	A4 F2	LDY F2	sinon, récupère l'index de lecture
E935-	C8	INY	et l'incrémente
E936-	D0 B8	BNE E8F0	reprise forcée en E8F0 tant que longueur < #100
E938-	4C 77 E9	JMP E977	sinon, "STRING_TOO_LONG_ERROR"

Tous les octets de la chaîne source ont été traités

E93B-	8A	TXA	nouvelle longueur de la chaîne
E93C-	20 64 D2	JSR D264	JSR D5AB/ROM réserve une place en mémoire pour la chaîne de longueur A, retourne avec cette longueur en D0 et l'adresse de la chaîne en D1/D2
E93F-	A4 D0	LDY D0	longueur de la nouvelle chaîne
E941-	88	DEY	index de copie
E942-	B9 00 C1	LDA C100,Y	lit un octet de BUF1
E945-	91 D1	STA (D1),Y	et le copie dans l'emplacement réservé en ROM

E947-	98	TYA	teste si Y atteint 0 (terminé)
E948-	D0 F7	BNE E941	reboucle en E941 si ce n'est pas le cas
E94A-	4C D6 E8	JMP E8D6	suite en E8D6 si terminé (XMAJSTR copie l'adresse et la longueur d'une chaîne alphanumérique (les 3 octets D0, D1 et D2) en B6, B7 et B8)

XSETOFF teste si le paramètre à TXTPTR est SET ou OFF, retourne avec C = 1 si SET ou C = 0 si OFF sinon génère une "SYNTAX_ERROR"
(code appelé par les commandes ACCENT, ERR, KEY et PR)

E94D-	A0 02	LDY #02	pour tester les 3 caractères de "SET"
E94F-	B1 E9	LDA (E9),Y	lit un caractère à TXTPTR
E951-	29 DF	AND #DF	1101 1111 force le b5 à zéro (conversion minuscule -> MAJUSCULE)
E953-	D9 2E CD	CMP CD2E,Y	compare avec caractère de la table CD2E/CD30 ("SET")
E956-	D0 05	BNE E95D	continue en E95D si différent
E958-	88	DEY	indexe le caractère précédant (travaille par la fin)
E959-	10 F4	BPL E94F	et reboucle en E94F tant que Y n'est pas négatif
E95B-	30 0F	BMI E96C	suite forcée en E96C (SET à été trouvé et C = 1)
E95D-	A0 02	LDY #02	pour tester les 3 caractères de "OFF"
E95F-	B1 E9	LDA (E9),Y	lit un caractère à TXTPTR
E961-	29 DF	AND #DF	1101 1111 force le b5 à zéro (conversion minuscule -> MAJUSCULE)
E963-	D9 2B CD	CMP CD2B,Y	compare avec caractère de la table CD2B/CD2D ("OFF")
E966-	D0 0C	BNE E974	"SYNTAX_ERROR" (autre paramètre impossible)
E968-	88	DEY	indexe le caractère précédant (travaille par la fin)
E969-	10 F4	BPL E95F	et reboucle en E95F tant que Y n'est pas négatif
E96B-	18	CLC	C = 0 si OFF trouvé
E96C-	08	PHP	sauvegarde les indicateurs 6502 dont C
E96D-	A0 03	LDY #03	pour sauter les trois caractères lus à TXTPTR
E96F-	20 E3 D1	JSR D1E3	JSR CA3F/ROM met à jour TXTPTR en ajoutant Y
E972-	28	PLP	récupère les indicateurs dont C
E973-	60	RTS	
E974-	4C 23 DE	JMP DE23	"SYNTAX_ERROR"
E977-	A2 12	LDX #12	pour "STRING_TOO_LONG_ERROR"
E979-	4C 7E D6	JMP D67E	incrémente X et traite l'erreur n° X
E97C-	4C 20 DE	JMP DE20	"ILLEGAL_QUANTITY_ERROR"

EXÉCUTION DE LA COMMANDE SEDORIC ERR

Rappel de la syntaxe

ERR SET ou **ERR OFF**

Lorsqu'une erreur concernant SEDORIC se produit (par exemple "INVALID_FILE_NAME_ERROR"), le programme s'arrête: par défaut SEDORIC travaille en ERR OFF, c'est à dire que la gestion des erreurs est OFF et le programme doit être arrêté, car il ne peut se débrouiller tout seul. Mais il est possible de

prévoir certaines erreurs et de gérer par programme leur traitement: dans ce cas il faudra utiliser un ERR SET. Les erreurs BASIC ne sont pas concernées. La liste des 20 erreurs SEDORIC est données dans le manuel (page 96) et dans la table CDBF/CEE6. Il est possible de définir des erreurs supplémentaires dont le n° peut aller de 50 à 255 (voir la commande ERROR).

Lorsqu'un ERR SET est en cours et qu'une erreur est rencontrée, le programme continue à la ligne spécifiée par la commande ERRGOTO, si elle existe, sinon il s'arrête et génère soit un des 20 messages d'erreur SEDORIC, soit "USER_x_ERROR" (x étant le n° de l'erreur utilisateur). Il est donc indispensable d'avoir un ERRGOTO après chaque ERR SET.

Dans tous les cas (ERR SET ou OFF), lorsqu'une erreur est rencontrée, les variables EN (n° de l'erreur) et EL (n° de la ligne dont l'exécution a produit l'erreur) sont mises à jour. En mode direct le n° de ligne est #FFFF (65535 et non 65635 comme indiqué dans le manuel (bogue))!

Non documenté

Attention, par défaut ERR SET et ERR OFF remettent à zéro le n° de ligne ERRGOTO et valident l'affichage du message d'erreur en guise de traitement.

Saisie du paramètre, analyse de syntaxe et exécution de la commande

E97F-	20 4D E9	JSR E94D	XSETOFF teste si le paramètre à TXTPTR est SET ou OFF, retourne avec C = 1 si SET ou C = 0 si OFF sinon génère une "SYNTAX_ERROR"
E982-	A9 00	LDA #00	
E984-	8D 1C C0	STA C01C	mise à zéro de l'adresse ERRGOTO
E987-	8D 1B C0	STA C01B	(n° de la ligne de gestion des erreurs)
E98A-	6A	ROR	A = #00 si "OFF" et #80 si "SET" par injection de C dans le b7 de #00
E98B-	8D 18 C0	STA C018	flag ERR (b7 à 1 si "SET")
E98E-	A0 37	LDY #37	
E990-	A2 FF	LDX #FF	C019/C01A reçoit l'adresse FF37
E992-	8C 19 C0	STY C019	(adresse par défaut où ajuster TXTPTR pour traiter l'erreur,
E995-	8E 1A C0	STX C01A	c'est à dire simple affichage du message d'erreur)
E998-	60	RTS	

EXÉCUTION DE LA COMMANDE SEDORIC ERRGOTO

Rappel de la syntaxe

ERRGOTO n°_de_ligne

Lorsqu'un ERR SET est en cours et qu'une erreur est rencontrée, le programme continue à la ligne spécifiée par la commande ERRGOTO, si elle existe, sinon il s'arrête et génère soit un des 20 messages d'erreur SEDORIC, soit "USER_x_ERROR" (x étant le n° de l'erreur utilisateur). Il est donc indispensable d'avoir un ERRGOTO après chaque ERR SET.
Il suffit de consulter la variable EN pour savoir de quelle erreur il s'agit et déclencher ou demander une correction spécifique.

Saisie du paramètre et exécution de la commande

E999-	20 FA D2	JSR D2FA	E853/ROM évalue un nombre non signé à TXTPTR et revient avec ce nombre dans YA, 33/34 et D3/D4 et mise à jour de TXTPTR sur l'octet suivant ce nombre (C = 0, N et Z positionnés selon l'octet de poids faible)
E99C-	8D 1C C0	STA C01C	et l'écrit en C01B/C01C
E99F-	8C 1B C0	STY C01B	(n° de la ligne de gestion des erreurs)
E9A2-	20 9C D1	JSR D19C	JSR C6B3/ROM cherche l'adresse de cette ligne BASIC
E9A5-	A6 CF	LDX CF	
E9A7-	A4 CE	LDY CE	YX = adresse de cette ligne
E9A9-	D0 01	BNE E9AC	calcule YX = YX - #01
E9AB-	CA	DEX	
E9AC-	88	DEY	
E9AD-	4C 92 E9	JMP E992	copie YX dans C019/C01A (adresse où ajuster TXTPTR)

EXÉCUTION DE LA COMMANDE SEDORIC ERROR

Rappel de la syntaxe

ERROR n°_d'erreur

Lorsqu'elle est rencontrée, cette commande simule la survenue de l'erreur dont le n° est indiqué (de 50 à 255). En supposant que le programme soit bien conçu, il continuera à la ligne spécifiée par la commande ERRGOTO. Ceci permet de centraliser la gestion de "toutes" les erreurs (SEDORIC et utilisateur, mais pas BASIC) dans le même sous-programme. Si aucun ERR SET n'est en cours, une "USER_x_ERROR" est générée (x étant le n° de l'erreur). Ceci est très utile pour déboguer un programme.

Saisie et vérification du paramètre et exécution de la commande

E9B0-	20 7F D2	JSR D27F	CF17/ROM et CF09/ROM et D8CB/ROM évalue un nombre entier à TXTPTR et le retourne dans X (numéro de l'erreur utilisateur)
E9B3-	E0 32	CPX #32	teste si X < 50 (c'est à dire si erreur SEDORIC)
E9B5-	90 C5	BCC E97C	si oui, "ILLEGAL_QUANTITY_ERROR" (50 à 255 SVP!)
E9B7-	CA	DEX	pour neutraliser l'incrémentation qui suit
E9B8-	4C 7E D6	JMP D67E	incrémente X et traite l'erreur n° X

EXÉCUTION DE LA COMMANDE SEDORIC RESUME

Rappel de la syntaxe

RESUME ou **RESUME NEXT**

A la fin du traitement de l'erreur par le sous-programme indiqué avec la commande ERRGOTO, il est possible de reprendre le cours normal du programme là où il avait été interrompu avec la commande RESUME (la cause de l'erreur ayant été corrigée entre temps) ou à la commande suivante avec la

commande RESUME NEXT (la cause de l'erreur ne pouvant pas être corrigée). Dans les deux cas le contexte est identique, mis à part ce qui a été effectué par le sous-programme de gestion des erreurs.

Non documenté

Bogue dans le manuel: avec RESUME NEXT, l'exécution est reprise non pas à la ligne suivante, mais à la commande qui suit celle qui a provoqué l'erreur!

Enfin, toujours le même problème avec l'utilisation possible des minuscules pour entrer les commandes SEDORIC: ça ne marche pas à tous les coups (bogue). Ici, vous avez intérêt à taper "NEXT" et non "next" si vous ne voulez pas écoper d'une belle "SYNTAX_ERROR"!

Analyse de la syntaxe

E9BB-	F0 06	BEQ E9C3	suite en E9C3 si pas de paramètre (avec Z = 1, flag RESUME tout court)
E9BD-	A9 90	LDA #90	A = token "NEXT"
E9BF-	20 2E D2	JSR D22E	JSR D067/ROM puis D3A1/RAM overlay demande un "NEXT" à TXTPTR, lit le caractère suivant et le convertit en MAJUSCULE
E9C2-	C8	INY	Y a été mis à zéro dans le sous-programme ci-dessus et passe ici à #01 pour mettre Z à zéro (flag "NEXT" présent) (un peu tortueux!)

Re-initialise TXTPTR et n° de ligne

E9C3-	08	PHP	sauvegarde les indicateurs 6502 dont Z
E9C4-	AD 21 C0	LDA C021	
E9C7-	AC 22 C0	LDY C022	AY = point où l'erreur s'est produite
E9CA-	85 E9	STA E9	
E9CC-	84 EA	STY EA	remet en place TXTPTR au début de la commande fautive
E9CE-	AD FE 04	LDA 04FE	
E9D1-	AC FF 04	LDY 04FF	AY = n° de ligne de l'erreur
E9D4-	85 A8	STA A8	
E9D6-	84 A9	STY A9	remet en place le n° de ligne (s'il a changé)
E9D8-	28	PLP	récupère les indicateurs 6502 dont Z

Teste si RESUME ou RESUME NEXT

E9D9-	F0 03	BEQ E9DE	saute l'instruction suivante si Z = 1, (RESUME tout court) et reprend l'exécution là où elle avait été interrompue

Rajuste TXTPTR sur l'instruction suivante

E9DB-	4C DC D1	JMP D1DC	si Z = 0, "NEXT" a été trouvé: JSR CA4E/ROM qui calcule le déplacement Y à l'instruction suivante, JSR CA3F/ROM qui met à jour TXTPTR en y ajoutant Y et enfin, retour à l'interpréteur

Rajuste TXTPTR sur l'instruction ayant causé l'erreur

E9DE-	C6 EA	DEC EA	décrémente TXTPTR de #100

E9E0- A0 FF LDY #FF indexe #FF octets plus loin
E9E2- B1 E9 LDA (E9),Y lit le caractère à TXTPTR - #100 + #FF = -1
E9E4- C9 3A CMP #3A est-ce un ":"?

Bogue: cette procédure est dangereuse car la valeur #3A peut ainsi être le HH d'un n° de ligne et alors bonjour le plantage...

E9E6- F0 02 BEQ E9EA si oui, continue en E9EA (recule de 1 octet)
E9E8- A0 FB LDY #FB sinon Y = #FB, recule de 5 octets (pour l'en-tête de ligne)
E9EA- 4C E3 D1 JMP D1E3 CA3F/ROM met à jour TXTPTR en ajoutant Y et retourne

De E9ED à EA3A, se trouvait la commande EXT qui a été déplacée dans la BANQUE n°7.

L'entrée de la commande EXT se fait toujours à la même adresse, mais elle est redirigée sur la BANQUE n°7. Le reste de la zone précédemment occupée par la commande EXT a été ré-utilisée pour implémenter les entrées des commandes STATUS, PROT, UNPROT, et SYSTEM (qui ont été également déplacées dans la BANQUE n°7), ainsi que celles des deux nouvelles commandes CHKSUM et VISUHIRES. J'ai également utilisé une partie de cet espace pour implanter deux sous-programmes destinés à corriger les bogues "LOVE" et "LINPUT". Il reste encore un peu de place (42 octets) pour implanter de nouvelles commandes ou des routines de débogage. Etant donné l'ampleur des changements, il n'est pas surprenant de trouver 75 octets différents sur les 78 octets de cette zone.

EXÉCUTION DE LA COMMANDE SEDORIC EXT

E9ED- **A0 03** LDY #03 LL de adr-1 entrée de **EXT** dans la BANQUE n°7, suite en EA01

EXÉCUTION DE LA COMMANDE SEDORIC VISUHIRES

E9EF- **2C A0 51** BIT 51A0 cache en E9F0
E9F0- A0 51 LDY #51 l'entrée de **VISUHIRES** dans BANQUE n°7

EXÉCUTION DE LA COMMANDE SEDORIC STATUS

E9F2- **2C A0 54** BIT 54A0 cache en E9F3
E9F3- A0 54 LDY #54 l'entrée de **STATUS** dans BANQUE n°7

EXÉCUTION DE LA COMMANDE SEDORIC PROT

E9F5- **2C A0 57** BIT 57A0 cache en E9F6
E9F6- A0 57 LDY #57 l'entrée de **PROT** dans BANQUE n°7

EXÉCUTION DE LA COMMANDE SEDORIC UNPROT

E9F8- **2C A0 5A** BIT 5AA0 cache en E9F9
E9F9- A0 5A LDY #5A l'entrée de **UNPROT** dans BANQUE n°7

EXÉCUTION DE LA COMMANDE SEDORIC SYSTEM

E9FB- **2C A0 5D** BIT 5DA0 cache en E9FC
E9FC- A0 5D LDY #5D l'entrée de **SYSTEM** dans BANQUE n°7

EXÉCUTION DE LA COMMANDE SEDORIC CHKSUM

E9FE- **2C** A0 **79** BIT 79A0 cache en E9FF
E9FF- A0 79 LDY #79 l'entrée de **CHKSUM** dans BANQUE n°7

EA01- **A2 60** LDX #60 la BANQUE n°7 se trouve au 96 ème secteur de la disquette Master
EA03- **4C 5E F1** JMP F15E routine de gestion des BANQUES

De EA06 à EA2F, suite de **42 NOP**s, espace libre pour implanter de nouvelles commandes.

Suite de la correction de la bogue "**LOVE**"
Ce greffon est appelé de D90A (Routine "Prendre un caractère au clavier")

EA30- **E8** INX la mauvaise gestion de l'index X était la cause de la bogue!
EA31- **86 F2** STX F2 restauration de 2 commandes supprimées
EA33- **A2 3F** LDX #3F pour placer le JSR EA30 en D90A (voir à cet endroit)
EA35- **60** RTS et voilà on peut retourner

Correction de la bogue **LINPUT**
Ce greffon est appelé de ECB5 (Routine "Gestion des coordonnées xy" de la commande LINPUT)

EA36- **86 30** STX 30 ce sont les deux octets qui manquaient dans la routine d'origine
EA38- 4C **3E D7** JMP D73E reprise du cours normal de la routine LINPUT avec XCURON rend le curseur visible (= vidéo inverse) puis RTS au point d'appel du greffon.

EXÉCUTION DE LA COMMANDE SEDORIC SWAP

Rappel de la syntaxe

SWAP variable_1,variable_2

Echange le contenu des 2 variables indiquées qui doivent évidemment être du même type.

EA3B-	20 38 D2	JSR D238	JSR D188/ROM décode le nom de la première variable à TXTPTR et place "l'adresse" de cette variable dans AY, B6/B7 et D3/D4, cette adresse pointe en fait sur les data (longueur/adresse dans le cas d'une chaîne) de la variable dans la zone des variables BASIC
EA3E-	85 B8	STA B8	
EA40-	84 B9	STY B9	sauve l'adresse de variable_1 en B8/B9
EA42-	A5 28	LDA 28	
EA44-	48	PHA	empile le type de la variable trouvée à TXTPTR
EA45-	A5 29	LDA 29	
EA47-	48	PHA	
EA48-	20 2C D2	JSR D22C	D067/ROM exige une "," lit le caractère suivant et le convertit en MAJUSCULE
EA4B-	20 38 D2	JSR D238	JSR D188/ROM décode le nom de la première variable à TXTPTR et place "l'adresse" de cette variable dans AY, B6/B7 et D3/D4, cette adresse pointe en fait sur les data (longueur/adresse dans le cas d'une chaîne) de la variable dans la zone des variables BASIC
EA4E-	85 91	STA 91	
EA50-	84 92	STY 92	sauve l'adresse de variable_2 en 91/92
EA52-	68	PLA	
EA53-	C5 29	CMP 29	compare le type des deux variables
EA55-	D0 20	BNE EA77	si différent, "TYPE_MISMATCH_ERROR"
EA57-	68	PLA	
EA58-	C5 28	CMP 28	
EA5A-	D0 1B	BNE EA77	

Calcule l'index Y en fonction du type de variable (entier, réel, chaîne)

EA5C-	A0 01	LDY #01	index par défaut pour copie si type "nombre entier"
EA5E-	24 28	BIT 28	teste si b7 de 28 à 1 (type "chaîne")
EA60-	30 06	BMI EA68	si oui, continue en EA68
EA62-	24 29	BIT 29	teste si b7 de 29 à 1 (type "nombre entier")
EA64-	30 03	BMI EA69	si oui, continue en EA69
EA66-	C8	INY	Y = nombre d'octets à copier selon type de variable
EA67-	C8	INY	2 octets en plus pour un nombre réel
EA68-	C8	INY	1 octet en plus pour un nombre réel ou une chaîne
EA69-	B1 91	LDA (91),Y	lit un octet de la variable 2
EA6B-	AA	TAX	et le place temporairement dans X
EA6C-	B1 B8	LDA (B8),Y	lit un octet de la variable 1 et le copie à
EA6E-	91 91	STA (91),Y	l'emplacement homologue de la variable 2
EA70-	8A	TXA	récupère l'octet de la variable 2 et le copie à
EA71-	91 B8	STA (B8),Y	l'emplacement homologue de la variable 1
EA73-	88	DEY	octet précédent
EA74-	10 F3	BPL EA69	reboucle en EA69 tant qu'il en reste
EA76-	60	RTS	(effectue donc Y + 1 tours)
EA77-	A2 0B	LDX #0B	pour "TYPE_MISMATCH_ERROR"
EA79-	4C 7E D6	JMP D67E	incrémente X et traite l'erreur n° X

EA7C- 4C 20 DE JMP DE20 "ILLEGAL_QUANTITY_ERROR"

EXÉCUTION DE LA COMMANDE SEDORIC USER

Rappel de la syntaxe

USER n°_de_routine,DEF adresse (,O) ou
USER n°_de_routine(,A valeur)(,X valeur)(,Y valeur)(,P valeur)

Cette commande est très intéressante, mais elle est très peu utilisée, car le manuel n'explique pas assez clairement son mode d'emploi. Elle permet d'appeler des routines en "Langage Machine" avec passage et retour de paramètres. Dans les 2 cas le numéro de la routine indiqué est une expression numérique de 0 à 3 (il est donc possible d'utiliser 4 sous-programmes en "Langage Machine" au maximum avec la commande USER).

La première forme permet de définir l'adresse d'exécution de la routine. Le paramètre ",O" sert à indiquer que cette routine se trouve en RAM overlay ("O" pour Overlay). Très intéressant, on peut donc placer une routine dans une zone libre de la RAM overlay et l'exécuter, ce qu'un CALL ne peut pas faire.

La seconde forme exécute la routine préalablement définie et permet en plus de charger, avant l'exécution de la routine, les registres A, X, Y et P avec les valeurs indiquées. Chacune de ces valeurs, qui peut être n'importe quelle expression numérique de 0 à 255, doit bien entendu être possible selon l'usage que l'on veut en faire. Au retour les variables RA, RX, RY et RP contiennent les valeurs des registres A, X, Y et P. Cette commande est donc très puissante, voyez ci-après.

Non ou mal documenté

Bien que cela ne soit pas clairement indiqué dans le manuel, il n'est pas possible de mélanger les paramètres de la première et de la seconde forme. En effet, aucun autre paramètre que ",O" n'est possible après DEF et la première forme se termine par un simple RTS. Au contraire, la seconde forme se termine par un appel à la routine dont l'adresse doit avoir été préalablement définie.

Le manuel comporte une bogue dans la syntaxe indiquée: la virgule est indispensable devant DEF (les exemples donnés sont eux corrects), sinon SEDORIC considère alors qu'il s'agit d'un paramètre utilisateur.

En effet, comme la commande CALL du BASIC, la deuxième forme de la commande USER (mais attention, pas la première) peut être suivie d'autant de paramètres que l'on veut (paramètres utilisateur). Les seules différences avec CALL sont: 1) que l'on peut travailler directement en RAM overlay et 2) que l'on peut directement affecter les registres du 6502. L'analyse de syntaxe de la commande USER s'arrête dès que le paramètre rencontré n'est pas un des paramètres prévus par SEDORIC. Le système ne génère pas d'erreur, mais considère qu'il s'agit alors d'un paramètre utilisateur. Attention dans la syntaxe de vos paramètres personnels! C'est évidemment la routine utilisateur qui doit se charger de l'analyse de la syntaxe des paramètres utilisateurs présents à TXTPTR.

Enfin, toujours le même problème avec l'utilisation possible des minuscules pour entrer les commandes SEDORIC: ça ne marche pas à tous les coups (bogue). Ici, le "DEF " et le ",O" doivent être tapés en MAJUSCULES, par contre "A", "X", "Y ou "P" sont acceptés en minuscule! Cela dépend de la routine qui

lit à TXTPTR avec ou sans conversion en MAJUSCULE.

Analyse de la syntaxe et saisie des paramètres

EA7F-	20 7F D2	JSR D27F	CF17/ROM et CF09/ROM et D8CB/ROM évalue un nombre entier à TXTPTR et le retourne dans X (numéro de la routine utilisateur)
EA82-	8A	TXA	le n° de routine (0 à 3) passe dans A
EA83-	C9 04	CMP #04	teste si > 3
EA85-	B0 F5	BCS EA7C	si oui, "ILLEGAL_QUANTITY_ERROR"
EA87-	0A	ASL	ce n° est multiplié par 2 puis on ajoute sa value
EA88-	65 D4	ADC D4	initiale (présente dans ACC1) au résultat
EA8A-	85 F6	STA F6	sauve la valeur triple (0 à 9) pour faire un index
EA8C-	AA	TAX	qui est immédiatement utilisé dans X
EA8D-	BD 68 C0	LDA C068,X	sauve dans F7 l'ancienne valeur du flag ",O" correspondant
EA90-	85 F7	STA F7	à ce n° de routine (en vue d'une exécution éventuelle)
EA92-	A9 00	LDA #00	
EA94-	A2 03	LDX #03	force F2, F3, F4 et F5 à zéro
EA96-	95 F2	STA F2,X	(pour les valeurs des registres A, X, Y et P)
EA98-	CA	DEX	
EA99-	10 FB	BPL EA96	reboucle en EA96 tant que ce n'est pas terminé

Suite de l'analyse de syntaxe

EA9B-	20 9E D3	JSR D39E	XCRGOT relit le caractère à TXTPTR (sans incrémenter TXTPTR = CHRGOT), puis le convertit en MAJUSCULE, les espaces sont sautés, Z = 1 si fin d'instruction (0 ou :), C = 0 si caractère chiffre 0 à 9 (soit #30 à #39), sinon C = 1, Y et X inchangés
EA9E-	C9 2C	CMP #2C	le caractère à TXTPTR est-il une virgule?
EAA0-	D0 46	BNE EAE8	sinon, continue en EAE8 (exécution de la routine indiquée)
EAA2-	20 98 D3	JSR D398	XCRGET lit le caractère suivant à TXTPTR et le convertit en MAJUSCULE
EAA5-	A0 04	LDY #04	doit être un des 5 cas suivants: A, X, Y, P ou DEF
EAA7-	D9 83 CD	CMP CD83,Y	compare avec le contenu de la table CD83/CD87
EAAA-	F0 05	BEQ EAB1	si trouvé, continue en EAB1 avec Y positionné
EAAC-	88	DEY	sinon, vise l'octet précédent de la table
EAAD-	10 F8	BPL EAA7	reboucle tant qu'il en reste à examiner
EAAF-	30 37	BMI EAE8	rien trouvé, continue en EAE8 (exécution de la routine indiquée) (le paramètre n'est ni une valeur de registre, ni DEF, mais un paramètre utilisateur, ou une erreur de syntaxe!)

Traite le code trouvé (A, Y, X, P ou DEF)

EAB1-	20 98 D3	JSR D398	XCRGET incrémente TXTPTR, lit un caractère (CHRGET), les espaces sont sautés, le met dans A, le convertit en MAJUSCULE, Z = 1 si fin d'instruction (0 ou :), C = 0 si chiffre 0 à 9 (soit #30 à #39), sinon C =1. Y et X inchangés (en fait place TXTPTR sur le caractère suivant)
EAB4-	C0 04	CPY #04	teste l'index Y: était-ce le token DEF?
EAB6-	D0 22	BNE EADA	sinon, continue en EADA

Traite le token DEF

EAB8-	20 FA D2	JSR D2FA	E853/ROM évalue un nombre non signé à TXTPTR et revient avec ce nombre dans YA, 33/34 et D3/D4 et mise à jour de TXTPTR sur l'octet suivant ce nombre (C = 0, N et Z positionnés selon l'octet de poids faible) (adresse d'exécution de la routine utilisateur)
EABB-	A6 F6	LDX F6	récupère l'index sauvé en EA8A (valant de 0 à 9)
EABD-	9D 67 C0	STA C067,X	écrit HH du nombre évalué dans la table C066/C071
EAC0-	98	TYA	idem avec LL (place l'adresse d'exécution de la
EAC1-	9D 66 C0	STA C066,X	routine dans la table des adresses C066/C071)
EAC4-	20 9E D3	JSR D39E	XCRGOT relit le caractère à TXTPTR (sans incrémenter TXTPTR = CHRGOT), puis le convertit en MAJUSCULE, les espaces sont sautés, Z = 1 si fin d'instruction (0 ou :), C = 0 si caractère chiffre 0 à 9 (soit #30 à #39), sinon C = 1, Y et X inchangés
EAC7-	F0 0A	BEQ EAD3	fin de commande (il n'y a plus de paramètre), continue en EAD3 avec A = #00, flag positif qui indique par défaut que cette routine se trouve en ROM et/ou lower RAM
EAC9-	20 2C D2	JSR D22C	D067/ROM exige une "," lit le caractère suivant et le convertit en MAJUSCULE
EACC-	A9 4F	LDA #4F	caractère "O" MAJUSCULE (toujours le même problème)
EACE-	20 2E D2	JSR D22E	JSR D067/ROM puis D3A1/RAM overlay demande un "O" à TXTPTR, lit le caractère suivant et le convertit éventuellement en MAJUSCULE
EAD1-	A2 80	LDX #80	flag négatif qui indique que cette routine se
EAD3-	8A	TXA	trouve en RAM overlay et/ou lower RAM
EAD4-	A6 F6	LDX F6	récupère l'index sauvé en EA8A (valant de 0 à 9)
EAD6-	9D 68 C0	STA C068,X	écrit le flag ",O" à la suite de l'adresse d'exécution dans la table des adresses C066/C071
EAD9-	60	RTS	

Suite: traite le code trouvé (A, Y, X, P)

EADA-	98	TYA	index Y significatif du code trouvé (A, Y, X ou P)
EADB-	48	PHA	empile cet index
EADC-	20 7F D2	JSR D27F	CF17/ROM et CF09/ROM et D8CB/ROM évalue un nombre entier à TXTPTR et le retourne dans X (valeur pour le registre A, Y, X ou P)
EADF-	68	PLA	récupère index
EAE0-	A8	TAY	et le passe dans Y
EAE1-	96 F2	STX F2,Y	place la valeur évaluée dans F2, F3, F4 ou F5 selon qu'elle concerne le registre A, Y, X ou P
EAE3-	20 9E D3	JSR D39E	XCRGOT relit le caractère à TXTPTR (sans incrémenter TXTPTR = CHRGOT), puis le convertit en MAJUSCULE, les espaces sont sautés, Z = 1 si fin d'instruction (0 ou :), C = 0 si caractère chiffre 0 à 9 (soit #30 à #39), sinon C = 1, Y et X inchangés, en fait, ce JSR sert uniquement à forcer la reprise en EA9B s'il y a encore des paramètres et chose curieuse, en EA9B se trouve un autre JSR D39E!
EAE6-	D0 B3	BNE EA9B	reboucle en EA9B pour traiter le paramètre suivant

Exécution de la routine

EAE8-	A4 F4	LDY F4	valeur indiquée pour le registre Y
EAEA-	A5 F5	LDA F5	valeur indiquée pour le registre P
EAEC-	48	PHA	empile P
EAED-	A6 F6	LDX F6	récupère l'index sauvé en EA8A (valant de 0 à 9)
EAEF-	BD 66 C0	LDA C066,X	
EAF2-	8D F0 04	STA 04F0	copie l'adresse d'exécution dans EXEVEC
EAF5-	BD 67 C0	LDA C067,X	en page 4
EAF8-	8D F1 04	STA 04F1	
EAFB-	A5 F2	LDA F2	valeur indiquée pour le registre A
EAFD-	A6 F3	LDX F3	valeur indiquée pour le registre X
EAFF-	24 F7	BIT F7	teste si la routine se trouve en ROM
EB01-	10 07	BPL EB0A	si oui, continue en EB0A

Exécution en RAM overlay (et/ou RAM < C000)

EB03-	28	PLP	sinon, récupère les indicateurs
EB04-	20 22 EB	JSR EB22	appelle la routine en RAM overlay selon EXEVEC
EB07-	4C 0E EB	JMP EB0E	suite et fin en EB0E (mise à jour des variables)

Exécution en ROM (et/ou RAM < C000)

EB0A-	28	PLP	récupère les indicateurs
EB0B-	20 71 04	JSR 0471	appelle une routine en ROM selon EXEVEC

Met à jour les variables RA, RX, RY et RP

EB0E-	48	PHA	sauvegarde le registre A
EB0F-	08	PHP	sauvegarde le registre P
EB10-	8A	TXA	
EB11-	48	PHA	sauvegarde le registre X
EB12-	98	TYA	
EB13-	20 CF D7	JSR D7CF	sauvegarde le registre Y dans la variable RY
EB16-	68	PLA	récupère X
EB17-	20 CC D7	JSR D7CC	sauvegarde le registre X dans la variable RX
EB1A-	68	PLA	récupère P
EB1B-	20 D2 D7	JSR D7D2	sauvegarde le registre P dans la variable RP
EB1E-	68	PLA	récupère A
EB1F-	4C C9 D7	JMP D7C9	sauvegarde le registre A dans la variable RA

EB22-	6C F0 04	JMP (04F0)	appelle la routine en RAM overlay (et/ou lower RAM)

EXÉCUTION DE LA COMMANDE SEDORIC NUM

Rappel de la syntaxe

NUM (n°_de_la_prochaine_ligne) (,pas_de_la_re-numérotation)

NUM END

Permet de modifier en mémoire les paramètres de la re-numérotation automatique. Si END est indiqué, le dernier n° de ligne utilisé par le programme en cours est recherché et le suivant calculé en ajoutant le pas en cours de validité. A chaque appui sur FUNCT/RETURN, le numéro courant est affiché et le nouveau est calculé. Les valeurs indiquées, comme les valeurs par défaut (en principe 100 pour l'origine et 10 pour le pas), sont également utilisées par RENUM. Il faut utiliser la commande DNUM pour modifier les paramètres par défaut sur la disquette.

Non documenté

NUM tout court est possible et se contente de restaurer les valeurs par défaut présentes en mémoire comme valeurs de travail (très pratique).

La commande NUM n'effectue malheureusement aucune vérification de la validité des paramètres. Il est donc possible de placer dans TRAVNUM (et même dans TRAVPAS) une valeur supérieure à 63999 qui est la limite maximale des n° de ligne BASIC! Cette négligence est assimilable à une bogue: attention à ce que vous tapez!

Les paramètres de certaines commandes peuvent être permutées. Ce n'est pas le cas ici. La première valeur rencontrée est attribuée au n° de la prochaine ligne (sauf si elle est précédée d'une virgule).

Enfin, toujours le même problème avec l'utilisation possible des minuscules pour entrer les commandes SEDORIC: ça ne marche pas à tous les coups (bogue). Ici, le "END" doit être tapé en MAJUSCULES.

Régénère les valeurs à utiliser pour la re-numérotation automatique

EB25-	A0 03	LDY #03	pour copier 4 octets de C03E/C041 (DEFNUM et DEFPAS) en C042/C044 (TRAVNUM et TRAVPAS) afin d'utiliser les valeurs système par défaut comme valeurs de travail
EB27-	B9 3E C0	LDA C03E,Y	lit un octet en commençant par la fin
EB2A-	99 42 C0	STA C042,Y	et le recopie
EB2D-	88	DEY	octet précédent
EB2E-	10 F7	BPL EB27	reboucle en EB27 tant qu'il en reste à copier

Analyse de la syntaxe et saisie des paramètres

EB30-	20 9E D3	JSR D39E	XCRGOT relit le caractère à TXTPTR (sans incrémenter TXTPTR = CHRGOT), puis le convertit en MAJUSCULE, les espaces sont sautés, Z = 1 si fin d'instruction (0 ou :), C = 0 si caractère chiffre 0 à 9 (soit #30 à #39), sinon C = 1, Y et X inchangés
EB33-	F0 5B	BEQ EB90	simple RTS en EB90 s'il n'y a pas de paramètre
EB35-	C9 80	CMP #80	le paramètre est-il le token END?
EB37-	D0 39	BNE EB72	sinon, continue en EB72, si oui...

Cas de NUM END: recherche le n° de la dernière ligne en cours

EB39-	20 98 D3	JSR D398	XCRGET incrémente TXTPTR, lit un caractère (CHRGET), les espaces

			sont sautés, le met dans A, le convertit en MAJUSCULE, Z = 1 si fin d'instruction (0 ou :), C = 0 si chiffre 0 à 9 (soit #30 à #39), sinon C =1. Y et X inchangés
EB3C-	A6 9A	LDX 9A	
EB3E-	A5 9B	LDA 9B	XA = DEBBAS, adresse du début du programme BASIC
EB40-	86 CE	STX CE	
EB42-	85 CF	STA CF	CE/CF reçoit l'adresse de la ligne courante
EB44-	A0 00	LDY #00	
EB46-	B1 CE	LDA (CE),Y	
EB48-	AA	TAX	
EB49-	C8	INY	
EB4A-	B1 CE	LDA (CE),Y	XA reçoit le lien (adresse de la ligne suivante)
EB4C-	F0 10	BEQ EB5E	si HH nul, fin du programme BASIC atteinte, continue en EB5E (seule sortie de cette boucle)
EB4E-	48	PHA	sauvegarde provisoirement le registre A
EB4F-	C8	INY	
EB50-	B1 CE	LDA (CE),Y	
EB52-	8D 42 C0	STA C042	
EB55-	C8	INY	
EB56-	B1 CE	LDA (CE),Y	
EB58-	8D 43 C0	STA C043	C042/C043 reçoit le n° de la ligne courante
EB5B-	68	PLA	récupère A qui n'était pas nul
EB5C-	D0 E2	BNE EB40	donc rebouclage obligatoire en EB40

Calcule le n° de la ligne suivante

EB5E-	18	CLC	
EB5F-	AD 42 C0	LDA C042	
EB62-	6D 44 C0	ADC C044	n° de la dernière ligne trouvé et placé en
EB65-	8D 42 C0	STA C042	C042/C043, calcule le n° de la ligne suivante en
EB68-	AD 43 C0	LDA C043	ajoutant TRAVPAS (C044/C045) à TRAVNUM (C042/C043)
EB6B-	6D 45 C0	ADC C045	
EB6E-	8D 43 C0	STA C043	C042/C043 = C042/C043 + C044/C045
EB71-	60	RTS	et retourne

Suite de l'analyse de syntaxe: nouvelles valeurs de n° et de pas

EB72-	C9 2C	CMP #2C	le caractère à TXTPTR est-il une virgule?
EB74-	F0 0E	BEQ EB84	si oui (premier paramètre absent), continue en EB84
EB76-	20 FA D2	JSR D2FA	E853/ROM évalue un nombre non signé à TXTPTR et revient avec ce nombre dans YA, 33/34 et D3/D4 et mise à jour de TXTPTR sur l'octet suivant ce nombre (C = 0, N et Z positionnés selon l'octet de poids faible)
EB79-	8D 43 C0	STA C043	
EB7C-	8C 42 C0	STY C042	place cette valeur dans C042/C043 (TRAVNUM)
EB7F-	20 9E D3	JSR D39E	XCRGOT relit le caractère à TXTPTR (sans incrémenter TXTPTR = CHRGOT), puis le convertit en MAJUSCULE, les espaces sont sautés, Z = 1 si fin d'instruction (0 ou :), C = 0 si caractère chiffre 0 à 9 (soit #30 à #39), sinon C = 1, Y et X inchangés

EB82-	F0 0C	BEQ EB90	simple RTS en EB90 si fin d'instruction
EB84-	20 2C D2	JSR D22C	D067/ROM exige une "," place TXTPTR sur l'octet suivant
EB87-	20 FA D2	JSR D2FA	E853/ROM évalue un nombre non signé à TXTPTR et revient avec ce nombre dans YA, 33/34 et D3/D4 et mise à jour de TXTPTR sur l'octet suivant ce nombre (C = 0, N et Z positionnés selon l'octet de poids faible)
EB8A-	8D 45 C0	STA C045	
EB8D-	8C 44 C0	STY C044	place cette valeur dans C044/C045 (TRAVPAS)
EB90-	60	RTS	et retourne

EXÉCUTION DE LA COMMANDE SEDORIC ACCENT

Rappel de la syntaxe

ACCENT SET ou **ACCENT OFF**

Permet de définir le jeu de caractères à utiliser: le jeu normal régénéré par la ROM avec ACCENT OFF ou le jeu français dit "accentué" avec ACCENT SET, dans lequel 6 caractères sont redessinés:

Les caractères de code ASCII:	40	5C	7B	7C	7D	7E
représentent avec ACCENT OFF:	@	\	{	|	}	▌
et sont modifiés avec ACCENT SET:	à	ç	é	ù	è	ê

Cette commande doit être utilisée en mode TEXT. Les caractères ainsi générés sont eux utilisables en mode HIRES, il faut donc effectuer la commande ACCENT avant de passer sous HIRES.

Lorsqu'un programme a re-defini ses propres caractères, il est intéressant de régénérer les caractères avec un ACCENT OFF, puis éventuellement un ACCENT SET.

Non documenté

ACCENT tout court génère une “SYNTAX_ERROR”.
La commande AZERTY force automatiquement le mode ACCENT SET.
La commande QWERTY force automatiquement le mode ACCENT OFF.
La commande ACCENT n’est donc utile que dans les cas suivants:

1) ACCENT SET pour avoir les caractères accentués avec un clavier anglais (c’est le cas par exemple d’un utilisateur français avec un ATMOS).

2) AZERTY : ACCENT OFF pour avoir les caractères non accentués avec un clavier français (c’est le cas par exemple d’un français utilisant EUPHORIC avec un logiciel anglais).

3) ACCENT OFF pour régénérer les caractères re-definis par un programme. Ceci équivaut à un CALL#F8D0 (dont on à tendance à oublier l’adresse) ou encore à l’utilisation de FUNCT+”8" avec SEDORIC V3.0 (c’est le cas par exemple d’un utilisateur anglais après avoir utilisé un jeu).

4) ACCENT OFF : ACCENT SET idem pour un utilisateur français qui en plus veut retrouver ses caractères accentués. Ça semble barbare, mais essayez, c’est génial!

Avec les version précédentes de SEDORIC, il n'était pas possible d'obtenir directement le "ê" au clavier. Seul un CHR$(#7E) ou CHR$(126) devait être utilisé. Sinon, il fallait re-definir une touche de fonction. Ce n'est plus le cas avec la version 3.0 qui par simple appui sur FUNCT+"^" affiche bravement un beau ê. Et comble du bonheur, si vous êtes en ACCENT OFF vous obtiendrez le fameux pavé en damier de l'ORIC-1/ATMOS (code ASCII 126), qui était lui aussi inaccessible!

La configuration initiale du clavier (AZERTY/QWERTY et ACCENT SET/OFF) peut utilement être définie et inscrite sur la disquette de boot à l'aide de la commande DKEY. Autre possibilité, la commande INIST. Ainsi par exemple, un utilisateur français peut y insérer la commande ACCENT SET afin de bénéficier dès le démarrage des "caractères accentués".

EB91-	20 4D E9	JSR E94D	XSETOFF teste si le paramètre à TXTPTR est SET ou OFF, retourne avec C = 1 si SET ou C = 0 si OFF sinon génère une "SYNTAX_ERROR"
EB94-	20 DE DF	JSR DFDE	vérifie que l'on est bien en mode TEXT (C n'est pas modifié), sinon génère une "DISP_TYPE_MISMATCH_ERROR", réinitialise la pile et retourne au "Ready"
EB97-	AD 3D C0	LDA C03D	flag MODCLA: b6 à 1 = ACCENT SET, b7 à 1 = AZERTY
EB9A-	29 80	AND #80	remet à zéro tous les bits sauf le b7
EB9C-	90 02	BCC EBA0	si OFF, saute l'instruction suivante
EB9E-	09 40	ORA #40	si ON, force le b6 à 1 (ACCENT SET)
EBA0-	8D 3D C0	STA C03D	remet en place le flag MODCLA

XCHAR sélectionne le jeu de caractères correct ("normal" ou "accentué") selon MODCLA
Suite commune aux commandes ACCENT, AZERTY, QWERTY et à la routine XSTATUS en EC17

EBA3-	2C 3D C0	BIT C03D	teste MODCLA (ACCENT SET, b6 = 1; AZERTY, b7 =1)
EBA6-	70 05	BVS EBAD	si ACCENT SET continue en EBAD
EBA8-	A2 05	LDX #05	sinon indexe X pour sous-programme F982/ROM: copie les caractères
EBAA-	4C 32 D3	JMP D332	normaux de la ROM dans la RAM et retourne

ACCENT SET: re-definit certains caractères:

EBAD-	A9 06	LDA #06	nombre de caractères spéciaux
EBAF-	85 F2	STA F2	(6 soit: à ç é ù è ê)
EBB1-	A2 00	LDX #00	index pour lecture octets
EBB3-	A9 08	LDA #08	8 octets DATA de définition par caractère
EBB5-	85 F3	STA F3	F3 = nombre de DATA à transférer (8)
EBB7-	85 F5	STA F5	F4/F5 adresse où écrire les DATA (pas encore calculée)
EBB9-	BD 4D CD	LDA CD4D,X	table de conversion ACCENT OFF / ACCENT SET
EBBC-	E8	INX	vise l'octet suivant, puis calcule l'adresse d'écriture des DATA (pour un caractère du jeu normal l'adresse des DATA = #B400 + (8 x code ASCII), ici HH de l'adresse vaut déjà #08, après les 2 ROL ce HH deviendra #20 + #94 = #B4)
EBBD-	0A	ASL	A contient le code ASCII du caractère à re-definir
EBBE-	0A	ASL	multiplie par 4 (exemple, pour ç: A = #5C x 4 = #70 et C = 1)
EBBF-	26 F5	ROL F5	sauve C en F5 qui devient égal à 0001 0001 avec C = 0
EBC1-	0A	ASL	multiplie par 8 (A = #70 x 2 = #E0 avec C = 0)
EBC2-	26 F5	ROL F5	sauve C en F5 = 0010 0010 = #22 = 34 et C = 0

EBC4-	85 F4	STA F4	sauve A en F4 = 1110 0000 = #E0 = 224 et C = 0
EBC6-	A5 F5	LDA F5	reprend HH de l'adresse (actuellement adresse = #22E0)
EBC8-	69 94	ADC #94	et ajoute #94 pour pointer au début des caractères normaux
EBCA-	85 F5	STA F5	(dans mon exemple, adresse = #B6E0 c'est celle de "\")
EBCC-	A0 00	LDY #00	index pour écriture
EBCE-	BD 4D CD	LDA CD4D,X	lit un octet de DATA
EBD1-	91 F4	STA (F4),Y	et l'écrit à la bonne place dans la zone des caractères normaux
EBD3-	E8	INX	indexe la lecture de l'octet suivant
EBD4-	C8	INY	indexe l'écriture du DATA suivant
EBD5-	C6 F3	DEC F3	nombre de DATA restant à transférer
EBD7-	D0 F5	BNE EBCE	reboucle tant qu'il en reste
EBD9-	C6 F2	DEC F2	nombre de caractères restant à re-definir
EBDB-	D0 D6	BNE EBB3	reboucle tant qu'il en reste
EBDD-	60	RTS	

EXÉCUTION DE LA COMMANDE SEDORIC AZERTY

Rappel de la syntaxe

AZERTY

Force les modes clavier AZERTY **et** ACCENT SET (en une seule commande).

Cette commande (ATMOS seulement) émule un clavier français en modifiant l'affectation du code ASCII de certaines touches: d'une part les touches **Q** et **A**, **W** et **Z**, **M** et **;** changent de place deux à deux, d'autre part les 6 caractères "accentués" sont validés (ACCENT SET).

Les commandes qui se réfèrent au code de touche (touches de fonctions, KEY$, KEYIF) ne sont pas affectées. Par contre, les commandes qui se réfèrent au code ASCII le sont: pour faire CTRL/A il faudra taper CTRL/Q.

Cette commande doit être utilisée en mode TEXT. Les caractères ainsi générés sont eux utilisables en mode HIRES, il faut donc effectuer la commande AZERTY avant de passer sous HIRES.

Non documenté

La configuration initiale du clavier (AZERTY/QWERTY et ACCENT SET/OFF) peut utilement être définie et inscrite sur la disquette de boot à l'aide de la commande DKEY. Autre possibilité, la commande INIST. Ainsi par exemple, un utilisateur français peut y insérer la commande ACCENT SET afin de bénéficier dès le démarrage des "caractères accentués".

EBDE-	A9 C0	LDA #C0	masque 1100 0000 pour forcer les b5 **et** b7 à 1
EBE0-	2C A9 00	BIT 00A9	suite en EBE3

EXÉCUTION DE LA COMMANDE SEDORIC QWERTY

Rappel de la syntaxe

QWERTY

Force les modes clavier QWERTY et ACCENT OFF (en une seule commande).

Régénère la situation initiale: clavier anglais **et** caractères non "accentués". Cette commande doit être utilisée en mode TEXT. Les caractères ainsi régénérés sont eux utilisables en mode HIRES, il faut donc effectuer la commande AZERTY avant de passer sous HIRES.

Non documenté

La configuration initiale du clavier (AZERTY/QWERTY et ACCENT SET/OFF) peut utilement être définie et inscrite sur la disquette de boot à l'aide de la commande DKEY. Autre possibilité, la commande INIST. Ainsi par exemple, un utilisateur français peut y insérer la commande ACCENT SET afin de bénéficier dès le démarrage des "caractères accentués".

EBE1-	A9 00	LDA #00	masque 0000 0000 pour forcer les b5 **et** b7 à 0
EBE3-	8D 3D C0	STA C03D	écrit le flag dans MODCLA. C'est la routine XKEY "Prendre un caractère au clavier" (D845) qui teste MODCLA et affecte tel ou tel code ASCII selon la touche pressée.
EBE6-	20 DE DF	JSR DFDE	vérifie que l'on est bien en mode TEXT (c'est un peu tard), sinon génère une "DISP_TYPE_MISMATCH_ERROR", réinitialise la pile et retourne au "Ready"
EBE9-	4C A3 EB	JMP EBA3	XCHAR sélectionne le jeu de caractères correct ("normal" ou "accentué") selon MODCLA

EXÉCUTION DE LA COMMANDE SEDORIC LCUR

Rappel de la syntaxe

LCUR

Lit les coordonnées x et y du curseur TEXT et en retourne la valeur dans les variables CX et CY.
Voilà une commande anodine qui devrait bien être plus utilisée car elle est vraiment très pratique.

Non documenté

Ne génère pas d'erreur si on se trouve en mode HIRES.

EBEC-	AD 69 02	LDA 0269	colonne du curseur TEXT

```
EBEF-  AC 68 02   LDY 0268     ligne du curseur TEXT
EBF2-  4C FB EB   JMP EBFB     initialise les variables CX et CY
```

EXÉCUTION DE LA COMMANDE SEDORIC HCUR

Rappel de la syntaxe

HCUR

Lit les coordonnées x et y du curseur HIRES et en retourne la valeur dans les variables CX et CY. Voilà une commande anodine qui devrait bien être plus utilisée car elle est vraiment très pratique.

Non documenté

Ne génère pas d'erreur si on se trouve en mode TEXT.

```
EBF5-  AD 19 02   LDA 0219     colonne du curseur HIRES
EBF8-  AC 1A 02   LDY 021A     ligne du curseur HIRES
```

Initialise les variables CX et CY

```
EBFB-  48         PHA          empile le n° de colonne
EBFC-  98         TYA          prend le n° de ligne
EBFD-  20 E7 D7   JSR D7E7     et le copie dans la variable CY
EC00-  68         PLA          récupère le n° de colonne
EC01-  4C E4 D7   JMP D7E4     le copie dans la variable CX et retourne
```

EXÉCUTION DE LA COMMANDE SEDORIC "]"

Rappel de la syntaxe

DOKE #2F9,adresse_de_la_routine_utilisateur, puis **] (paramètres_utilisateur)**

Cette commande fonctionne exactement comme le couple: DOKE #2F5,adresse_de_la_routine_utilisateur et **!** (paramètres_utilisateur) à ceci près que le **]** ne sera plus reconnu après un QUIT. Comme pour **!**, lorsque des paramètres sont utilisés, la routine utilisateur doit en assurer l'analyse et la saisie dans le buffer d'entrée (TIB en 0035/0084).

```
EC04-  08         PHP          sauvegarde les indicateurs 6502
EC05-  48         PHA          sauvegarde le registre A
EC06-  AD F9 02   LDA 02F9
EC09-  AC FA 02   LDY 02FA     l'adresse présente au vecteur ] est
EC0C-  8D F0 04   STA 04F0     copiée dans EXEVEC
EC0F-  8C F1 04   STY 04F1
EC12-  68         PLA          récupère le registre A
```

EC13-	28	PLP	récupère les indicateurs
EC14-	4C EC 04	JMP 04EC	exécute la routine indiquée à EXEVEC sur RAM overlay (si ROM active) ou sur ROM (si RAM overlay active)

XSTATUS initialise PAPER, INK, mode clavier et status console

Ce sous-programme arrive ici comme un cheveu sur la soupe: il est appelé par le sous-programme NMI SEDORIC en 04C4, avant de passer au warmstart BASIC.

EC17-	A9 10	LDA #10	papier noir
EC19-	A0 07	LDY #07	encre blanche
EC1B-	8D 6B 02	STA 026B	dans PAPER
EC1E-	8C 6C 02	STY 026C	et dans INK
EC21-	A9 0F	LDA #0F	status pour Double Hauteur OFF, colonnes 0 et 1 non protégées, flag ESC OFF, key-clic OFF, affichage écran ON et curseur ON
EC23-	8D 6A 02	STA 026A	mode console ORIC-1/ATMOS
EC26-	A9 0C	LDA #0C	CTRL/L pour vider l'écran
EC28-	20 2A D6	JSR D62A	XAFCAR affiche le caractère ASCII contenu dans A
EC2B-	4C A3 EB	JMP EBA3	XCHAR sélectionne le jeu de caractères correct (“normal” ou “accentué”) selon MODCLA

EXÉCUTION DE LA COMMANDE SEDORIC INSTR

Rappel de la syntaxe

INSTR expression_alphanumérique_n°1,expression_alphanumérique_n°2,position

Recherche la prochaine occurrence de la chaîne n°2 (expression alphanumérique n°2) dans la chaîne n°1 (expression alphanumérique n°1) et ceci en commençant la recherche à partir de la position indiquée. Retourne dans la variable IN la position de l'occurrence trouvée (IN = 1 si la chaîne n°2 commence au premier caractère de la chaîne n°1). Renvoie IN = 0 si la chaîne à examiner (expression alphanumérique n°1) est vide ou si la chaîne recherchée (expression alphanumérique n°2) est vide ou n'est pas trouvée et "ILLEGAL_QUANTITY_ERROR" si la position indiquée est nulle ou supérieure à la longueur de la chaîne à examiner.

Bogue dans le manuel concernant ce que retourne cette commande. Le manuel indique qu’une “ILLEGAL_QUANTITY_ERROR” est générée si la chaîne alphanumérique recherchée (expression alphanumérique n°2) est vide, ce qui n’est pas le cas (voir ci-dessus).

EC2E-	20 24 D2	JSR D224	JSR CF17/ROM évalue une expression à TXTPTR, retourne avec une valeur numérique dans ACC1 (et #00 dans 28) ou l'adresse d'une chaîne dans D3/D4 (et #FF dans 28) et A, N et Z positionnés selon l'exposant (résultat numérique) ou la longueur de chaîne
EC31-	20 74 D2	JSR D274	JSR D7D0 et CF09/ROM vérifie si l'expression évaluée à TXTPTR est bien alphanumérique, retourne l'adresse de la chaîne dans XY et 91/92 ainsi que sa longueur dans A
EC34-	85 F2	STA F2	longueur de la première chaîne

EC36-	A8	TAY	
EC37-	88	DEY	
EC38-	B1 91	LDA (91),Y	copie la première chaîne au début de BUF1
EC3A-	99 00 C1	STA C100,Y	
EC3D-	98	TYA	
EC3E-	D0 F7	BNE EC37	
EC40-	20 2C D2	JSR D22C	D067/ROM exige une "," place TXTPTR sur l'octet suivant
EC43-	20 24 D2	JSR D224	JSR CF17/ROM évalue une expression à TXTPTR, retourne avec une valeur numérique dans ACC1 (et #00 dans 28) ou l'adresse d'une chaîne dans D3/D4 (et #FF dans 28) et A, N et Z positionnés selon l'exposant (résultat numérique) ou la longueur de chaîne
EC46-	20 74 D2	JSR D274	JSR D7D0 et CF09/ROM vérifie si l'expression évaluée à TXTPTR est bien alphanumérique, retourne l'adresse de la chaîne dans XY et 91/92 ainsi que sa longueur dans A
EC49-	85 F3	STA F3	longueur de la deuxième chaîne
EC4B-	86 B8	STX B8	B8/B9 vise le début de la seconde chaîne
EC4D-	84 B9	STY B9	
EC4F-	20 2C D2	JSR D22C	D067/ROM exige une "," place TXTPTR sur l'octet suivant
EC52-	20 7F D2	JSR D27F	CF17/ROM et CF09/ROM et D8CB/ROM évalue un nombre entier à TXTPTR et le retourne dans X (position où commencer la recherche)
EC55-	D0 37	BNE EC8E	si pas fin de commande, "SYNTAX_ERROR"
EC57-	CA	DEX	index = position - 1
EC58-	86 F6	STX F6	index où commencer la recherche
EC5A-	E4 F2	CPX F2	vérifie la validité de la position indiquée
EC5C-	B0 33	BCS EC91	"ILLEGAL_QUANTITY_ERROR" si >= longueur première chaîne
EC5E-	A5 F2	LDA F2	longueur de la première chaîne
EC60-	F0 1C	BEQ EC7E	la première chaîne est vide, termine en EC7E (IN = 0)
EC62-	A6 F3	LDX F3	longueur de la deuxième chaîne
EC64-	F0 18	BEQ EC7E	la deuxième chaîne est vide, termine en EC7E (IN = 0)
EC66-	A5 F6	LDA F6	LL de l'adresse du premier caractère à traiter
EC68-	85 F7	STA F7	dans la première chaîne (où commencer la comparaison)
EC6A-	A9 C1	LDA #C1	HH de l'adresse du premier caractère à traiter
EC6C-	85 F8	STA F8	dans la première chaîne (c'est le HH de BUF1)
EC6E-	A0 00	LDY #00	
EC70-	B1 F7	LDA (F7),Y	lit un caractère de la première chaîne
EC72-	D1 B8	CMP (B8),Y	et le compare au caractère homologue de la deuxième
EC74-	D0 0E	BNE EC84	continue en EC84 s'ils sont différents
EC76-	C8	INY	vise le caractère suivant s'ils sont identiques
EC77-	CA	DEX	et décrémente le nombre de caractères à trouver
EC78-	D0 F6	BNE EC70	reboucle en EC70 s'il en reste à trouver
EC7A-	A4 F6	LDY F6	récupère la position trouvée dans la première chaîne
EC7C-	C8	INY	et l'ajuste (rang 0 correspond au premier caractère)
EC7D-	2C A0 00	BIT 00A0	continue en EC80
EC7E-	A0 00	LDY 00	recherche infructueuse (où chaîne vide)
EC80-	98	TYA	dans tous les cas, A reçoit la position
EC81-	4C DB D7	JMP D7DB	mise à jour de la variable IN avec la valeur A

Caractères différents: reprend la comparaison

EC84-	E6 F6	INC F6	vise le caractère suivant de la première chaîne
EC86-	A5 F6	LDA F6	
EC88-	C5 F2	CMP F2	teste si la fin de la première chaîne est atteinte
EC8A-	F0 F2	BEQ EC7E	si oui, recherche infructueuse, termine en EC7E
EC8C-	D0 D4	BNE EC62	sinon, reboucle en EC62
EC8E-	4C 23 DE	JMP DE23	"SYNTAX_ERROR"
EC91-	4C 20 DE	JMP DE20	"ILLEGAL_QUANTITY_ERROR"

EXÉCUTION DE LA COMMANDE SEDORIC LINPUT

Rappel de la syntaxe

LINPUT(@x,y,)(car,)long;VA(,E)(,S)(,C)(,J)(,K)

@x,y, permet d'effectuer la saisie à la position xy de l'écran (cette fonction, qui était boguée, marche correctement maintenant).

car, caractère ou premier caractère d'une expression alphanumérique. Sert à matérialiser la place des caractères demandés ("." par défaut). Est conservé d'un LINPUT à l'autre, tant qu'il n'est pas modifié). Pour revenir au ".", il faut re-indiquer "." ou faire RESET!

long nombre de caractères à saisir (1 à 255), paramètre obligatoire.

;VA variable alphanumérique où placer la chaîne qui sera saisie, paramètre obligatoire.

Si aucun des 5 paramètres qui suivent n'est indiqué, le curseur, arrivé en fin de fenêtre, rebouclera au début de la fenêtre. Pour sortir, il faudra utiliser les flèches, ESC ou RETURN. Le choix des caractères E, S, C, K et J qui a été retenu est particulièrement hermétique. Il faut le vouloir pour s'en souvenir!

,E Le manuel indique "permet de ne pas effacer la fenêtre avant l'entrée du texte". En fait, ce paramètre permet de ne pas afficher le caractère de remplissage ("matérialisation" de la fenêtre, qui est toujours effectuée par défaut). NB: On n'efface vraiment la fenêtre qu'en indiquant " " comme caractère de remplissage.

,S Contrairement à ce qui est indiqué dans le manuel (**bogue**), interdit de sortir avec les flèches de déplacement (mode par défaut).

,C Sortie automatique lorsque le curseur atteint la fin de la fenêtre. Sinon, par défaut, le curseur revient au début de la fenêtre.

,K "Justifie" le texte entré à l'écran (mais pas la variable) en remplaçant les caractères de remplissage par des espaces.

,J Inversement, "justifie la variable", sans affecter l'affichage. La combinaison ",J,K" permet de "justifier" l'écran et la variable.

Cette commande permet la saisie formatée de texte dans une variable alphanumérique. Il s'agit en fait d'une fenêtre avec éditeur de type pleine page. Sont valides: CTRL/D (double hauteur), CTRL/T (minuscules/MAJUSCULES), CTRL/N (effacement ligne), CTRL/Z (ESC pour attributs vidéo), DEL, ESC (sortie), RETURN (sortie) et flèches (déplacement et sortie). Attention à la bogue du manuel qui oublie le CTRL/D.

LINPUT, dont l'entrée est en EC94, analyse la syntaxe et les options demandées, fait appel au sous-programme XLINPU, qui est en fait la partie principale de LINPUT et qui altère en sortie les adresses suivantes:

12/13	adresse de la ligne du curseur TEXT
1F/20	calcul de l'adresse de la ligne du curseur TEXT
28	flag numérique/alphanumérique (#00 si numérique et #FF si chaîne)
91/92	adresse de la chaîne pointée par TXTPTR
B6/B7	pointe sur le descripteur de chaîne
B8/B9	pointe sur le descripteur de chaîne
D0	longueur de la chaîne saisie
D1/D2	adresse de la chaîne saisie
F2	nombre de caractères à saisir
F3	indique quels paramètres ont été sélectionnés (E, S, C, J et K)
F4	index lié au dernier paramètre: b3 pour E, b4 pour K, b5 pour J, b6 pour C et b7 pour S; puis, à la fin, contient le mode de sortie
F5	nombre de colonnes actives selon 026A (indicateur état console), puis à la fin constitue une réplique de F2
0268	n° de la ligne du curseur (ordonnée y, de 1 à 27)
0269	et 02F8 le n° de la colonne du curseur (abscisse x, de 0 à 39)
C075	caractère à utiliser pour remplir la fenêtre de saisie.

Enfin, la variable OM (Output Mode) contient le mode de sortie: 0 (si RETURN), 1 (si ESC), 2 à 5 (si flèches gauche, droite, bas et haut respectivement) et 6 (si sortie automatique).

La bogue de LINPUT

LINPUT présentait un grave problème de gestion du curseur. Après un LINPUT, le positionnement du curseur était complètement faussé, rendant impossible l'utilisation de certaines autres commandes, par exemple un PRINT@ ou un autre LINPUT@. Ceci apparaissait aussi lorsque la longueur de la chaîne demandée dépassait 38 caractères. Les facéties du curseur étaient quasiment imprévisibles et rendaient impossible l'utilisation, à coup sûr, du paramètre "@x,y,". Ceci venait du fait d'une mauvaise gestion des coordonnées xy du curseur. J'ai corrigé cette bogue en ajoutant un simple STX 30 (voir en ECB5).

Début de l'analyse de la commande LINPUT

EC94-	AA	TAX	met de coté le premier caractère après commande LINPUT (paramètre)
EC95-	AD 6A 02	LDA 026A	empile mode console ORIC-1/ATMOS à l'entrée
EC98-	48	PHA	notamment affichage curseur qui sera changé
EC99-	E0 C6	CPX #C6	premier caractère est-il "@", préfixe de coordonnées xy?

EC9B-	D0 1E	BNE ECBB	sinon, saute la gestion des coordonnées xy

Gestion des coordonnées xy

EC9D-	20 98 D3	JSR D398	XCRGET incrémente TXTPTR, lit un caractère (CHRGET), les espaces sont sautés, le met dans A, le convertit en MAJUSCULE, Z = 1 si fin d'instruction (0 ou :), C = 0 si chiffre 0 à 9 (soit #30 à #39), sinon C =1. Y et X inchangés, ceci afin de positionner convenablement TXTPTR pour la routine DA22/ROM
ECA0-	20 40 D7	JSR D740	XCUROFF cache le curseur (= vidéo normale)
ECA3-	20 92 D2	JSR D292	JSR DA22/ROM Prend 2 coordonnées à TXTPTR. Au retour, X contient le n° de ligne (y), 02F8 contient le n° de colonne (x) et 1F/20 contient l'adresse LLHH de la ligne (A contient aussi le LL de l'adresse)
ECA6-	A4 20	LDY 20	HH poids fort de l'adresse de la ligne cible
ECA8-	85 12	STA 12	copie en 12/13 l'adresse de la ligne où
ECAA-	84 13	STY 13	devra agir LINPUT pour la saisie formatée
ECAC-	8E 68 02	STX 0268	copie ordonnée y en 0268 (n° de ligne cible)
ECAF-	AE F8 02	LDX 02F8	copie abscisse x en 0269 (n° colonne cible)
ECB2-	8E 69 02	STX 0269	

J'ai corrigé la bogue de LINPUT en utilisant le principe usuel: pour palier à un manque local de place, une partie du code (ici un JMP D73E, XCURON rend le curseur visible) est remplacée par un saut à une routine insérée ailleurs (ici un JSR EA36) et dans laquelle on peut à "loisir" ajouter les opérations supplémentaires (ici un simple STX 30! et reprendre le code supprimé (ici le JMP D73E).

ECB5-	20 **36 EA**	JSR EA36	remplaçant l'ancien JSR D73E (voir ci-dessus)
ECB8-	20 2C D2	JSR D22C	D067/ROM exige une "," place TXTPTR sur l'octet suivant

Gestion du caractère de remplissage

ECBB-	20 24 D2	JSR D224	JSR CF17/ROM évalue une expression à TXTPTR, retourne avec une valeur numérique dans ACC1 (et #00 dans 28) ou l'adresse d'une chaîne dans D3/D4 (et #FF dans 28) et A, N et Z positionnés selon l'exposant (résultat numérique) ou la longueur de chaîne
ECBE-	24 28	BIT 28	valeur numérique si #00 dans 28, chaîne si #FF
ECC0-	10 15	BPL ECD7	si numérique, saute le sous-programme de gestion du caractère à afficher et continue au sous-programme d'évaluation du nombre de caractères à saisir. Si aucune chaîne n'est spécifiée, le programme poursuit en EDC7 et le caractère de remplissage sera celui présent en C075 ("." lors du boot).
ECC2-	20 77 D2	JSR D277	JSR D7D0/ROM (longueur de la chaîne dans A avec Z selon cette longueur et adresse de la chaîne dans XY et 91/92)
ECC5-	F0 05	BEQ ECCC	branche en ECCC si la longueur de la chaîne est nulle
ECC7-	A0 00	LDY #00	Y = premier caractère de cette chaîne qui
ECC9-	B1 91	LDA (91),Y	permettra de remplir ultérieurement la fenêtre
ECCB-	2C A9 2E	BIT 2EA9	et continue en ECCE
ECCC-	A9 2E	LDA #2E	sinon A = "." (pris par défaut)
ECCE-	8D 75 C0	STA C075	le caractère de remplissage A sera gardé en C075 d'un LINPUT à l'autre, tant que pas de nouvelle chaîne spécifiée

ECD1-	20 2C D2	JSR D22C	D067/ROM exige une "," place TXTPTR sur l'octet suivant
ECD4-	20 24 D2	JSR D224	JSR CF17/ROM évalue une expression à TXTPTR, retourne avec une valeur numérique dans ACC1 (et #00 dans 28) ou l'adresse d'une chaîne dans D3/D4 (et #FF dans 28) et A, N et Z positionnés selon l'exposant (résultat numérique) ou la longueur de chaîne

Evaluation du nombre de caractères à saisir

ECD7-	20 19 D2	JSR D219	vérifie que l'expression évaluée à TXTPTR est bien numérique. Retourne avec la valeur dans ACC1 ou "TYPE_MISMATCH_ERROR". Le paramètre "nombre de caractères à saisir" est obligatoire
ECDA-	20 82 D2	JSR D282	JSR D8CB/ROM Prend un entier dans ACC1 et le retourne dans X (nombre de caractères à saisir)
ECDD-	8A	TXA	teste ce nombre de caractères (Z = 1 si nul)
ECDE-	F0 4B	BEQ ED2B	si X = 0, "ILLEGAL_QUANTITY_ERROR"
ECE0-	86 F2	STX F2	sauve le nombre de caractères à saisir dans F2

Demande la variable alphanumérique requise

ECE2-	A9 3B	LDA #3B	A = ";" (marqueur situé devant la variable)
ECE4-	20 2E D2	JSR D22E	JSR D067/ROM demande ";" à TXTPTR, lit le caractère suivant en continuant au sous-programme CHARGET en E2 (qui saute les espaces), puis à l'interpréteur SEDORIC en 0400, puis au sous-programme ECB9/ROM et finalement convertit ce caractère en MAJUSCULE (sous-programme D3A1/RAM overlay). Au cours de ce périple, le sort de Y est assez indéfini, mais semble finir avec Y = 0
ECE7-	84 F3	STY F3	supposons que Y = #00 (sauf contre indication!)
ECE9-	20 2E ED	JSR ED2E	prendre adresse de la variable à TXTPTR (B8/B9)
ECEC-	20 1B D2	JSR D21B	SEC et JSR CF09/ROM vérifie si alphanumérique
ECEF-	20 9E D3	JSR D39E	XCRGOT relit le caractère à TXTPTR (sans incrémenter TXTPTR = CHRGOT), puis le convertit en MAJUSCULE, les espaces sont sautés, Z = 1 si fin d'instruction (0 ou :), C = 0 si caractère chiffre 0 à 9 (soit #30 à #39), sinon C = 1, Y et X inchangés
ECF2-	F0 25	BEQ ED19	Si nul, fin des paramètres, continue à XLINPU

Analyse des paramètres de fin de commande, lorsqu'ils existent

ECF4-	20 2C D2	JSR D22C	Si pas nul, il y a encore des paramètres, JSR D067/ROM demande une "," lit le caractère suivant et le convertit en MAJUSCULE
ECF7-	20 A1 D3	JSR D3A1	re-convertit en MAJUSCULE (pour être bien sûr!)
ECFA-	A2 04	LDX #04	X = 4 pour lire 5 paramètres dans une table
ECFC-	86 F4	STX F4	sauve cet index dans F4 (seul bit b2 est à 1, soit 0000 00**1**0)
ECFE-	06 F4	ASL F4	décale vers la gauche le bit qui est à 1. Ce bit passera donc successivement des positions b3 à b4 puis b5 puis b6 puis b7 selon le paramètre trouvé.
ED00-	DD BA CD	CMP CDBA,X	cherche si le paramètre visé est l'une de ces 5 lettres suivantes: **E** (en CDBE avec X = #04), **K** (en CDBD avec X = #03), **J** (en CDBC avec X = #02), **C** (en CDBB avec X = #01) ou **S** (en CDBA avec X = #00)

Index X	0	1	2	3	4
Paramètre	S	C	J	K	E
n° bit à 1 dans F4	7	6	5	4	3

ED03-	F0 05	BEQ ED0A	si oui, branche en ED0A
ED05-	CA	DEX	sinon, indexe la lettre précédente dans table
ED06-	10 F6	BPL ECFE	et reboucle en ECFE (suite de la recherche)
ED08-	30 1E	BMI ED28	"SYNTAX_ERROR": le paramètre situé après la virgule n'est pas l'un de ceux qui sont autorisés par la syntaxe de LINPUT
ED0A-	A5 F4	LDA F4	porte le flag du paramètre trouvé: b3 à 1 si **E**, b4 à 1 si **K**, b5 à 1 si **J**, b6 à 1 si **C** ou b7 à 1 si **S**
ED0C-	45 F3	EOR F3	A = F3 plus copie du bit qui est à 1 dans F4 (5 paramètres et 5 places de b3 à b7), mais "efface" tout bit qui serait déjà à 1 dans F3 (doublon dans commande), ce qui entraîne comme résultat A < F3
ED0E-	C5 F3	CMP F3	compare le résultat avec F3
ED10-	90 16	BCC ED28	si A < F3 branche vers "SYNTAX_ERROR", ce qui se produit si un paramètre figure en double dans la commande!
ED12-	85 F3	STA F3	sinon, sauve A dans F3 qui garde donc trace de tous les paramètres qui ont été trouvés et sera utilisé par XLINPU
ED14-	20 98 D3	JSR D398	XCRGET incrémente TXTPTR, lit un caractère (CHRGET), les espaces sont sautés, le met dans A, le convertit en MAJUSCULE, Z = 1 si fin d'instruction (0 ou :), C = 0 si chiffre 0 à 9 (soit #30 à #39), sinon C =1. Y et X inchangés
ED17-	D0 DB	BNE ECF4	reboucle s'il reste des paramètres, sinon...
ED19-	20 36 ED	JSR ED36	XLINPU routine principale de LINPUT (routine de saisie de chaîne), au retour F4 contient le mode de sortie et D0, D1, D2 donne la longueur et l'adresse de la chaîne dans la zone de stockage des chaînes sous HIMEM.
ED1C-	20 8E EE	JSR EE8E	au retour, XCSTR copie la longueur et l'adresse de la chaîne (les 3 octets D0, D1 et D2) “dans” la variable alphanumérique indiquée dans la ligne de commande et pointée par les 3 octets B8, B9 et BA.
ED1F-	68	PLA	
ED20-	8D 6A 02	STA 026A	restaure le mode console ORIC-1/ATMOS initial
ED23-	A5 F4	LDA F4	copie n° du mode de sortie
ED25-	4C D8 D7	JMP D7D8	dans la variable OM et retourne
ED28-	4C 23 DE	JMP DE23	"SYNTAX_ERROR"
ED2B-	4C 20 DE	JMP DE20	"ILLEGAL_QUANTITY_ERROR"

Prend l'adresse de la valeur de la variable à TXTPTR

ED2E-	20 38 D2	JSR D238	JSR D188/ROM décode le nom de la première variable à TXTPTR et place

"l'adresse" de cette variable dans AY, B6/B7 et D3/D4, cette adresse pointe en fait sur les data (longueur/adresse dans le cas d'une chaîne) de la variable dans la zone des variables BASIC

ED31-	85 B8	STA B8	et la copie aussi dans B8/B9
ED33-	84 B9	STY B9	
ED35-	60	RTS	

XLINPU Routine principale de LINPUT (routine de saisie de chaîne)

Cette routine est aussi utilisée par les commandes DNAME et INIST.

En entrée, 0269 et 0268 contiennent les coordonnées xy du point de début de fenêtre. C075 contient le caractère à utiliser pour matérialiser la fenêtre. F2 contient la longueur de la chaîne à saisir. F3 indique quelles options (E, S, C, J, K) ont été demandées. B8/B9 contient l'adresse de la variable où il faudra copier la longueur et l'adresse de la chaîne.

XLINPU, qui utilise la routine fondamentale D843 "Prendre un caractère au clavier", s'occupe de saisir les caractères selon les options choisies.

Au retour, F4 contient le mode de sortie, D0 et D1/D2 donnent la longueur et l'adresse de la chaîne (erreur dans le manuel p 109) dans la zone de stockage des chaînes sous HIMEM.

ED36-	A5 F3	LDA F3	indique quels paramètres ont été sélectionnés: b3 à 1 si **E**, b4 à 1 si **K**, b5 à 1 si **J**, b6 à 1 si **C** et b7 à 1 si **S**
ED38-	29 08	AND #08	teste si b3 est à 1 (**E**, ne pas "effacer" la fenêtre)
ED3A-	D0 16	BNE ED52	si oui, saute "l'effacement" et continue en ED52

"Efface" la fenêtre avant l'entrée du texte
En fait, remplit avec le caractère en C075 puis revient en début de fenêtre.

ED3C-	20 40 D7	JSR D740	XCUROFF cache le curseur (= vidéo normale)
ED3F-	A6 F2	LDX F2	nombre de caractères à saisir
ED41-	AD 75 C0	LDA C075	caractère à afficher dans la fenêtre de saisie
ED44-	20 2A D6	JSR D62A	affiche X fois ce caractère dans la fenêtre (routine XAFCAR)
ED47-	CA	DEX	
ED48-	D0 FA	BNE ED44	
ED4A-	20 40 D7	JSR D740	XCUROFF cache le curseur (= vidéo normale)
ED4D-	A6 F2	LDX F2	nombre de caractères à saisir
ED4F-	20 69 EE	JSR EE69	XAFXGAU affiche X fois "flèche gauche": curseur au début du masque de saisie

Met en F5 le nombre de colonnes actives selon 026A

ED52-	20 3E D7	JSR D73E	XCURON rend le curseur visible (= vidéo inverse)
ED55-	A2 00	LDX #00	compteur du nombre de caractères saisis
ED57-	A0 26	LDY #26	soit 38 colonnes par défaut
ED59-	AD 6A 02	LDA 026A	mode console ORIC-1/ATMOS
ED5C-	29 20	AND #20	teste si b5 à 1 (mode 38 colonnes)

```
ED5E- F0 02     BEQ ED62    sinon, saute instruction suivante
ED60- A0 28     LDY #28     40 colonnes
ED62- 84 F5     STY F5      F5 = nombre de colonnes actives selon 026A
```

Début de la saisie dans la fenêtre

```
ED64- 20 43 D8  JSR D843    sous-programme "Prendre un caractère au clavier"
ED67- 10 FB     BPL ED64    reboucle tant que pas de touche
ED69- C9 14     CMP #14     est-ce CTRL/T?
ED6B- F0 23     BEQ ED90    si oui, caractère valide, continue en ED90
ED6D- C9 7F     CMP #7F     est-ce DEL?
ED6F- D0 0E     BNE ED7F    continue en ED7F si ce n'est pas le cas
```

Examine si le DEL saisi est valable

```
ED71- 8A        TXA         teste si X = 0 (nombre de caractères actuellement saisis)
ED72- F0 F0     BEQ ED64    si oui, DEL invalide, reboucle en ED64 (saisie)
ED74- 20 73 EE  JSR EE73    sinon, DEL valide, XAF1GAU affiche une "flèche gauche"
ED77- AD 75 C0  LDA C075    reprend le caractère de remplissage fenêtre
ED7A- 20 2A D6  JSR D62A    XAFCAR: affiche ce caractère et avance à droite
ED7D- A9 08     LDA #08     reprend suite normale avec A = "flèche gauche" pour compenser
```

Suite "1" de l'analyse du caractère saisi

```
ED7F- C9 0E     CMP #0E     est-ce CTRL/N? (effacement de ligne en cours)
ED81- D0 05     BNE ED88    continue en ED88 si ce n'est pas le cas
ED83- 20 69 EE  JSR EE69    si oui, XAFXGAU affiche X fois "flèche gauche"
ED86- F0 B4     BEQ ED3C    et reprend au début de la saisie
```

Suite "2" de l'analyse du caractère saisi

```
ED88- C9 04     CMP #04     est-ce CTRL/D? (double hauteur)
ED8A- F0 04     BEQ ED90    si oui, caractère valide, continue en ED90
ED8C- C9 1A     CMP #1A     est-ce CTRL/Z? (ESC pour attribut écran)
ED8E- D0 05     BNE ED95    continue en ED95 si ce n'est pas le cas
```

Affiche un caractère de CTRL valide et reboucle en saisie

```
ED90- 20 2A D6  JSR D62A    XAFCAR "affiche" le caractère de contrôle valide
ED93- D0 CF     BNE ED64    et reboucle en ED64 (sans incrémenter X)
```

Suite "3" de l'analyse du caractère saisi

```
ED95- C9 20     CMP #20     est-ce un autre code de contrôle? (A < #20)
ED97- 90 14     BCC EDAD    si oui, continue en EDAD, sinon...
```

Affiche caractère valide et reboucle si nécessaire

```
ED99- 20 2A D6  JSR D62A   XAFCAR: affiche A (et déplace curseur à droite)
ED9C- E8        INX        compteur de caractères saisis et affichés
ED9D- E4 F2     CPX F2     le nombre requis est-il atteint?
ED9F- D0 C3     BNE ED64   sinon, reprend en ED64 (saisie au clavier)
EDA1- 24 F3     BIT F3     si oui, teste si b6 est à 1 (C, sortie auto)
EDA3- 50 A5     BVC ED4A   sinon, reboucle (curseur en début de fenêtre)
```

Sortie automatique à la fin de la fenêtre

```
EDA5- CA        DEX        si oui, décrémente le nombre de caractères
EDA6- 20 73 EE  JSR EE73   et XAF1GAU affiche une "flèche gauche"
EDA9- A0 06     LDY #06    Y = 6 pour sortie automatique en fin de fenêtre
EDAB- D0 57     BNE EE04   qui n'est pas nul... continue en EE04
```

Suite "1" de l'analyse des codes de contrôle

```
EDAD- A0 00     LDY #00    Y à zéro indiquera "RETURN"
EDAF- C9 0D     CMP #0D    est-ce RETURN?
EDB1- F0 49     BEQ EDFC   si oui, continue en EDFC avec Y = 0
EDB3- C8        INY        sinon, incrémente Y (qui passe à 1)
EDB4- C9 1B     CMP #1B    est-ce ESC?
EDB6- F0 44     BEQ EDFC   si oui, continue en EDFC avec Y = 1
EDB8- C8        INY        sinon, incrémente Y (qui passe à 2)
EDB9- C9 08     CMP #08    est-ce "flèche gauche"?
EDBB- D0 09     BNE EDC6   sinon, continue en EDC6
```

Gestion flèche gauche

```
EDBD- 8A        TXA        si oui, teste si X est nul (aucun caractère n'a été saisi)
EDBE- F0 3C     BEQ EDFC   si oui, continue en EDFC avec Y = 2
EDC0- CA        DEX        s'il y a déjà des caractères, décrémente X
EDC1- 20 73 EE  JSR EE73   XAF1GAU affiche une "flèche gauche"
EDC4- D0 9E     BNE ED64   et reboucle en ED64 (saisie au clavier)
```

Suite "2" de l'analyse des codes de contrôle

```
EDC6- C8        INY        incrémente Y (qui passe à 3)
EDC7- C9 09     CMP #09    est-ce "flèche droite"?
EDC9- D0 0E     BNE EDD9   sinon, continue en EDD9
```

Gestion flèche droite

```
EDCB- E8        INX        si oui, incrémente le nombre de caractères
EDCC- E4 F2     CPX F2     le nombre requis est-il atteint?
EDCE- F0 05     BEQ EDD5   si oui, continue en EDD5 (saisie terminée)
EDD0- 20 76 EE  JSR EE76   sinon, XAF1DR affiche une "flèche droite"
EDD3- D0 BE     BNE ED93   et reboucle en ED64 (saisie au clavier)
```

Saisie au clavier terminée

EDD5-	CA	DEX	décrémente le nombre de caractères
EDD6-	4C FC ED	JMP EDFC	continue en EDFC pour sortie

Suite "3" de l'analyse des codes de contrôle

EDD9-	C8	INY	incrémente Y (qui passe à 4)
EDDA-	C9 0A	CMP #0A	est-ce "**flèche vers le bas**"?
EDDC-	D0 0F	BNE EDED	sinon, continue en EDED

gestion flèche vers le bas

EDDE-	18	CLC	prépare une addition
EDDF-	8A	TXA	copie dans A le nombre de caractères saisis
EDE0-	65 F5	ADC F5	et ajoute nombre de colonnes actives selon 026A
EDE2-	B0 18	BCS EDFC	si résultat > 255, branche en EDFC avec Y = 4
EDE4-	C5 F2	CMP F2	teste si résultat >= nombre de caractères requis
EDE6-	B0 14	BCS EDFC	si oui, continue en EDFC avec Y = 4
EDE8-	AA	TAX	le nombre de caractères saisis augmente de 38 (ou 40)
EDE9-	A9 0A	LDA #0A	"flèche vers le bas"
EDEB-	D0 A3	BNE ED90	affiche un caractère de CTRL valide et reboucle en saisie

Suite "4" de l'analyse des codes de contrôle

EDED-	C8	INY	incrémente Y (qui passe à 5)
EDEE-	C9 0B	CMP #0B	est-ce "**flèche vers le haut**"?
EDF0-	D0 A1	BNE ED93	sinon, reboucle en ED64 (saisie au clavier)

Gestion flèche vers le haut

EDF2-	8A	TXA	copie dans A le nombre de caractères saisis
EDF3-	E5 F5	SBC F5	et retire nombre de colonnes actives selon 026A
EDF5-	90 05	BCC EDFC	si résultat < 0, continue en EDFC avec Y = 5
EDF7-	AA	TAX	le nombre de caractères saisis diminue de 38 (ou 40)
EDF8-	A9 0B	LDA #0B	"flèche vers le haut"
EDFA-	D0 94	BNE ED90	affiche un caractère de CTRL valide et reboucle en saisie

Sortie

EDFC-	C0 02	CPY #02	Y vaut 0 ou 1? (RETURN ou ESC)
EDFE-	90 04	BCC EE04	si oui, saute les deux instructions suivantes
EE00-	A5 F3	LDA F3	teste si S (permet de sortir avec les flèches)
EE02-	30 8F	BMI ED93	si oui, reboucle en ED64 (saisie au clavier)
EE04-	84 F4	STY F4	F4 contient désormais le mode de sortie: 0 si RETURN, 1 si ESC, 2 si flèche gauche, 3 si flèche droite, 4 si flèche vers le bas, 5 si flèche vers le haut et 6 si sortie auto en fin de fenêtre.

Positionne le curseur à la fin de la fenêtre

EE06-	20 40 D7	JSR D740	XCUROFF cache le curseur (= vidéo normale)
EE09-	E8	INX	incrémente le nombre de caractères
EE0A-	E4 F2	CPX F2	le nombre requis est-il atteint?
EE0C-	B0 05	BCS EE13	si X >= F2, continue en EE13
EE0E-	20 76 EE	JSR EE76	XAF1DR affiche une "flèche droite"
EE11-	D0 F6	BNE EE09	si X < F2 reboucle en EE09

Recopie la fenêtre dans la variable alphanumérique

EE13-	A5 F2	LDA F2	nombre de caractères requis
EE15-	20 64 D2	JSR D264	JSR D5AB/ROM réserve une place en mémoire pour la chaîne de longueur A, retourne avec cette longueur en D0 et l'adresse de la chaîne en D1/D2
EE18-	A4 F2	LDY F2	
EE1A-	84 F5	STY F5	nombre de caractères requis
EE1C-	AC 69 02	LDY 0269	n° de colonne du curseur (de 0 à 39)
EE1F-	B1 12	LDA (12),Y	lit caractère sous le curseur
EE21-	C9 20	CMP #20	teste si >= #20 (c'est à dire si c'est un caractère proprement dit)
EE23-	B0 02	BCS EE27	si oui, saute l'instruction suivante
EE25-	09 80	ORA #80	sinon, force à 1 le b7 du code lu au curseur, afin de retomber sur des caractères affichables normalement. A l'écran les attributs vidéo sont codés de 0 (encre noire) à 31 (GRAPHICS 50Hz). Mais, pour être inclus dans une chaîne, ces attributs sont codés de 128 à 159.
EE27-	A4 F5	LDY F5	nombre de caractères requis
EE29-	88	DEY	que l'on diminue
EE2A-	08	PHP	sauvegarde les indicateurs 6502 dont Z
EE2B-	91 D1	STA (D1),Y	écrit le caractère ou code lu dans la chaîne
EE2D-	20 73 EE	JSR EE73	XAF1GAU affiche une "flèche gauche" (recule le curseur)
EE30-	28	PLP	récupère les indicateurs 6502 dont Z
EE31-	D0 E7	BNE EE1A	reboucle tant qu'il en reste à copier

Retourne à la fin de la fenêtre

EE33-	A6 F2	LDX F2	nombre de caractères requis
EE35-	20 76 EE	JSR EE76	XAF1DR affiche une "flèche droite"
EE38-	CA	DEX	décrémente le nombre de caractères requis
EE39-	D0 FA	BNE EE35	reboucle tant que la fin n'est pas atteinte

Remplace les caractères de remplissage par des espaces

EE3B-	06 F3	ASL F3	les bits indiquant les paramètres sélectionnés
EE3D-	06 F3	ASL F3	sont décalés 2 fois à gauche (J passe en b7, K en b6 et E en b5), car S et C ont déjà été traités et peuvent être perdus
EE3F-	A4 F2	LDY F2	nombre de caractères requis
EE41-	88	DEY	que l'on diminue
EE42-	B1 D1	LDA (D1),Y	lit caractère de chaîne en commençant par la fin
EE44-	CD 75 C0	CMP C075	est-ce un caractère de remplissage?

EE47-	D0 18	BNE EE61	sinon, "justification" finie, continue en EE61
EE49-	A9 20	LDA #20	si oui, remplace le caractère lu par un espace
EE4B-	24 F3	BIT F3	teste si **J** ou **K** était demandé (**J** pour justifier la variable alphanumérique en ajoutant des espaces, sans affecter l'écran et **K** pour justifier à l'écran seulement, sans affecter la variable alphanumérique)
EE4D-	10 02	BPL EE51	si **J** non demandé, saute l'instruction suivante
EE4F-	91 D1	STA (D1),Y	écrit un espace à la place du caractère de remplissage
EE51-	50 06	BVC EE59	si **K** non demandé, saute les 2 instructions suivantes
EE53-	20 2A D6	JSR D62A	XAFCAR affiche cet espace (et déplace à droite)
EE56-	20 73 EE	JSR EE73	puis XAF1GAU affiche une "flèche gauche" (revient en place)
EE59-	20 73 EE	JSR EE73	XAF1GAU affiche une "flèche gauche" (caractère précédent)
EE5C-	98	TYA	teste s'il reste des caractères à examiner
EE5D-	D0 E2	BNE EE41	si oui, reboucle en EE41
EE5F-	24 C8	BIT C8	sinon, "justification" finie, continue en EE61

Repositionne en fin de fenêtre

EE60-	C8	INY	point de rebouclage: caractère suivant
EE61-	20 76 EE	JSR EE76	XAF1DR affiche une "flèche droite"
EE64-	C4 F2	CPY F2	tous les caractères ont-ils été examinés?
EE66-	D0 F8	BNE EE60	sinon, reboucle en EE60
EE68-	60	RTS	si oui, retourne à la commande LINPUT. Il aurait été préférable de récupérer les coordonnées d'origine et de repositionner convenablement le curseur au début de la fenêtre en simulant un affichage.

XAFXGAU affiche X fois "flèche gauche"

EE69-	8A	TXA	teste le nombre de caractères à saisir
EE6A-	F0 06	BEQ EE72	simple RTS si 0
EE6C-	20 73 EE	JSR EE73	XAF1GAU affiche une "flèche gauche"
EE6F-	CA	DEX	décrémente le nombre de caractères à saisir
EE70-	D0 F7	BNE EE69	reboucle en EE69 tant qu'il en reste à saisir
EE72-	60	RTS	

XAF1GAU affiche une "flèche gauche"

EE73-	A9 08	LDA #08	A = "flèche gauche"
EE75-	2C A9 09	BIT 09A9	et continue en EE7A

XAF1DR affiche une "flèche droite"

EE76-	A9 09	LDA #09	A = "flèche droite"
EE78-	24 68	BIT 68	et continue en EE7A

Passe la marge en mode 38 colonnes

EE79-	68	PLA	reprend dans A la flèche qui est sur pile

Affiche une "flèche gauche" ou une "flèche droite"

EE7A-	48	PHA	empile "flèche gauche" ou "flèche droite"
EE7B-	20 2A D6	JSR D62A	XAFCAR affiche "flèche gauche" ou "flèche droite"
EE7E-	AD 6A 02	LDA 026A	mode console (ici la flèche de A est perdue)
EE81-	29 20	AND #20	teste b5 (à 0, si mode normal 38 colonnes)
EE83-	D0 07	BNE EE8C	continue en EE8C si mode 40 colonnes
EE85-	AD 69 02	LDA 0269	n° de colonne (abscisse x)
EE88-	29 FE	AND #FE	ET 1111 1110 donne 0 si n° colonne vaut 0 ou 1
EE8A-	F0 ED	BEQ EE79	branche en EE79 si dans la marge
EE8C-	68	PLA	retire "flèche gauche" ou "flèche droite"
EE8D-	60	RTS	de la pile et retourne

XCSTR copie la longueur et l'adresse d'une chaîne alphanumérique (les 3 octets D0, D1 et D2) "dans" la variable BASIC pointée en B8, B9 et BA

EE8E-	A0 02	LDY #02	pour 3 copies d'octets (longueur et adresse chaîne)
EE90-	B9 D0 00	LDA 00D0,Y	lecture octet, en commençant par la fin (D2, D1 et D0)
EE93-	91 B8	STA (B8),Y	écriture dans la variable BASIC (BA, B9 et B8)
EE95-	88	DEY	vise octet précédent
EE96-	10 F8	BPL EE90	reboucle tant que Y est supérieur ou égal à 0
EE98-	60	RTS	

EXÉCUTION DE LA COMMANDE SEDORIC USING

Rappel de la syntaxe

USING expression_numérique,expression_alphanumérique_modèle(,variable_alphanumérique)

L'expression numérique indiquée sera formatée selon la chaîne modèle proposée et le résultat sera soit affiché (par défaut) ou affecté à la variable alphanumérique (s'il y en a une de spécifiée). "ILLEGAL_QUANTITY_ERROR" si le nombre ne peut entrer dans la chaîne modèle proposée.

La chaîne modèle peut utiliser les caractères spéciaux suivants:

+	affiche le signe (affiche donc obligatoirement "+" ou "-")
-	affiche un espace (nombre positif) ou "-" (nombre négatif)
^	affiche l'exposant en notation scientifique (+2 par exemple)
&a	a = caractère de remplissage devant la partie entière (espace par défaut)
%x	x est le nombre (de 0 à 9) de caractères de la partie entière
#x	x est le nombre (de 0 à 9) de caractères de la partie décimale
!x	arrondit au x ème caractère (de 0 à 9) de la partie entière
@x	arrondit au x ème caractère (de 0 à 9) de la partie décimale

Tout autre caractère présent dans la chaîne modèle est reproduit tel quel

Variables utilisées

La commande USING élabore la chaîne formatée en page zéro en utilisant les adresses suivantes (22

octets):

22	longueur de la chaîne modèle
28	flag numérique / alphanumérique
91/92	adresse de la chaîne alphanumérique modèle
C4	signe du nombre
C5/CD	partie entière (9 caractères au maximum)
CE/D6	partie décimale (9 caractères au maximum)
D0/D1/D2	ACC1, longueur et adresse de la chaîne
D7	signe de l'exposant
D8/D9	exposant (2 caractères au maximum)
F2	nombre de caractères dans la partie entière
F4	index dans la chaîne alphanumérique modèle
F5	index dans la chaîne alphanumérique formatée
F6	caractère de tête
0100 et suivants	chaîne décimale du nombre
C100 BUF1	chaîne formatée finale

Analyse la syntaxe et saisit les paramètres

EE99-	20 16 D2	JSR D216	JSR CF17/ROM et CF09/ROM évalue une expression numérique à TXTPTR, retourne avec cette valeur numérique dans ACC1
EE9C-	20 D2 D2	JSR D2D2	E0D5/ROM convertit ACC1 en chaîne décimale AY située à partir de #0100 (qui contient le signe "-" ou un espace) et terminée par #00
EE9F-	20 2C D2	JSR D22C	D067/ROM exige une "," place TXTPTR sur l'octet suivant
EEA2-	20 24 D2	JSR D224	JSR CF17/ROM évalue une expression à TXTPTR, retourne avec une valeur numérique dans ACC1 (et #00 dans 28) ou l'adresse d'une chaîne dans D3/D4 (et #FF dans 28) et A, N et Z positionnés selon l'exposant (résultat numérique) ou la longueur de chaîne
EEA5-	20 74 D2	JSR D274	JSR D7D0 et CF09/ROM vérifie si l'expression évaluée à TXTPTR est bien alphanumérique, retourne l'adresse de la chaîne dans XY et 91/92 ainsi que sa longueur dans A
EEA8-	85 22	STA 22	longueur de la chaîne modèle

Initialise la chaîne formatée

EEAA-	20 CE DA	JSR DACE	XVBUF1 rempli BUF1 de 0
EEAD-	A9 30	LDA #30	"0" (caractère zéro)
EEAF-	A0 2B	LDY #2B	"+"
EEB1-	84 D7	STY D7	signe de l'exposant de la chaîne formatée par défaut
EEB3-	85 D8	STA D8	écrit un zéro en guise de premier chiffre décimal de l'exposant de la chaîne formatée
EEB5-	85 D9	STA D9	idem pour deuxième chiffre
EEB7-	85 C5	STA C5	écrit un zéro après le signe en guise de premier chiffre décimal de la partie entière de la chaîne formatée
EEB9-	A2 09	LDX #09	
EEBB-	95 CD	STA CD,X	écrit 9 zéros dans la partie décimale
EEBD-	CA	DEX	de la chaîne formatée (en CE/D6)

EEBE-	D0 FB	BNE EEBB	reboucle en EEBB tant qu'il en reste (X = 0 à la fin).

Il aurait été plus simple de remplir de “0" la zone C5/D9, puis de mettre le signe “+” en D7, ce qui aurait gagné 6 octets.

Met le signe en place dans la chaîne formatée

EEC0-	AD 00 01	LDA 0100	lit le signe du nombre à formater
EEC3-	C9 2D	CMP #2D	est-ce le signe "-"?
EEC5-	F0 02	BEQ EEC9	si oui, continue en EEC9
EEC7-	A9 2B	LDA #2B	sinon, prend le signe "+"
EEC9-	85 C4	STA C4	met le signe en place dans la chaîne formatée

Initialise pour la partie entière

EECB-	86 F4	STX F4	initialise F4 = #00 = index dans la chaîne modèle
EECD-	86 F5	STX F5	initialise F5 = #00 = index dans la chaîne formatée
EECF-	A9 20	LDA #20	"espace"
EED1-	85 F6	STA F6	met par défaut un espace comme caractère de tête
EED3-	A0 01	LDY #01	
EED5-	84 F2	STY F2	nombre de caractères dans la partie entière
EED7-	88	DEY	Y repasse à zéro
EED8-	2C A2 09	BIT 09A2	continue en EEDB (toujours avec X = #00) pour commencer la construction de la partie entière de la chaîne formatée

Initialise pour la partie décimale

EED9-	A2 09	LDX #09	début de la partie décimale de la chaîne formatée

Sous-programme commun: élaboration de la partie entière ou décimale

EEDB-	C8	INY	vise le caractère suivant
EEDC-	B9 00 01	LDA 0100,Y	lit un caractère de la chaîne non formatée
EEDF-	F0 25	BEQ EF06	continue en EF06 s'il n'y en a plus
EEE1-	C9 2E	CMP #2E	est-ce le "." décimal?
EEE3-	F0 F4	BEQ EED9	si oui, reprend en EED9 pour élaborer la partie décimale
EEE5-	C9 45	CMP #45	est-ce un "E"? (début de l'exposant)
EEE7-	F0 0B	BEQ EEF4	si oui, continue en EEF4 pour élaborer l'exposant
EEE9-	95 C5	STA C5,X	sinon, copie le caractère dans la partie entière (ou décimale, selon X) de la chaîne formatée
EEEB-	E0 09	CPX #09	teste si X vise dans la partie décimale
EEED-	B0 02	BCS EEF1	si oui, saute l'instruction suivante
EEEF-	84 F2	STY F2	met à jour le compte de caractères de la partie entière
EEF1-	E8	INX	vise l'octet suivant de la chaîne formatée
EEF2-	D0 E7	BNE EEDB	reprise forcée en EEDB

Elabore l'exposant

EEF4-	B9 01 01	LDA 0101,Y	lit le signe de l'exposant
EEF7-	85 D7	STA D7	et l'écrit en D7 dans la chaîne formatée
EEF9-	B9 02 01	LDA 0102,Y	lit le premier chiffre décimal de l'exposant
EEFC-	AA	TAX	et le passe dans X
EEFD-	B9 03 01	LDA 0103,Y	lit le deuxième chiffre de l'exposant, le garde dans A
EF00-	F0 02	BEQ EF04	s'il n'y en a pas, saute l'instruction suivante
EF02-	85 D9	STA D9	met en place le deuxième chiffre de l'exposant
EF04-	86 D8	STX D8	met en place le premier chiffre de l'exposant

Justifie à droite la partie entière

EF06-	A6 F2	LDX F2	longueur de la partie entière de la chaîne formatée (soit 8 au maximum)
EF08-	A0 08	LDY #08	vise le dernier caractère de la partie entière de la chaîne formatée
EF0A-	B5 C4	LDA C4,X	lit un caractère de la partie entière de la chaîne formatée
EF0C-	CA	DEX	décrémente le nombre de caractères de la partie entière de la chaîne formatée
EF0D-	10 02	BPL EF11	saute l'instruction suivante tant qu'il reste des caractères de la partie entière de la chaîne formatée à écrire
EF0F-	A9 20	LDA #20	remplace par un espace dès que tous les caractères significatifs de la partie entière de la chaîne formatée ont été écrits
EF11-	99 C5 00	STA 00C5,Y	justifie à droite la partie entière
EF14-	88	DEY	vise l'emplacement précédent
EF15-	10 F3	BPL EF0A	reboucle en EF0A pour déplacer si possible tous les caractères à droite
EF17-	2C 84 F5	BIT F584	continue en EF1A

Point de rebouclage lors du formatage

EF18-	84 F5	STX F5	met à jour l'index de la chaîne produite

Y a t-il un exposant à formater?

EF1A-	A4 F4	LDY F4	index de la chaîne modèle
EF1C-	C4 22	CPY 22	teste si la fin de la chaîne modèle est atteinte
EF1E-	D0 28	BNE EF48	sinon, continue en EF48
EF20-	A9 00	LDA #00	si oui,
EF22-	85 D7	STA D7	force D7 à zéro (signe de l'exposant)

Teste si une variable a été indiquée

EF24-	20 9E D3	JSR D39E	XCRGOT relit le caractère à TXTPTR (sans incrémenter TXTPTR = CHRGOT), puis le convertit en MAJUSCULE, les espaces sont sautés, Z = 1 si fin d'instruction (0 ou :), C = 0 si caractère chiffre 0 à 9 (soit #30 à #39), sinon C = 1, Y et X inchangés
EF27-	F0 18	BEQ EF41	si fin de la commande, il n'y a pas de variable, termine en affichant la chaîne formatée finale
EF29-	A5 F5	LDA F5	longueur de la chaîne produite
EF2B-	20 64 D2	JSR D264	JSR D5AB/ROM réserve une place en mémoire pour la chaîne de longueur A, retourne avec cette longueur en D0 et l'adresse de la chaîne en D1/D2
EF2E-	A8	TAY	longueur de la chaîne

EF2F-	88	DEY	
EF30-	B9 00 C1	LDA C100,Y	copie la chaîne dans la place réservée
EF33-	91 D1	STA (D1),Y	
EF35-	98	TYA	
EF36-	D0 F7	BNE EF2F	
EF38-	20 2C D2	JSR D22C	D067/ROM exige une "," place TXTPTR sur l'octet suivant
EF3B-	20 38 D2	JSR D238	JSR D188/ROM décode le nom de la première variable à TXTPTR et place "l'adresse" de cette variable dans AY, B6/B7 et D3/D4, cette adresse pointe en fait sur les data (longueur/adresse dans le cas d'une chaîne) de la variable dans la zone des variables BASIC
EF3E-	4C D6 E8	JMP E8D6	XMAJSTR copie l'adresse et la longueur d'une chaîne alphanumérique (les 3 octets D0, D1 et D2) en B6, B7 et B8 et retourne

Termine en affichant la chaîne formatée produite

EF41-	A9 00	LDA #00	AY, adresse de début de BUF1, c'est à dire
EF43-	A0 C1	LDY #C1	adresse de la chaîne produite
EF45-	4C 37 D6	JMP D637	XAFSTR affiche chaîne terminée par 0 d'adresse AY

Suite de l'analyse de la chaîne modèle

Au retour, A = caractère, Y = index dans la chaîne formatée et X = #FF

EF48-	20 2B F0	JSR F02B	lit un caractère de la chaîne modèle
EF4B-	C9 5E	CMP #5E	est-ce un "^" (exposant)?
EF4D-	D0 19	BNE EF68	sinon, suite de l'analyse en EF68

Affiche l'exposant en notation scientifique

EF4F-	A2 FD	LDX #FD	si oui, lit un caractère de l'exposant de la chaîne
EF51-	BD DA FF	LDA FFDA,X	non formatée en FFDA + FD = 00D7 (c'est le signe)
EF54-	2C A9 20	BIT 20A9	continue en EF5A
EF55-	A9 20	LDA 20	"espace"
EF57-	2C A5 C4	BIT C4A5	continue en EF5A
EF58-	A5 C4	LDA C4	signe du nombre

Ecrit le caractère dans la chaîne formatée finale

EF5A-	99 00 C1	STA C100,Y	élabore la chaîne formatée finale dans BUF1
EF5D-	C8	INY	vise le suivant (cible)
EF5E-	D0 03	BNE EF63	saute l'instruction suivante tant que la chaîne produite n'atteint pas 256 caractères
EF60-	4C 77 E9	JMP E977	sinon, "STRING_TOO_LONG_ERROR"
EF63-	E8	INX	vise le suivant (source)
EF64-	D0 EB	BNE EF51	reboucle en EF51 tant que l'exposant n'est pas fini (c'est assez génial)
EF66-	F0 B0	BEQ EF18	reboucle en EF18 pour continuer le formatage

Suite de l'analyse de la chaîne modèle

EF68-	C9 2B	CMP #2B	est-ce un "+"?
EF6A-	F0 EC	BEQ EF58	si oui, reboucle en EF58
EF6C-	C9 2D	CMP #2D	est-ce un "-"?
EF6E-	D0 08	BNE EF78	sinon, suite de l'analyse en EF78
EF70-	AD 00 01	LDA 0100	si oui, lit le signe du nombre
EF73-	4A	LSR	teste le b0 en le poussant dans C
EF74-	B0 E2	BCS EF58	reboucle en EF58 si négatif (insère un "-")
EF76-	90 DD	BCC EF55	reboucle en EF55 si positif (insère un "espace")

Suite de l'analyse de la chaîne modèle

EF78-	C9 23	CMP #23	est-ce un "#"?
EF7A-	D0 07	BNE EF83	sinon, suite de l'analyse en EF83

Détermine le nombre de caractères de la partie décimale

EF7C-	20 A7 EF	JSR EFA7	lit le nombre de caractères de la partie décimale de la chaîne modèle et le met en F3
EF7F-	A2 09	LDX #09	pour 9 caractères
EF81-	D0 10	BNE EF93	suite forcée en EF93 pour élaborer la partie décimale formatée

Suite de l'analyse de la chaîne modèle

EF83-	C9 25	CMP #25	est-ce un "%"?
EF85-	D0 32	BNE EFB9	sinon, suite de l'analyse en EFB9

Détermine le nombre de caractères de la partie entière

EF87-	20 A7 EF	JSR EFA7	lit le nombre de caractères de la partie entière de la chaîne modèle et le met en F3
EF8A-	C5 F2	CMP F2	est-ce aussi long que ce que nous avons déjà?
EF8C-	90 25	BCC EFB3	sinon, "ILLEGAL_QUANTITY_ERROR"
EF8E-	A9 09	LDA #09	longueur maximum
EF90-	E5 F3	SBC F3	longueur que doit avoir la partie entière
EF92-	AA	TAX	index où commencer dans la chaîne non formatée

Elabore la partie décimale de la chaîne formatée

EF93-	C6 F3	DEC F3	décrémente le compteur
EF95-	10 03	BPL EF9A	saute l'instruction suivante pour continuer à élaborer la chaîne formatée
EF97-	4C 18 EF	JMP EF18	pas de partie entière
EF9A-	B5 C5	LDA C5,X	lit un chiffre de la partie entière
EF9C-	29 7F	AND #7F	en force le b7 à zéro
EF9E-	99 00 C1	STA C100,Y	l'écrit dans la chaîne formatée en cours
EFA1-	C8	INY	vise le suivant (cible)
EFA2-	F0 12	BEQ EFB6	"STRING_TOO_LONG_ERROR" si > 256 caractères

```
EFA4-  E8        INX       vise le suivant (source)
EFA5-  D0 EC     BNE EF93  reprise forcée en EF93
```

Lit un nombre (de 0 à 9) qui suit les signes %, #, ! ou @ dans la chaîne modèle et le met en F3

```
EFA7-  20 2B F0  JSR F02B  lit un caractère dans la chaîne modèle
EFAA-  E9 30     SBC #30   convertit le code ASCII en chiffre de 0 à 9
EFAC-  85 F3     STA F3    et sauve ce nombre en F3
EFAE-  C9 0A     CMP #0A   teste si < 10
EFB0-  B0 01     BCS EFB3  sinon, "ILLEGAL_QUANTITY_ERROR"
EFB2-  60        RTS       et retourne

EFB3-  4C 20 DE  JMP DE20  "ILLEGAL_QUANTITY_ERROR"

EFB6-  4C 77 E9  JMP E977  "STRING_TOO_LONG_ERROR"
```

Suite de l'analyse de la chaîne modèle

```
EFB9-  C9 21     CMP #21   est-ce un "!"?
EFBB-  D0 3F     BNE EFFC  sinon, suite de l'analyse en EFFC
```

Arrondit la partie entière

```
EFBD-  20 A7 EF  JSR EFA7  lit un caractère de la partie décimale de la chaîne modèle et le met en F3
EFC0-  38        SEC       prépare une soustraction
EFC1-  A9 09     LDA #09
EFC3-  E5 F3     SBC F3
```

Entrée secondaire "Arrondit la partie décimale"

```
EFC5-  85 F3     STA F3    index où commencer
EFC7-  AA        TAX       index de lecture dans la partie entière
EFC8-  B5 C5     LDA C5,X  lit un caractère dans la partie entière
EFCA-  C5 F6     CMP F6    est-ce un caractère de tête (non significatif)
EFCC-  F0 5A     BEQ F028  si oui, le nombre de chiffres de la partie entière est inférieur au nombre
                           requis, reprend le formatage en EF1A
EFCE-  A9 30     LDA #30   vide le reste de la partie entière et toute la partie décimale avec des "0"
EFD0-  E8        INX       pour en garder un (celui qui sera arrondi)
EFD1-  E8        INX       suivant
EFD2-  E0 12     CPX #12   teste si c'est fini
EFD4-  F0 04     BEQ EFDA  si oui, continue en EFDA
EFD6-  95 C5     STA C5,X  sinon, écrit un "0"
EFD8-  D0 F7     BNE EFD1  reboucle en EFD1 tant qu'il en reste
EFDA-  A6 F3     LDX F3    nombre de chiffres requis
EFDC-  E8        INX       pour viser le caractère le moins significatif
EFDD-  B5 C5     LDA C5,X  lit le chiffre à arrondir
EFDF-  C9 35     CMP #35   est-il inférieur à 5?
EFE1-  A9 30     LDA #30   "0"
```

EFE3-	95 C5	STA C5,X	écit un zéro à la place du chiffre arrondi
EFE5-	90 41	BCC F028	si < "5", il n'y a pas à arrondir, on le garde tel quel, continue le formatage en EF1A

Il faut arrondir

EFE7-	CA	DEX	ajuste l'index de lecture
EFE8-	30 3E	BMI F028	continue le formatage en EF1A si terminé
EFEA-	B5 C5	LDA C5,X	lit un chiffre de la partie entière en commençant par le moins significatif
EFEC-	C5 F6	CMP F6	est-ce un caractère de tête (non significatif)
EFEE-	D0 04	BNE EFF4	sinon, saute les deux instructions suivantes
EFF0-	E6 F2	INC F2	incrémente le nombre de caractères de la partie entière, puisque celle-ci sera maintenant étendue
EFF2-	A9 30	LDA #30	commence avec un nouveau zéro de tête pour arrondir
EFF4-	C9 39	CMP #39	est-ce un "9"?
EFF6-	F0 E9	BEQ EFE1	si oui, reprend en EFE1 (écrit un "0" et continue pour arrondir le chiffre suivant)
EFF8-	69 01	ADC #01	si ce n'est pas un "9", on se contente de l'incrémenter
EFFA-	90 E7	BCC EFE3	reprise forcée en EFE3 (écrit le chiffre incrémenté et continue pour arrondir le chiffre suivant)

Suite de l'analyse de la chaîne modèle

EFFC-	C9 40	CMP #40	est-ce un "@"?
EFFE-	D0 07	BNE F007	sinon, suite de l'analyse en F007

Arrondit la partie décimale

F000-	20 A7 EF	JSR EFA7	lit un caractère de la partie décimale de la chaîne modèle et le met en F3
F003-	69 08	ADC #08	ajuste l'offset pour viser la partie décimale
F005-	90 BE	BCC EFC5	reprise forcée en EFC5 pour arrondir

Suite de l'analyse de la chaîne modèle

F007-	C9 26	CMP #26	est-ce un "&"?
F009-	F0 03	BEQ F00E	si oui, continue en F00E
F00B-	4C 5A EF	JMP EF5A	tous les autres caractères sont insérés tels quels dans la chaîne formatée finale

Caratère pour remplacer les zéros en tête

F00E-	20 2B F0	JSR F02B	lit un caractère de la chaîne modèle
F011-	C9 30	CMP #30	est-ce un "0"?
F013-	D0 02	BNE F017	sinon, saute l'instruction suivante
F015-	09 80	ORA #80	si oui, en marque le b7 pour le repérer
F017-	AA	TAX	garde le caractère de remplissage dans X
F018-	A0 00	LDY #00	vise tout au début de la chaîne
F01A-	B9 C5 00	LDA 00C5,Y	lit un caractère

```
F01D-  C5 F6      CMP F6      est-ce un caractère non significatif de tête?
F01F-  D0 05      BNE F026    sinon, pas de convertion, continue en F026
F021-  96 C5      STX C5,Y    si oui, on le remplace par le caractère remplissage
F023-  C8         INY         vise le suivant
F024-  D0 F4      BNE F01A    reprise forcée en F01A
F026-  86 F6      STX F6      sauve le nouveau caractère à placer en tête
F028-  4C 1A EF   JMP EF1A    reprend le formatage
```

Lit un caractère de la chaîne modèle

```
F02B-  A4 F4      LDY F4      récupère l'index de la chaîne modèle
F02D-  B1 91      LDA (91),Y  lit un caractère de la chaîne modèle
F02F-  E6 F4      INC F4      vise le caractère suivant
F031-  A4 F5      LDY F5      récupère l'index de la chaîne formatée
F033-  A2 FF      LDX #FF     réinitialise index X
F035-  60         RTS
```

EXÉCUTION DE LA COMMANDE SEDORIC LUSING

Rappel de la syntaxe

LUSING expression_numérique,expression_alphanumérique_modèle(,variable_alphanumérique)

L'expression numérique indiquée sera formatée selon la chaîne modèle proposée et le résultat sera soit imprimé (par défaut) ou affecté à la variable alphanumérique (s'il y en a une de spécifiée). "ILLEGAL_QUANTITY_ERROR" si le nombre ne peut entrer dans la chaîne modèle proposée. A part la sortie sur imprimante, cette commande est identique à USING (EE99), notamment en ce qui concerne les caractères spéciaux de formatage (voir plus haut).

```
F036-  20 C5 E7   JSR E7C5    PR SET    voici donc un moyen simple de convertir
F039-  20 99 EE   JSR EE99    USING     toute commande d'affichage
F03C-  4C D6 E7   JMP E7D6    PR OFF    en commande d'impression
```

COMMANDES SEDORIC LINE ET BOX

Entrées de la commande LINE en F079 et de la commande BOX en F0DE

Convertit AN en radians, résultat en BFE0/BFE4, place -PI/2 en BFEA/BFEE

Attention, cette routine utilise la zone BFE0 à BFFF en RAM. Ceci est un choix malheureux, quasiment assimilable à une bogue, car de nombreux programmes utilisent cette zone pour loger une petite routine en langage machine. Toute utilisation des commandes LINE et BOX entraînera donc l'écrasement de la routine. Il y a gros a parier que l'utilisateur ne comprendra pas ce qui lui arrive!

```
F03F-  A2 05      LDX #05     pour copier 5 octets
F041-  BD 1A CD   LDA CD1A,X  de CD1B/CD1F en BFE0/BFE4
```

F044-	9D DF BF	STA BFDF,X	(7B, 0E, FA, 35 et 10 soit PI/180)
F047-	BD 1F CD	LDA CD1F,X	et de CD20/CD24 en BFEA/BFEE
F04A-	9D E9 BF	STA BFE9,X	(81, C9, 0F, DA et A2 soit -PI/2
F04D-	CA	DEX	
F04E-	D0 F1	BNE F041	reboucle en F041 tant qu'il en reste à copier
F050-	E8	INX	qui repasse à un
F051-	8E 72 C0	STX C072	force à 1 le b0 pour LINE et BOX (flag pour lecture du code fb)
F054-	A9 41	LDA #41	"A"
F056-	A0 4E	LDY #4E	"N"
F058-	85 B4	STA B4	initialise le nom de variable AN
F05A-	84 B5	STY B5	
F05C-	20 44 D2	JSR D244	JSR D1E8/ROM cherche l'adresse de la valeur de la variable dont les 2 caractères significatifs sont en B4/B5 revient avec cette adresse dans AY et B6/B7 (crée la variable avec une valeur nulle si elle n'existe pas encore) (la valeur d'une chaîne est remplacée par sa longueur et son adresse)
F05F-	20 BA D2	JSR D2BA	JSR DE7B/ROM place dans ACC1 (floating point accumulator) la valeur pointée par AY
F062-	A9 E0	LDA #E0	adresse de la constante PI/180
F064-	A0 BF	LDY #BF	soit BFE0
F066-	20 AA D2	JSR D2AA	JSR DCED/ROM multiplie le contenu de ACC1 (floating point accumulator) par la valeur pointée par AY et replace le résultat dans ACC1
F069-	A2 E0	LDX #E0	adresse où mettre le résultat
F06B-	A0 BF	LDY #BF	soit BFE0
F06D-	20 C2 D2	JSR D2C2	JSR DEAD/ROM recopie les 5 octets de ACC1 de l'adresse XY à l'adresse XY + 4

XVERHRS vérifie si on est bien en mode HIRES

Sinon génère une "DISP_TYPE_MISMATCH_ ERROR", réinitialise la pile et retourne au "Ready".

F070-	AD 1F 02	LDA 021F	teste si mode HIRES
F073-	D0 03	BNE F078	si oui, tout va bien, simple RTS en F078
F075-	4C 6F D1	JMP D16F	sinon, JSR C47E/ROM affiche le message "DISP_TYPE_MISMATCH" puis effectue un JSR C496/ROM qui réinitialise la pile, affiche "ERROR" et retourne au "Ready"

F078- 60 RTS

EXÉCUTION DE LA COMMANDE SEDORIC LINE

Rappel de la syntaxe

LINE longueur_en_nombre_de_points,code_foreground/background

Trace une ligne de la longueur indiquée, à partir de la position courante du curseur HIRES, dans la direction indiquée par la valeur courante de la variable AN (en degrés selon la convention courante trigonométrique).

La valeur initiale de AN doit être fixée par l'utilisateur (exemple AN = 90 pour se diriger vers le haut). Les coordonnées initiales du curseur HIRES sont également à la charge de l'utilisateur (exemple CURSET 119,99 pour se placer au centre de l'écran).

Les coordonnées finales du curseur sont mises à jour automatiquement.

Le code foreground/background est celui du BASIC: la ligne est tracée selon PAPER si 0, selon INK si 1, en inversion vidéo si 2 et enfin seule la position du curseur HIRES est modifiée (sans traçage réel de ligne) si 3.

Cette commande utilise le DRAW du BASIC dont la syntaxe est:

DRAW delta_x,delta_y,fb

avec:

delta_x	déplacement relatif le long des abscisses (0 à ±199),
delta_y	déplacement relatif le long des ordonnées (0 à ±239) et
fb	code foreground/background (0 à 3).

DRAW trace un trait de coordonnées relatives à partir de la position actuelle du curseur HIRES.

"ILLEGAL_QUANTITY_ERROR" si DRAW sort de l'écran (x de 0 à 239 et y de 0 à 199), d'où l'intérêt d'utiliser la commande HCUR et de bien contrôler son travail.

Non documenté

Les commandes LINE et BOX utilisent la zone BFE0/BFFF RAM située entre la zone de l'écran et celle de la ROM/RAM overlay. Attention donc aux routines en "Langage Machine" qui pourraient utiliser cette zone!

F079-	20 3F F0	JSR F03F	convertit la valeur de AN en radians, place le résultat en BFE0/BFE4, place -PI/2 en BFEA/BFEE et force à 1 le b0 de C072 pour compter les paramètres de DRAW
F07C-	20 16 D2	JSR D216	JSR CF17/ROM et CF09/ROM évalue une expression numérique à TXTPTR, retourne avec cette valeur numérique dans ACC1 (nombre de points)
F07F-	A2 E5	LDX #E5	pour mettre le résultat en BFE5/BFE9
F081-	A0 BF	LDY #BF	
F083-	20 C2 D2	JSR D2C2	JSR DEAD/ROM recopie les 5 octets de ACC1 de l'adresse XY à l'adresse XY + 4
F086-	A2 00	LDX #00	index pour les paramètres DRAW
F088-	86 F2	STX F2	les paramètres delta_x et delta_y pour DRAW sont calculés à partir du premier paramètre de LINE (longueur de la ligne en points)
F08A-	A9 E0	LDA #E0	
F08C-	A0 BF	LDY #BF	AY pointe sur la valeur de AN en radians
F08E-	20 BA D2	JSR D2BA	JSR DE7B/ROM place dans ACC1 (floating point accumulator) la valeur pointée par AY
F091-	A6 F2	LDX F2	index pour les paramètres DRAW

```
F093-  F0 09      BEQ F09E      suite forcée en F09E si c'est le premier
```

Calcule le deuxième paramètre de DRAW

```
F095-  20 F2 D2   JSR D2F2      JSR E392/ROM calcule SIN(AN) dans ACC1
F098-  20 DA D2   JSR D2DA      JSR E271/ROM changement de signe (echelle y inversée)
F09B-  4C A1 F0   JMP F0A1      saute l'instruction suivante
```

Calcule le premier paramètre de DRAW

```
F09E-  20 EA D2   JSR D2EA      JSR E38B/ROM calcule COS(AN) dans ACC1
```

Calcule la longueur de la projection de la ligne (delta_x et delta_y) sur chacun des axes

```
F0A1-  A9 E5      LDA #E5
F0A3-  A0 BF      LDY #BF       AY pointe sur la longueur de la ligne
F0A5-  20 AA D2   JSR D2AA      JSR DCED/ROM multiplie le contenu de ACC1 (floating point accumulator)
                                par la valeur pointée par AY et replace le résultat dans ACC1 qui contient
                                maintenant soit delta_x soit delta_y
F0A8-  20 8A D2   JSR D28A      JSR D926/ROM convertit le nombre présent dans ACC1 en entier signé dans
                                YA
F0AB-  AA         TAX           YA devient YX et teste HH, c'est à dire A
F0AC-  F0 04      BEQ F0B2      suite directe en F0B2 si HH nul
```

Incrémente le paramètre si HH non nul (delta négatif)

```
F0AE-  C8         INY
F0AF-  D0 01      BNE F0B2
F0B1-  E8         INX           YX = YX + 1
```

Met en place les paramètres de DRAW

```
F0B2-  8A         TXA           YX redevient YA (delta_x ou delta_y)
F0B3-  A6 F2      LDX F2        index pour les paramètres DRAW
F0B5-  9D E2 02   STA 02E2,X
F0B8-  98         TYA           copie YA en 02E1/02E2 (delta_x)
F0B9-  9D E1 02   STA 02E1,X    ou en 02E3/02E4 (delta_y)
F0BC-  E8         INX
F0BD-  E8         INX           index + 2 pour les paramètres DRAW
F0BE-  E0 02      CPX #02       teste si c'est delta_x qui vient d'être fait
F0C0-  F0 C6      BEQ F088      si oui, reboucle en F088 pour calculer delta_y
```

Teste si LINE (ou première LINE de BOX)

```
F0C2-  4E 72 C0   LSR C072      teste le b0 du flag C072 (n'est utile que pour BOX)
F0C5-  90 0C      BCC F0D3      continue directement en F0D3 avec la commande DRAW, s'il n'y a pas de
                                paramètre foreground/background à lire (cas de BOX, qui appelle le
                                sous-programme LINE 4 fois, le paramètre foreground/background n'a
```

besoin d'être lu qu'une seule fois)

Lit le paramètre foreground/background pour LINE ou premier appel de LINE par BOX

F0C7-	20 2C D2	JSR D22C	D067/ROM exige une "," place TXTPTR sur l'octet suivant
F0CA-	20 FA D2	JSR D2FA	E853/ROM évalue un nombre non signé à TXTPTR et revient avec ce nombre dans YA et mise à jour de TXTPTR sur l'octet suivant ce nombre
F0CD-	8C E5 02	STY 02E5	place AY (paramètre foreground/background)
F0D0-	8D E6 02	STA 02E6	en 02E5/02E6

Appelle la commande DRAW du BASIC

F0D3-	20 12 D3	JSR D312	JSR F110/ROM commande "DRAW"
F0D6-	4E E0 02	LSR 02E0	teste le b0 du flag "Graphic error"
F0D9-	90 9D	BCC F078	simple RTS en F078 si pas d'erreur
F0DB-	4C 7C E9	JMP E97C	sinon, "ILLEGAL_QUANTITY_ERROR"

EXÉCUTION DE LA COMMANDE SEDORIC BOX

Rappel de la syntaxe

BOX longueur_coté_1,longueur_coté_2,code_foreground/background

Trace un rectangle en appelant 4 fois la commande LINE: trace d'abord le premier coté selon la position courante du curseur HIRES, l'angle AN courant, la longueur indiquée pour le premier coté, fait pivoter AN de -90°, trace le deuxième coté selon la longueur indiquée pour le deuxième coté, fait pivoter AN de -90°, trace le troisième coté selon la longueur indiquée pour le premier coté, fait pivoter AN de -90°, trace le dernier coté selon la longueur indiquée pour le deuxième coté, fait pivoter AN de -90° et termine avec les mêmes positions de curseur et d'angle qu'au départ. Le rectangle est donc tracé dans le sens des aiguilles d'une montre (sens trigonométrique négatif). Voir aussi la commande LINE.

Non documenté

Les commandes LINE et BOX utilisent la zone BFE0/BFFF RAM située entre la zone de l'écran et celle de la ROM/RAM overlay. Attention donc aux routines en "Langage Machine" qui pourraient utiliser cette zone!

F0DE-	20 3F F0	JSR F03F	convertit la valeur de AN en radians, place le résultat en BFE0/BFE4, place -PI/2 en BFEA/BFEE et force à 1 le b0 de C072 pour compter les paramètres de DRAW
F0E1-	20 7F D2	JSR D27F	CF17/ROM et CF09/ROM et D8CB/ROM évalue un nombre entier à TXTPTR et le retourne dans X (longueur du premier coté)
F0E4-	86 F3	STX F3	sauve le premier paramètre dans F3
F0E6-	20 2C D2	JSR D22C	D067/ROM exige une "," place TXTPTR sur l'octet suivant
F0E9-	20 7F D2	JSR D27F	CF17/ROM et CF09/ROM et D8CB/ROM évalue un nombre entier à TXTPTR et le retourne dans X (longueur du deuxième coté)
F0EC-	86 F4	STX F4	sauve le deuxième paramètre dans F4

F0EE-	A9 04	LDA #04	pour dessiner 4 cotés
F0F0-	85 F5	STA F5	compteur de cotés restant à dessiner
F0F2-	A9 00	LDA #00	
F0F4-	85 F6	STA F6	pour commencer avec le premier paramètre de BOX
F0F6-	A6 F6	LDX F6	copié dans X pour usage prochain
F0F8-	8A	TXA	
F0F9-	49 01	EOR #01	bascule le b0 de F6 (flag pour l'autre paramètre)
F0FB-	85 F6	STA F6	et le remet en place
F0FD-	B4 F3	LDY F3,X	lit la longueur du premier ou du deuxième coté
F0FF-	A9 00	LDA #00	force à zéro le HH de cette longueur
F101-	20 CA D2	JSR D2CA	JSR DF40/ROM transfère un nombre non signé YA dans ACC1 (floating point accumulator)
F104-	20 7F F0	JSR F07F	appelle LINE pour dessiner un coté du rectangle selon la longueur indiquée dans ACC1, l'angle courant indiqué en BFE0/BFE4 (le paramètre foreground/background n'est lu que lors du premier appel à la commande LINE)

Pivote AN de -90°

F107-	A9 E0	LDA #E0	
F109-	A0 BF	LDY #BF	valeur actuelle de AN
F10B-	20 BA D2	JSR D2BA	JSR DE7B/ROM place dans ACC1 (floating point accumulator) la valeur pointée par AY
F10E-	A9 EA	LDA #EA	
F110-	A0 BF	LDY #BF	valeur de -90°
F112-	20 A2 D2	JSR D2A2	JSR DB22/ROM additionne le contenu de ACC1 (floating point accumulator) et la valeur pointée par AY et replace le résultat dans ACC1
F115-	A2 E0	LDX #E0	
F117-	A0 BF	LDY #BF	nouvelle valeur de AN
F119-	20 C2 D2	JSR D2C2	JSR DEAD/ROM recopie les 5 octets de ACC1 de l'adresse XY à l'adresse XY + 4
F11C-	C6 F5	DEC F5	compteur de cotés restant à dessiner
F11E-	D0 D6	BNE F0F6	reprend en F0F6 s'il en reste à tracer
F120-	60	RTS	

COMMANDES SEDORIC FAISANT APPEL A UNE BANQUE EXTERNE

Les entrées des commandes SEDORIC faisant appel à une BANQUE externe se font de F121 à F16A (exemples: VUSER en F121 ou INIT en F169). De F121 à F15D, X et Y sont initialisés, pour chaque commande, selon le tableau ci-dessous:

COMMANDE	BANQUE	X	Y	Adresse dans la BANQUE
BACKUP	2	#47	#03	C404
CHANGE	3	#4C	#06	C407
COPY	4	#51	#03	C404
DELETE	1	#42	#06	C407
DKEY	5	#56	#18	C419
DNAME	5	#56	#06	C407
DNUM	5	#56	#12	C413
DSYS	5	#56	#15	C416
DTRACK	5	#56	#09	C40A
INIST	5	#56	#0F	C410
INIT	6	#5B	(pas de valeur Y pour INIT)	
MERGE	3	#4C	#09	C40A
MOVE	1	#42	#09	C40A
RENUM	1	#42	#03	C404
SEEK	3	#4C	#03	C404
SYS	5	#56	#03	C404
TRACK	5	#56	#0C	C40D
VUSER	5	#56	#1B	C41C

NB: X = #01 si la BANQUE est occupée par un masque WINDOW

Y représente l'octet de poids faible LL de l'adresse d'exécution "-1" de la commande dans la BANQUE. L'octet de poids fort HH est toujours #C4 par défaut.

X représente l'endroit de la disquette Master où il faudra lire la BANQUE voulue. C'est une sorte de n° de BANQUE qui est ainsi codé grâce à X = #3D + (5 x n°). La BANQUE n°1 commence donc au secteur #3D + 5 = #42, c'est à dire au soixante sixième secteur. Si la disquette est formatée en 16 secteurs/piste, le début de cette BANQUE se trouve au secteur n°2 de la piste n°4 (cinquième piste) car 66 = (16 x 5) + 2.

Sauf pour INIT, le programme se poursuit avec le sous-programme F15E/F168 où l'adresse d'exécution de la commande dans la BANQUE est empilée (exemple C41B pour VUSER). Dans le cas de INIT, le programme continue directement en F169. Ceci illustre donc deux manières de procéder pour exécuter une commande sur une BANQUE externe.

Puis X ("numéro" de la BANQUE requise) est comparé avec C015 ("numéro" de la BANQUE en place). S'ils sont identiques, suite en F1B9; sinon, il faut d'abord mettre en place la BANQUE désirée (en RAM overlay de C400 à C7FF), puis suite en F16B.

COMMANDES SEDORIC FAISANT APPEL A LA BANQUE EXTERNE n°7

Certaines commandes SEDORIC qui, dans les versions précédentes, se trouvaient dans le NOYAU ont été déplacée dans une BANQUE externe, nouvellement crée à cette occasion. C'est le cas des commandes EXT, PROT, STATUS, SYSTEM et UNPROT. Deux autres commandes ont été également ajoutées à cette nouvelle BANQUE n°7, ce sont CHKSUM et VISUHIRES. L'entrée de ces commandes dans le NOYAU se fait maintenant aux adresses suivantes (voir détails en E9ED):

Commande	Ancienne Entrée	Nouvelle Entrée	X	Y	Adresse dans Banque n°7
CHKSUM	néant	E9FF	60	79	C47A
EXT	E9ED	E9ED	60	03	C404
PROT	E6D0	E9F6	60	57	C458
STATUS	E62E	E9F3	60	54	C455
SYSTEM	E702	E9FC	60	5D	C45E
UNPROT	E6D3	E9F9	60	5A	C45B
VISUHIRES	néant	E9F0	60	51	C452

Comme l'indique ce tableau, la BANQUE n°7 se trouve à partir du #60 (quatre vingt seizième) secteur de la disquette MASTER (#3D + (3 x 7)). Voir en annexe les tableaux montrant l'emplacement piste / secteur des BANQUES sur la disquette MASTER en fonction du type de formatage.

EXÉCUTION DE LA COMMANDE SEDORIC VUSER

F121- A0 1B LDY #1B octet de poids faible de l'adresse d'exécution
F123- 2C A0 18 BIT 18A0 -> continue en #F132

EXÉCUTION DE LA COMMANDE SEDORIC DKEY

F124- A0 18 LDY #18 octet de poids faible de l'adresse d'exécution
F126- 2C A0 15 BIT 15A0 -> continue en #F132

EXÉCUTION DE LA COMMANDE SEDORIC DSYS

F127- A0 15 LDY #15 octet de poids faible de l'adresse d'exécution
F129- 2C A0 12 BIT 12A0 -> continue en #F132

EXÉCUTION DE LA COMMANDE SEDORIC DNUM

F12A- A0 12 LDY #12 octet de poids faible de l'adresse d'exécution

F12C- 2C A0 0F BIT 0FA0 -> continue en #F132

EXÉCUTION DE LA COMMANDE SEDORIC INIST

F12D- A0 0F LDY #0F octet de poids faible de l'adresse d'exécution
F12F- 2C A0 0C BIT 0CA0 -> continue en #F132

EXÉCUTION DE LA COMMANDE SEDORIC TRACK

F130- A0 0C LDY #0C octet de poids faible de l'adresse d'exécution
F132- A2 56 LDX #56 position de la BANQUE sur la disquette MASTER
F134- D0 28 BNE F15E -> branchement forcé en #F15E

EXÉCUTION DE LA COMMANDE SEDORIC MOVE

F136- A2 42 LDX #42 position de la BANQUE sur la disquette MASTER
F138- 2C A2 56 BIT 56A2 -> continue en #F13E

EXÉCUTION DE LA COMMANDE SEDORIC DTRACK

F139- A2 56 LDX #56 position de la BANQUE sur la disquette MASTER
F13B- 2C A2 4C BIT 4CA2 -> continue en #F13E

EXÉCUTION DE LA COMMANDE SEDORIC MERGE

F13C- A2 4C LDX #4C position de la BANQUE sur la disquette MASTER
F13E- A0 09 LDY #09 octet de poids faible de l'adresse d'exécution
F140- D0 1C BNE F15E -> branchement forcé en #F15E

EXÉCUTION DE LA COMMANDE SEDORIC DELETE

F142- A2 42 LDX #42 position de la BANQUE sur la disquette MASTER
F144- 2C A2 56 BIT 56A2 -> continue en #F14A

EXÉCUTION DE LA COMMANDE SEDORIC DNAME

F145- A2 56 LDX #56 position de la BANQUE sur la disquette MASTER
F147- 2C A2 4C BIT 4CA2 -> continue en #F14A

EXÉCUTION DE LA COMMANDE SEDORIC CHANGE

F148-	A2 4C	LDX #4C	position de la BANQUE sur la disquette MASTER
F14A-	A0 06	LDY #06	octet de poids faible de l'adresse d'exécution
F14C-	D0 10	BNE F15E	-> branchement forcé en #F15E

EXÉCUTION DE LA COMMANDE SEDORIC RENUM

F14E-	A2 42	LDX #42	position de la BANQUE sur la disquette MASTER
F150-	2C A2 47	BIT 47A2	-> continue en #F153

EXÉCUTION DE LA COMMANDE SEDORIC BACKUP

F151-	A2 47	LDX #47	position de la BANQUE sur la disquette MASTER
F153-	2C A2 4C	BIT 4CA2	-> continue en #F15C

EXÉCUTION DE LA COMMANDE SEDORIC SEEK

F154-	A2 4C	LDX #4C	position de la BANQUE sur la disquette MASTER
F156-	2C A2 51	BIT 51A2	-> continue en #F15C

EXÉCUTION DE LA COMMANDE SEDORIC COPY

F157-	A2 51	LDX #51	position de la BANQUE sur la disquette MASTER
F159-	2C A2 56	BIT 56A2	-> continue en #F15C

EXÉCUTION DE LA COMMANDE SEDORIC SYS

F15A-	A2 56	LDX #56	position de la BANQUE sur la disquette MASTER
F15C-	A0 03	LDY #03	octet de poids faible de l'adresse d'exécution

Entrée commune des commandes sur BANQUES externes (sauf INIT)

NB: C'est aussi l'entrée des commandes de la BANQUE n°7, qui sont greffées normalement dans l'ensemble.

F15E-	A9 C4	LDA #C4	empile l'adresse d'entrée de la commande
F160-	48	PHA	dans la BANQUE située de C400 à C7FF
F161-	98	TYA	cette adresse est composée de C4 dans tous les cas
F162-	48	PHA	et de Y (différent selon commande visée)

			X contient toujours le numéro de BANQUE de la commande (#42, #47 etc)
F163-	EC 15 C0	CPX C015	compare BANQUE voulue et n° de BANQUE active
F166-	F0 51	BEQ F1B9	si identiques, continue en F1B9
F168-	2C A2 5B	BIT 5BA2	si différentes, continue en F16B

EXÉCUTION DE LA COMMANDE SEDORIC INIT

F169-	A2 5B	LDX #5B	n° de BANQUE pour INIT

Mise en place de la BANQUE voulue

Le sous-programme F16B/F1B8 demande une disquette master, teste s'il faut abandonner (ESC) ou continuer (RETURN). Dans ce dernier cas, la bitmap est chargée, le sous-programme vérifie qu'il s'agit bien d'une disquette MASTER, lit le nombre de secteurs/piste, calcule à quelle piste et à quel secteur de la disquette se trouve la BANQUE a charger (voir ci-dessus) et charge en C400/C7FF les 4 secteurs correspondant à la BANQUE voulue.

NB: Grâce au PATCH.001 (voir en annexe), SEDORIC est devenu intelligent. Il regarde si une disquette MASTER est présente dans le drive SYSTEM avant d'en réclamer une si ce n'est pas le cas.

F16B-	8A	TXA	empile X le n° de
F16C-	48	PHA	la BANQUE qui sera chargée
F16D-	A2 0C	LDX #0C	indexe "INSERT_MASTER_"
F16F-	20 6C D3	JSR D36C	affiche (X+1) ème message de zone CEE7 terminé par "caractère + 128"
F172-	AD 0A C0	LDA C00A	copie n° du lecteur système dans DRIVE actif
F175-	8D 00 C0	STA C000	affiche "_DISC_IN_DRIVE_ "lettre du lecteur"AND_PRESS_'RETURN'"
F178-	20 48 D6	JSR D648	et attend touche 'ESC' (C = 1) ou 'RETURN' (C = 0)
F17B-	58	CLI	autorise les interruptions
F17C-	08	PHP	sauve les drapeaux, notamment C
F17D-	A9 0B	LDA #0B	XAFCAR affiche le caractère ASCII #0B
F17F-	20 2A D6	JSR D62A	c'est à dire remonte d'une ligne
F182-	28	PLP	restaure les drapeaux, notamment C
F183-	90 0A	BCC F18F	si c'était RETURN continue en F18F

Commande annulée

F185-	68	PLA	sinon, dépile le n° de BANQUE à charger
F186-	C9 5B	CMP #5B	était-ce la BANQUE n°6 (INIT)
F188-	F0 02	BEQ F18C	si oui, saute les deux instructions suivantes
F18A-	68	PLA	sinon, dépile d'abord l'adresse d'entrée
F18B-	68	PLA	de la commande dans la BANQUE
F18C-	4C DC D1	JMP D1DC	JSR CA4E/ROM calcule le déplacement à l'instruction suivante, puis JSR CA3F/ROM met à jour TXTPTR en ajoutant Y et retour à l'interprèteur

La disquette Master est en place

F18F-	20 4C DA	JSR DA4C	XPMAP prend le premier secteur de bitmap dans BUF2, vérifie le format,

			met à zéro le b7 de 2F (flag"première bitmap chargée").
F192-	AD 07 C2	LDA C207	huitième octet bitmap = nombre de secteurs par piste
F195-	8D 4B C0	STA C04B	sauvegarde nombre de secteurs par piste en C04B
F198-	AD 0A C2	LDA C20A	onzième octet bitmap = 00 si disquette MASTER
F19B-	D0 D0	BNE F16D	reboucle à "INSERT_MASTER_" etc... si pas nul
F19D-	A2 FF	LDX #FF	prépare pour X = 0 à l'entrée de la boucle
F19F-	68	PLA	dépile le n° de BANQUE à charger qui représente le n ème secteur de la disquette (début BANQUE)
F1A0-	8D 15 C0	STA C015	et le place dans "n° de BANQUE en place"
F1A3-	38	SEC	prépare calcul coordonnées du début de BANQUE sur disquette
F1A4-	A8	TAY	n° de BANQUE -> Y (sera le n° de secteur)
F1A5-	E8	INX	qui passe à 0 au premier tour (sera le n° de piste)
F1A6-	ED 07 C2	SBC C207	A = n° de BANQUE à charger - nombre secteurs/piste
F1A9-	F0 02	BEQ F1AD	si reste = 0 on a fini, sinon reboucle tant que
F1AB-	B0 F7	BCS F1A4	pas négatif, sort dans les 2 cas avec X = piste Y = secteur
F1AD-	8E 01 C0	STX C001	PISTE (n° de piste à charger) (Y garde n° de secteur)
F1B0-	A2 04	LDX #04	charge 4 secteurs pris à partir du secteur Y de la piste indiquée en
F1B2-	A9 C4	LDA #C4	C001 (PISTE) et les copie de C400 à C7FF (zone des BANQUEs)
F1B4-	20 E5 F1	JSR F1E5	XDLOAD charge X secteurs pris à partir du secteur Y de la piste indiquée en C001 (PISTE) et les copie en RAM à partir de la page A
F1B7-	38	SEC	flag C = 1 (la BANQUE a été mise en place)
F1B8-	24 18	BIT 18	et continue en F1BA

Suite de F166 lorsque la BANQUE était déjà en place

F1B9-	18	CLC	flag C = 0 (la BANQUE était déjà en place)

Lorsque BANQUE voulue est en place

Le sous-programme F1BA/F1E4 met à jour C016, flag "la BANQUE a été chargée", initialise C00D/C00E (EXTER, adresse des messages d'erreur externes) et C00F/C010 (EXTMS, adresse des messages externes) selon les 4 premiers octets du début de la BANQUE active (de C400 à C403), continue en F1D2 s'il s'agit de la commande INIT ou, pour les autres commandes, exécute un RTS qui effectuera un retour fictif à l'adresse précédemment empilée (qui est l'adresse d'entrée de la commande dans la BANQUE).

F1BA-	6E 16 C0	ROR C016	flag "la BANQUE a été changée" si b7 de C016 à 1
F1BD-	A2 03	LDX #03	copie en RAM overlay les 4 premiers octets du début
F1BF-	BD 00 C4	LDA C400,X	de la BANQUE active (de #C400 à #C403)
F1C2-	9D 0D C0	STA C00D,X	en #C00D/0E (EXTER adresse messages d'erreur externes)
F1C5-	CA	DEX	et #C00F/10 (EXTMS adresse messages externes)
F1C6-	10 F7	BPL F1BF	reboucle tant que X n'est pas négatif
F1C8-	AD 15 C0	LDA C015	n° du bloc externe (BANQUE active)
F1CB-	C9 5B	CMP #5B	si c'est la BANQUE #5B (n°6) (pour INIT)
F1CD-	F0 03	BEQ F1D2	branche en F1D2 (saute la ligne suivante)
F1CF-	4C 9E D3	JMP D39E	XCRGOT relit le caractère à TXTPTR (sans incrémenter TXTPTR = CHRGOT), puis le convertit en MAJUSCULE, les espaces sont sautés, Z = 1 si fin d'instruction (0 ou :), C = 0 si caractère chiffre 0 à 9 (soit #30 à #39), sinon C = 1, Y et X inchangés

Cas de INIT

Le sous-programme F1D2/F1E4 charge 99 secteurs pris à partir du secteur 1 de la piste 0 et les copie en RAM de 3000 à 92FF puis effectue le fameux JMP C404 dont j'ai déjà parlé (entrée pour exécution de INIT dans la BANQUE n°6).

C'est bien beau, mais voilà qui ne tient pas compte du fait que l'on ne désire peut-être pas formater en MASTER. Résultat: si finalement on formate en SLAVE, le chargement de 95 secteurs en RAM au lieu de 8 est une perte de temps et de place bien inutile. En fait, je n'ai pas corrigé cette bogue, car avec la version 3.0 de SEDORIC, on ne manque plus de place sur les disquettes et il devient inutile (ou rare en tout cas) de formater en SLAVE.

F1D2-	A2 **63**	LDX #**63**	chargera 99 secteurs à partir de secteur 1 de piste 0

Ici, une petite modification (l'octet ci-dessus en gras) a été apportée pour tenir compte de l'existence de la nouvelle BANQUE n°7. Les versions antérieures de SEDORIC chargeaient 95 secteurs (ce qui était d'ailleurs une bogue mineure, car il n'y en avait que 94 de nécessaire). La version 3.0 en charge 94 + 5 = 99.

F1D4-	A9 30	LDA #30	et les copiera en RAM de l'adresse 3000 à l'adresse 92FF
F1D6-	A0 00	LDY #00	la prochaine piste à copier sera
F1D8-	8C 01 C0	STY C001	la première (piste n°0) et le
F1DB-	C8	INY	prochain secteur à copier sera le secteur n°1
F1DC-	20 E5 F1	JSR F1E5	XDLOAD charge X secteurs pris à partir du secteur Y de la piste indiquée en C001 (PISTE) et les copie en RAM à partir de la page A
F1DF-	20 9E D3	JSR D39E	XCRGOT relit le caractère à TXTPTR (sans incrémenter TXTPTR = CHRGOT), puis le convertit en MAJUSCULE, les espaces sont sautés, Z = 1 si fin d'instruction (0 ou :), C = 0 si caractère chiffre 0 à 9 (soit #30 à #39), sinon C = 1, Y et X inchangés
F1E2-	4C 04 C4	JMP C404	entrée pour exécution de INIT dans la BANQUE n°6

XDLOAD charge X secteurs pris à partir du secteur Y de la piste indiquée en C001 (PISTE) et les copie en RAM à partir de la page A

F1E5-	86 F5	STX F5	nombre de secteurs à charger
F1E7-	8D 04 C0	STA C004	adresse de chargement du prochain secteur:
F1EA-	A9 00	LDA #00	(adresse RWBUF est formée de HH=A et LL=00)
F1EC-	8D 03 C0	STA C003	soit #C400 pour une BANQUE par exemple
F1EF-	78	SEI	interdit les interruptions
F1F0-	8C 02 C0	STY C002	SECTEUR (n° du prochain secteur à charger)
F1F3-	20 73 DA	JSR DA73	XPRSEC lit un secteur selon DRIVE, PISTE, SECTEUR et RWBUF
F1F6-	EE 04 C0	INC C004	page suivante (adresse prochain chargement)
F1F9-	AC 02 C0	LDY C002	compare n° du secteur chargé avec
F1FC-	CC 4B C0	CPY C04B	nombre de secteurs par piste
F1FF-	90 05	BCC F206	si pas encore égal augmente seulement n° de secteur
F201-	EE 01 C0	INC C001	si dernier lu augmente PISTE (n° de piste)
F204-	A0 00	LDY #00	et remet le n° de secteur à zéro
F206-	C8	INY	secteur suivant (n°1 si début de piste)

F207-	C6 F5	DEC F5	nombre de secteurs restant à charger
F209-	D0 E5	BNE F1F0	reboucle s'il en reste à charger
F20B-	58	CLI	restaure les interruptions
F20C-	60	RTS	et retourne

F20D- 4C E0 E0 JMP E0E0 "FILE_TYPE_MISMATCH_ERROR" (appelé seulement par WINDOW)

EXÉCUTION DE LA COMMANDE SEDORIC WINDOW

Rappel de la syntaxe

WINDOW (nom_de_fichier_non_ambigu)

La commande WINDOW affiche à l'écran le masque de saisie indiqué (nom_de_fichier_non_ambigu) (crée par CREATEW) et remplit les champs de données avec les valeurs trouvées dans le tableau WI$ (complétées avec des espaces si nécessaire). Le tableau WI$ doit avoir été correctement dimensionné. Lorsqu' aucun nom de masque n'est pas indiqué, le masque courant (dans le tampon C400/C7E7) est utilisé.

Rappel: un masque d'écran TEXT comporte 25 lignes de 40 caractères et commence à la ligne 2 de l'écran (une ligne reste libre entre la "ligne service" et la première ligne du masque). Les deux lignes situées au-dessus du masque ainsi que la dernière ligne en bas de l'écran sont utilisables pour afficher un en-tête ou un menu.

WINDOW permet la saisie des données dans les champs du masque d'écran et de se déplacer d'un champ à l'autre à l'aide des flèches ou de RETURN (la plupart des caractères de contrôle sont filtrés). Le curseur n'est visible que dans les champs. CTRL/C provoque la sortie: tous les champs sont alors relus et copiés dans WI$.

Rappel sur l'écran TEXT: il faut distinguer les coordonnées colonne/ligne BASIC (qui diffèrent entre l'ORIC-1 et l'ATMOS, voir plus loin) et les coordonnées colonne/ligne utilisées par SEDORIC, qui sont celles mises à jour en 0269 et 0268. Dans tous les cas la "ligne service" est hors-jeu et seule la partie accessible est prise en considération.

Pour SEDORIC, c'est très simple: la première colonne porte le n°0 (#00) et la dernière le n°39 (#27); la première ligne porte le n°1 (#01) et la dernière le n°27 (#1B). Ces valeurs sont mises à jour par la ROM en 0269 (n° de colonne = abscisse) et 0268 (n° de ligne = ordonnée), qui indiquent les **coordonnées de la case où se fera le prochain affichage**.

Qu'il s'agisse d'un ORIC-1 ou d'un ATMOS, les coordonnées de la première case accessible avec le curseur en mode 38 colonnes (mode normal) sont (2,1).

Cependant, petit détail, sur ORIC-1, après un CTRL/L ou un retour à la ligne, le curseur se place contre le bord gauche de l'écran (et 0269 contient la valeur #00), mais la case où se fera le prochain affichage est la troisième. A ce moment là, le curseur bondira directement à la troisième case.

Au contraire, avec l'ATMOS, après un CTRL/L ou un retour à la ligne, le curseur se place en attente à la troisième colonne (et 0269 contiendra la valeur #02), le caractère y sera affiché et le curseur passera à la

quatrième case (et 0269 contiendra la valeur #03).

Cette notation (X,Y), avec X de 0 à 39 et Y de 1 à 27, basée sur le contenu de 0269/0268 et qui est la plus homogène, sera utilisée par la suite dans cet exposé.

Les coordonnées colonne/ligne BASIC de l'ORIC-1 sont un peu farfelues en ce sens que la première colonne n'a pas de n°, la deuxième colonne a le n°0 (#00) et la dernière porte le n° 38 (#26); la première ligne s'est vu attribuer le n°0 (#00) et la dernière le n°26 (#1A).

Ces valeurs sont utilisées par exemple par PLOT. En mode 38 colonnes (mode normal), les coordonnées de la première case accessible avec le curseur sont donc (1,0). Curiosité: PLOT permet d'accéder à la deuxième case de la marge (PLOT 0,0,...), mais pas à la première!

Les coordonnées colonne/ligne BASIC de l'ATMOS sont plus raisonnables: la première colonne a le n°0 (#00) et la dernière le n° 39 (#27); la première ligne porte le n°0 (#00) et la dernière le n°26 (#1A). Ces valeurs sont utilisées par exemple par PLOT ou par PRINT@. En mode 38 colonnes (mode normal), les coordonnées de la première case accessible avec le curseur sont donc (2,0). PLOT permet enfin d'accéder à la première case de la marge (PLOT 0,0,...)!

Ainsi WINDOW fonctionne en mode TEXT et est interdit en HIRES, il faut faire attention au mode "40 colonnes" où des problèmes peuvent apparaître et l'ORIC-1 lui-même peut donner des résultats bizarres. Il faut donc prendre garde à gérer correctement les champs de données qui seront à cheval sur plusieurs lignes.

WINDOW utilise un "truc" assez joli: l'indicateur de champ (posé par CTRL/W de CREATEW est aussi le carré plein de couleur INK, utilisé par la touche DEL et est remplacé par un espace (carré plein de couleur PAPER) par la suite.

Analyse de la commande WINDOW

F210-	F0 27	BEQ F239	branche si fin d'instruction (pas de nom de masque)
F212-	20 4F D4	JSR D44F	XNF lit nom de fichier non-ambigu à TXTPTR et l'écrit dans BUFNOM
F215-	20 9E D7	JSR D79E	XNOWILD recherche "?" dans BUFNOM revient avec C = 1 si pas trouvé ou génère une "WILDCARD(S)_NOT_ALLOWED_ERROR" si trouvé
F218-	20 E6 DF	JSR DFE6	XDEFLO met des valeurs par défaut pour XLOADA, notamment met #FF dans C072 et remet à zéro VSALO0, VSALO1, C04F et C050
F21B-	A9 00	LDA #00	AY = DESALO = C400 (adresse de chargement)
F21D-	A0 C4	LDY #C4	un masque WINDOW occupe #03E8 octets de BBD0 (0,2) à BFB7 (39,26) dans l'écran TEXT

Ni la ligne service, ni les première (n°1) et dernière (n°27) lignes accessibles de l'écran ne sont utilisées par le masque. La taille maximale du masque a été dictée par la place disponible dans les 4 pages de C400 à C7FF soit #4000 octets. Un bel exemple d'optimisation!

F21F-	8D 52 C0	STA C052	la zone de BANQUE interchangeable située de
F222-	8C 53 C0	STY C053	C400 à C7FF sera donc écrasée de C400 à C7E7
F225-	A9 40	LDA #40	VSALO1 = #40 (code ",A" pour indiquer une
F227-	8D 4E C0	STA C04E	adresse de chargement spéciale)

F22A-	20 E5 E0	JSR E0E5	XLOADA charge le masque selon BUFNOM, VSALO0, VSALO1 et DESALO qui ont tous été remis à jour ci-dessus
F22D-	AD 51 C0	LDA C051	FTYPE: type du fichier chargé (manuel p 100)
F230-	29 20	AND #20	0010 0000 remet à zéro tous les bits sauf b5
F232-	F0 D9	BEQ F20D	si b5 est à 0, erreur: ce n'est pas un masque WINDOW, continue en F20D ("FILE_TYPE_MISMATCH_ERROR").

Bogue: il est un peu tard pour s'apercevoir du problème, la RAM overlay est probablement déjà écrasée!

F234-	A9 01	LDA #01	mise à jour de EXTNB, numéro du bloc externe
F236-	8D 15 C0	STA C015	(la valeur 1 indique "masque WINDOW en place")

Entrée secondaire si masque déjà en place (ou supposé en place)

F239-	AC 15 C0	LDY C015	Y = EXTNB (numéro du bloc externe)
F23C-	88	DEY	teste si un masque WINDOW est en place
F23D-	D0 CE	BNE F20D	sinon, vers "FILE_TYPE_MISMATCH_ERROR"
F23F-	AD 6A 02	LDA 026A	si oui, sauve sur la pile le registre d'état
F242-	48	PHA	de la console et les indicateurs du 6502
F243-	08	PHP	

Copie du masque dans l'écran TEXT

F244-	20 DE DF	JSR DFDE	vérifie que l'on est bien en mode TEXT, sinon génère une "DISP_TYPE_MISMATCH_ERROR", réinitialise la pile et retourne au "Ready"
F247-	A9 B8	LDA #B8	
F249-	A0 BB	LDY #BB	(#BBB8 = #BBD0 - #18)
F24B-	85 F2	STA F2	prépare un move de #03E8 octets
F24D-	84 F3	STY F3	(4 fois #100 moins #18)
F24F-	A9 E8	LDA #E8	de C400 à C7E7 (masque en RAM overlay)
F251-	A0 C3	LDY #C3	vers BBD0 à BFB7 (dans l'écran TEXT)
F253-	85 F4	STA F4	
F255-	84 F5	STY F5	
F257-	A2 04	LDX #04	pour 4 pages (une page = une zone de 256 octets)
F259-	A0 18	LDY #18	en commençant au dix huitième octet de la première page
F25B-	B1 F4	LDA (F4),Y	lit octet dans le masque en RAM overlay
F25D-	91 F2	STA (F2),Y	écrit cet octet dans l'écran texte
F25F-	C8	INY	octet suivant
F260-	D0 F9	BNE F25B	reboucle tant que la page n'est pas finie (jusqu'à ce que Y = 0)
F262-	E6 F3	INC F3	indexe page suivante (incrémente HH adresse écriture)
F264-	E6 F5	INC F5	indexe page suivante (incrémente HH adresse lecture)
F266-	CA	DEX	décrémente le nombre de page restant à copier
F267-	D0 F2	BNE F25B	reboucle tant qu'il reste des pages à copier

Copie WI$ dans les champs à l'écran et se place sur le premier champ

WINDOW ne vérifie pas si WI$ existe (re-bogue). De plus si WI$ est vide, sa recopie à l'écran est une perte

de temps inutile.

F269-	20 27 F3	JSR F327	remplit les champs avec les chaînes de WI$
F26C-	20 09 F3	JSR F309	cherche la première case du premier champ présent dans le masque de C400 à C7E7 et sort de WINDOW si ne trouve pas de champ

Saisie au clavier les données d'un champ (curseur visible)

F26F-	20 3E D7	JSR D73E	champ trouvé, force l'affichage du curseur avec routine XCURON
F272-	58	CLI	autorise les interruptions (pour saisir touche)
F273-	20 45 D8	JSR D845	XKEY prend un caractère au clavier (entrée générale)
F276-	10 FB	BPL F273	reboucle tant que b7 = 0 (attente touche)
F278-	78	SEI	interdit les interruptions (touche saisie)
F279-	C9 03	CMP #03	est-ce un CTRL/C?
F27B-	F0 68	BEQ F2E5	si oui, suite en F2E5 (sauvegarde des données)
F27D-	C9 7F	CMP #7F	sinon, est-ce la touche DEL?
F27F-	D0 15	BNE F296	si ce n'est pas le cas, continue en F296
F281-	A9 08	LDA #08	si c'est DEL, A = CTRL/H (une flèche gauche, sera "affichée" par la commande suivante pour tester si ce DEL est possible)
F283-	20 EC F2	JSR F2EC	Gestion déplacements du curseur dans le masque
F286-	30 E7	BMI F26F	reprend saisie si "bute" contre début de masque
F288-	20 CA F2	JSR F2CA	si ce n'est pas le cas, lit dans le masque l'octet correspondant au curseur (donc à gauche de la position initiale)
F28B-	D0 15	BNE F2A2	si hors champ, ce DEL est illégal, continue en F2A2 avec une position de curseur faussée par le test de validité du DEL
F28D-	A9 09	LDA #09	si c'est une case de champ, A = CTRL/I (flèche droite, sera "affichée" par la commande suivante pour revenir à la case initiale)
F28F-	20 2A D6	JSR D62A	XAFCAR affiche ce caractère (retour case initiale)
F292-	A9 7F	LDA #7F	A = DEL (reprend la valeur initiale de DEL)
F294-	D0 04	BNE F29A	suite forcée (où DEL sera traité car valide)

Suite de l'analyse de touche si ni CTRL/C, ni DEL illégal

F296-	C9 20	CMP #20	est-ce un code de CTRL? (A = code ASCII < #20)
F298-	90 0A	BCC F2A4	si oui, continue en F2A4...

NB: Tous les CTRL ont été éliminés, il ne reste que DEL (si valide) et les caractères affichables, compris entre #20 (espace) et #7E (damier ou ê)

F29A-	20 2A D6	JSR D62A	XAFCAR affiche ce caractère ASCII
F29D-	A9 08	LDA #08	affiche A = CTRL/H (flèche gauche, pour
F29F-	20 2A D6	JSR D62A	"neutraliser la neutralisation suivante")

Ici, on ne sait pas si c'est génial ou s'il s'agit d'un bricolage de débogage. La détection d'un DEL illégal a entraîné un mouvement inopiné à gauche, qui sera "réparé" par l'instruction en F2A2 (le prochain caractère affiché devient une flèche droite). Manque de chance, l'affichage des caractères valides, qui suit son cours normal en F29A, arrive en F2A2 sur cette "réparation", qui du coup devient gênante. Pour contrecarrer cette anomalie, une flèche gauche a été intercalée en F29D!

F2A2-	A9 09	LDA #09	A = flèche droite (pour neutraliser le DEL illégal)

Traitement des codes de CTRL

Tous les codes de CTRL sont acceptés sauf CTRL/L, CTRL/N et les DEL inappropriés. De plus CTRL/M est modifié pour passer au champ suivant.

F2A4-	C9 08	CMP #08	est-ce un code < #08? (de CTRL/@ à CTRL/G)
F2A6-	90 F2	BCC F29A	si oui, accepté (concerne notamment CTRL/A)
F2A8-	C9 0C	CMP #0C	est-ce un CTRL/L? (effacement de l'écran)
F2AA-	F0 C3	BEQ F26F	si oui, refusé (reboucle saisie d'autre chose)
F2AC-	90 12	BCC F2C0	si c'est une flèche, continue en F2C0
F2AE-	C9 0E	CMP #0E	est-ce CTRL/N? (effacement de la ligne)
F2B0-	F0 BD	BEQ F26F	si oui, refusé (reboucle saisie d'autre chose)
F2B2-	C9 0D	CMP #0D	est-ce CTRL/M? (RETURN)
F2B4-	D0 E4	BNE F29A	sinon, tous les autres codes de CTRL acceptés (de CTRL/O à CTRL/£ et notamment CTRL/Q, CTRL/T, CTRL/Z et CTRL/[)
F2B6-	A9 09	LDA #09	si RETURN, A = flèche droite (pour sauter d'un champ à l'autre) Au cours de la boucle suivante la valeur de A est conservée et permet d'explorer le masque pour trouver la fin du champ.
F2B8-	20 EC F2	JSR F2EC	Gestion déplacements du curseur dans le masque
F2BB-	20 CA F2	JSR F2CA	lit dans le masque l'octet correspondant au curseur
F2BE-	F0 F8	BEQ F2B8	si c'est une case de champ, reboucle en F2B8

Cherche le champ suivant et procède à une saisie de données

F2C0-	20 EC F2	JSR F2EC	Gestion déplacements du curseur dans le masque
F2C3-	20 CA F2	JSR F2CA	lit dans le masque l'octet correspondant au curseur
F2C6-	D0 F8	BNE F2C0	si pas champ, reboucle en F2C0 jusqu'à champ
F2C8-	F0 A5	BEQ F26F	si c'est une case de champ, reboucle en F26F

Lecture dans le masque de C400 à C7E7 de l'octet correspondant au curseur et teste si #7F (présence d'une case appartenant à un champ)

F2CA-	48	PHA	sauvegarde A sur la pile
F2CB-	20 40 D7	JSR D740	XCUROFF cache le curseur (= vidéo normale)
F2CE-	18	CLC	prépare une addition
F2CF-	A5 12	LDA 12	ajoute #0830 (#0830 = C400 - BBD0)
F2D1-	69 30	ADC #30	à l'adresse de la ligne du curseur TEXT
F2D3-	85 F8	STA F8	et place le résultat dans F8/F9
F2D5-	A5 13	LDA 13	qui contient alors l'adresse de la "ligne" du
F2D7-	69 08	ADC #08	masque (situé de C400 à C7E7) correspondant à
F2D9-	85 F9	STA F9	la ligne du curseur dans l'écran
F2DB-	AC 69 02	LDY 0269	Y = n° de colonne du curseur (de 0 à 39, indexe dans la ligne l'octet du masque correspondant au curseur dans l'écran)
F2DE-	B1 F8	LDA (F8),Y	lit octet dans le masque

F2E0-	A8	TAY	et le sauve dans Y
F2E1-	68	PLA	récupère la valeur de A d'origine
F2E2-	C0 7F	CPY #7F	et retourne avec Z = 1 si case de champ
F2E4-	60	RTS	

CTRL/C: sauvegarde les données avant de sortir de WINDOW

F2E5-	28	PLP	récupère les indicateurs 6502
F2E6-	20 25 F3	JSR F325	copie les champs dans le tableau WI$
F2E9-	4C 20 F3	JMP F320	et termine

Gestion des déplacements du curseur dans le masque avec les flèches:
(Pas simple, mais efficace!)

Entrée lorsque le curseur est dans le masque

Force b7 de F2 à 0 (flag "dans le masque"), affiche A (flèche) et teste si la case suivante est dans le masque. Si oui (C = 0), retourne avec flag à 0 et N = 0. Sinon (C = 1), effectue le déplacement inverse et retourne avec flag à 1 et N = 1. Après ce déplacement inverse, C = 0 si retour dans le masque.

F2EC-	46 F2	LSR F2	0 -> b7 de F2 (flag "curseur dans le masque")

Entrée secondaire: rebouclage pour mouvement inverse

En cas de rebouclage: le b7 de F2 est à 1 (flag "hors du masque"). Teste si la case suivante (après déplacement inverse) est dans le masque. Si oui C = 0, sinon C = 1, dans les deux cas le flag F2 et N restent inchangés à 1.

F2EE-	20 2A D6	JSR D62A	XAFCAR affiche le caractère ASCII contenu dans A
F2F1-	AC 68 02	LDY 0268	Y = n° de la ligne du curseur TEXT
F2F4-	C0 01	CPY #01	est-ce la ligne n°1? (donc hors masque)
F2F6-	F0 04	BEQ F2FC	si oui, continue en F2FC (avec C = 1)
F2F8-	C0 1B	CPY #1B	est-ce la ligne n°27? (donc hors masque)
F2FA-	D0 0A	BNE F306	sinon, continue en F306 (avec C = 0)
F2FC-	24 F2	BIT F2	si hors masque (C = 1), teste flag "masque"
F2FE-	30 08	BMI F308	si b7 à 1, simple RTS (rebouclage déjà fait)
F300-	66 F2	ROR F2	sinon, C -> b7 donc flag F2 et N passent à 1
F302-	49 01	EOR #01	0000 0001 inverse le b0 de A, c'est à dire inverse le sens de la flèche. Flèche gauche (#08) devient flèche droite (#09) et réciproquement. Idem pour flèche bas (#0A) et flèche haut (#0B).
F304-	D0 E8	BNE F2EE	et rebouclage forcé pour mouvement inverse
F306-	24 F2	BIT F2	positionne N selon b7 de F2 et retourne
F308-	60	RTS	

Cherche première case de champ présente dans le masque en C400/C7E7

F309-	A9 1E	LDA #1E	caractère de contrôle pour placer curseur en (0,1) c'est à dire au début de l'écran (première case de première ligne) (fonction HOME)

F30B-	20 2A D6	JSR D62A	XAFCAR "affiche" ce caractère ASCII (CTRL/^)
F30E-	20 06 D2	JSR D206	JSR CBF0/ROM (CRLF) qui place donc le curseur au début du masque
F311-	20 CA F2	JSR F2CA	lit dans le masque l'octet correspondant au curseur
F314-	F0 0E	BEQ F324	simple RTS si c'est #7F (indicateur de champ)
F316-	A9 09	LDA #09	sinon, A = curseur vers la droite (CTRL/I)
F318-	20 EC F2	JSR F2EC	gestion des déplacements du curseur dans le masque
F31B-	10 F4	BPL F311	reboucle si curseur est toujours dans le masque
F31D-	68	PLA	si fin de masque atteinte sans trouver de #7F,
F31E-	68	PLA	(curseur ligne 27) supprime deux octets de la pile, c'est à dire, ôte la première adresse de retour et **DONC** sort de WINDOW
F31F-	28	PLP	récupère les indicateurs 6502
F320-	68	PLA	récupère A
F321-	8D 6A 02	STA 026A	et le remet en 026A (flags d'état console)
F324-	60	RTS	

Copie de WI$ dans SCRN et de SCRN dans WI$

Il s'agit de 2 routines complexes qui s'entremêlent constamment, ce qui gagne quelques octets, mais perd beaucoup en clarté! En résumé, on a:

(1) l'écran avec le curseur et les pointeurs 0269, 0268 et 12/13
(2) la zone intermédiaire BUF1 (C100 à C1FF) pour stocker les chaînes
(3) la zone de stockage finale sous HIMEM (selon descripteurs) et
(4) le tableau WI$ contenant la liste de descripteurs de chaîne.

A chaque champ de données (dans le masque) correspond une chaîne (stockée sous HIMEM et repérée par un descripteur dans WI$) qui sera copiée de ou à l'écran (à la place correspondante). L'exploration du masque par un "curseur" fictif (qui suit les déplacements du vrai curseur dans l'écran) permet de savoir où sont les cases de champs (#7F).

Afin de comprendre ce qui suit, je vous conseille de suivre d'abord le fil du sous-programme WI$->SCRN qui est plus facile.

Organigramme de la routine WI$ -> SCRN

a) Cherche le premier champ dans le masque et positionne le curseur
b) Initialise la recherche de la première chaîne dans WI$
c) Cherche un descripteur de chaîne dans WI$, met son adresse dans B6/B7
d) Ecrit la longueur de la chaîne dans F2 et son adresse dans 91/92
e) Copie dans l'écran la chaîne lue dans la zone sous HIMEM (cette opération est pilotée par le déplacement simultané du "curseur" dans le masque, afin de détecter la présence de "#7F" matérialisant le champ)
f) Tant que la fin du masque n'est pas atteinte, reboucle en (c)

Organigramme de la routine SCRN -> WI$

a) Cherche le premier champ dans le masque et positionne le curseur
b) Initialise la recherche de la première chaîne dans WI$
c) Cherche un descripteur de chaîne dans WI$, met son adresse dans B6/B7

d) Ecrit la longueur dans F2 et l'adresse dans 91/92 (bogue, c'est inutile)
e) Copie dans BUF1 la chaîne présente à l'écran (opération pilotée par le déplacement du "curseur" dans le masque pour détecter les "#7F" de champ)
f) Réserve sous HIMEM une chaîne de longueur D0 et d'adresse D1/D2
g) Copie sous HIMEM (selon D0/D1/D2) la chaîne en attente dans BUF1
h) Met à jour le descripteur (pointé par B6/B7) dans WI$ selon D0/D1/D2
i) Tant que la fin du masque n'est pas atteinte, reboucle en (c)

Rappel: le tableau WI$ ne contient pas les chaînes, mais leurs descripteurs. Les chaînes proprement dites sont stockées dans la zone sous HIMEM. Par simplification de langage, on "lit" ou on "écrit" une chaîne dans WI$.

Recopie les champs de l'écran dans le tableau WI$ (SCRN->WI$)

F325-	18	CLC	C = 0, entrée appelée avant de sortir de WINDOW
F326-	24 38	BIT 38	et continue en F328

Recopie les chaînes de WI$ dans les champs de l'écran (WI$->SCRN)

F327-	38	SEC	C = 1, entrée appelée au début de WINDOW
F328-	6E 72 C0	ROR C072	C -> b7 de C072 qui sert désormais à savoir si on est entré en F325 ou en F327, c'est à dire, s'il faut copier de **champ à l'écran -> tableau (SCRN->WI$)** ou de **tableau -> champ (WI$->SCRN)**
F32B-	20 09 F3	JSR F309	cherche le premier champ dans le masque en C400/C7E7
F32E-	A9 57	LDA #57	place "WI$" en B4/B5 (caractères significatifs d'une
F330-	A0 C9	LDY #C9	variable) (#57="W" et #C9="I"+128 pour $)
F332-	85 B4	STA B4	bogue: pas de vérification concernant
F334-	84 B5	STY B5	l'existence de WI$, ni sa validité,
F336-	A9 00	LDA #00	ni s'il contient quelque chose à copier
F338-	85 F6	STA F6	#00 -> F6/F7 (n° de la première chaîne à chercher)
F33A-	85 F7	STA F7	

Cherche une chaîne dans le tableau WI$

F33C-	A0 01	LDY #01	
F33E-	84 26	STY 26	#01 -> 26 (nombre de dimensions du tableau)
F340-	88	DEY	#00 -> 29 (b7 = 0 flag "non entier")
F341-	84 29	STY 29	#00 -> 27 (flag consultation tableau, inhibe
F343-	84 27	STY 27	"REDIM'D_ARRAY_ERROR", si pas nul déclenche
F345-	88	DEY	un nouveau DIM cf "ORIC À NU" pages 154/155)
F346-	84 28	STY 28	#FF -> 28 (b7 = 1 flag "chaîne")
F348-	A4 F6	LDY F6	
F34A-	A6 F7	LDX F7	YX reçoit F6/F7 (n°de la chaîne à chercher)
F34C-	E6 F6	INC F6	puis F6/F7 est incrémenté (chaîne suivante)
F34E-	D0 02	BNE F352	
F350-	E6 F7	INC F7	
F352-	20 D1 04	JSR 04D1	D306/ROM cherche la chaîne dans le tableau WI$ retourne avec l'adresse du

			descripteur de chaîne dans B6/B7
F355-	A0 00	LDY #00	prépare index Y pour lire ce descripteur
F357-	B1 B6	LDA (B6),Y	lit le premier octet et le place en F2
F359-	85 F2	STA F2	(longueur de la chaîne)
F35B-	C8	INY	
F35C-	B1 B6	LDA (B6),Y	lit le deuxième octet et le place en 91
F35E-	85 91	STA 91	(LL de l'adresse de la chaîne)
F360-	C8	INY	
F361-	B1 B6	LDA (B6),Y	lit le troisième octet et le place en 92
F363-	85 92	STA 92	(HH de l'adresse de la chaîne)
F365-	A2 00	LDX #00	X pointe dans la chaîne (X = 0 pour premier caractère)
F367-	2C 72 C0	BIT C072	teste le flag b7 de C072 (0 si juste avant la sortie de WINDOW, cas où il faut copier de l'écran vers le tableau WI$)
F36A-	10 14	BPL F380	si c'est le cas, continue en F380 (SCRN->WI$)

Recopie la chaîne de WI$ dans le champ à l'écran (WI$->SCRN)

F36C-	E4 F2	CPX F2	pointeur X comparé à la longueur de chaîne F2, ce qui positionne C à 0 si X < F2 ou C à 1 si X >= F2, c'est à dire si le pointeur a atteint la fin de la chaîne (sera testé plus loin en F375)
F36E-	8A	TXA	sauve le pointeur X dans A (= pointeur actuel)
F36F-	E8	INX	X vise le prochain caractère (Z = 1 si X = #00)
F370-	F0 59	BEQ F3CB	si X = #00, "STRING_TOO_LONG_ERROR" Lorsque X = #100, le pointeur actuel A = #FF ce qui indique que la chaîne est trop longue. En effet sous SEDORIC les chaînes ne peuvent avoir plus de 255 caractères or le premier caractère est visé par X = #00 et le dernier par X = #FE (254)
F372-	A8	TAY	récupère le pointeur actuel dans Y pour index
F373-	B1 91	LDA (91),Y	lit octet selon adresse de chaîne en 91/92 + index
F375-	90 1C	BCC F393	si la fin de la chaîne n'était pas atteinte, continue en F393 pour affichage de cet octet à l'écran

Suite WI$->SCRN: cas où la chaîne est plus courte que le champ (C = 1)

F377-	A9 7F	LDA #7F	si la fin de la chaîne dans WI$ était atteinte, remplace cet octet par #7F (carré de couleur INK), pour compléter le champ
F379-	AC 69 02	LDY 0269	Y = n° de colonne où est le curseur TEXT
F37C-	91 12	STA (12),Y	copie #7F dans l'écran sous le curseur TEXT. NB: 12/13 contient l'adresse du début de la ligne dans l'écran et Y la position du curseur TEXT sur cette ligne.
F37E-	B0 11	BCS F391	suite forcée en F391 car C = 1 (chaîne < champ)

SCRN->WI$: copie chaîne de l'écran dans BUF1 (avant d'aller dans WI$)

F380-	AC 69 02	LDY 0269	Y = n° de colonne où est le curseur TEXT
F383-	B1 12	LDA (12),Y	lit le caractère sous le curseur TEXT
F385-	C9 7F	CMP #7F	est-ce #7F? (carré couleur INK utilisé par DEL)
F387-	D0 02	BNE F38B	sinon, saute l'instruction suivante
F389-	A9 20	LDA #20	remplace #7F par un espace

F38B-	9D 00 C1	STA C100,X	écrit le contenu de A dans l'octet n° X de BUF1 où la chaîne est assemblée avant "copie" dans WI$
F38E-	E8	INX	vise le prochain caractère à copier
F38F-	F0 3A	BEQ F3CB	"STRING_TOO_LONG_ERROR" si X = #00, on a écrit un caractère de trop, celui visé par X = #FF

Suite commune pour SCRN->WI$ ou champ->chaîne

Le STA (12),Y en F37C et le STA C100,X en F38B n'ont ni fait avancer le curseur, ni mis à jour les pointeurs 0269, 0268 et 12/13. Ceci sera fait ci-après, en "affichant" une flèche droite.

F391-	A9 09	LDA #09	A = curseur à droite (CTRL/I) (case suivante)

Suite commune pour tous (SCRN->WI$ et WI$->SCRN)

F393-	20 2A D6	JSR D62A	XAFCAR affiche le caractère ASCII contenu dans A
F396-	20 CA F2	JSR F2CA	lit dans le masque l'octet correspondant au curseur, au retour Z = 1 si une case de champ a été trouvée
F399-	F0 CC	BEQ F367	reboucle si #7F (lit caractère suivant de la chaîne)
F39B-	2C 72 C0	BIT C072	teste le flag b7 de C072 (0 si juste avant la sortie de WINDOW, cas où il faut copier de l'écran vers le tableau WI$)
F39E-	30 1C	BMI F3BC	si ce n'est pas le cas, continue en F3BC

Suite pour SCRN->WI$: recopie de BUF1 dans WI$

F3A0-	86 F2	STX F2	sauve X dans F2
F3A2-	8A	TXA	et dans A (rappel: X est le pointeur de chaîne)
F3A3-	20 64 D2	JSR D264	JSR D5AB/ROM réserve une place en mémoire pour la chaîne de longueur A, retourne avec cette longueur en D0 et l'adresse de la chaîne en D1/D2
F3A6-	A0 00	LDY #00	prépare index pour copie de BUF1 -> chaîne
F3A8-	B9 00 C1	LDA C100,Y	lit octet n°Y dans BUF1
F3AB-	91 D1	STA (D1),Y	écrit cet octet dans la chaîne réservée
F3AD-	C8	INY	octet suivant
F3AE-	C4 F2	CPY F2	compare Y et valeur dans F2 (longueur chaîne)
F3B0-	D0 F6	BNE F3A8	reboucle tant que pas fini
F3B2-	A0 02	LDY #02	prépare Y pour copier 3 octets (longueur et adresse de la chaîne)
F3B4-	B9 D0 00	LDA 00D0,Y	lit octet du descripteur de chaîne (longueur et adresse)
F3B7-	91 B6	STA (B6),Y	et le copie en B6/B7/B8
F3B9-	88	DEY	pour l'octet précédent
F3BA-	10 F8	BPL F3B4	reboucle tant que Y n'est pas négatif

Suite pour tous (SCRN->WI$ et WI$->SCRN): recherche le champ suivant

F3BC-	A9 09	LDA #09	A = curseur vers la droite (CTRL/I)
F3BE-	20 EC F2	JSR F2EC	Gestion déplacements du curseur dans le masque
F3C1-	30 0B	BMI F3CE	simple RTS si N = 1 (fin de masque atteinte)
F3C3-	20 CA F2	JSR F2CA	lit dans le masque l'octet correspondant au curseur
F3C6-	D0 F4	BNE F3BC	reboucle en F3BC tant que Z = 0 (cherche champ)

F3C8- 4C 3C F3 JMP F33C champ trouvé, reboucle en F33C pour chercher l'adresse de la chaîne suivante dans le tableau WI$

F3CB- 4C 77 E9 JMP E977 "STRING_TOO_LONG_ERROR"

F3CE- 60 RTS

GESTION DES FICHIERS

Remarques préliminaires

Trois types de fichiers de data peuvent être utilisés sous SEDORIC: les fichiers "**S**équentiels", "Di**R**ects" (en fait pour "**R**andom") et "Accès **D**isque". Lors de l'ouverture d'un fichier, à l'aide de OPEN S, R ou D, à chaque nom de fichier est associé un n° logique NL qui, pour plus de simplicité, est ensuite utilisé à la place du nom de fichier. Il est théoriquement possible d'en ouvrir 64 (NL de 0 à 63). Tous les fichiers ouverts, quel que soit leur type, utilisent (de manière transparente pour l'utilisateur) un "Pseudo-Tableau" de nom **FI**. Ce "Pseudo-tableau" **FI** est créé lors du premier OPEN. Sa structure, très complexe, ainsi que les variables utilisées par les commandes de gestion de fichiers sont expliqués plus loin..

Afin de comprendre ce qui suit, un peu de terminologie est nécessaire (dans ce qui suit, le terme "variables BASIC" implique en fait toute expression BASIC valide):

Et tout d'abord, une petite précision concernant les flags indicateurs de types de fichiers. Le flag FTYPE **seul connu de l'utilisateur** est décrit dans le manuel page 100. C'est lui qui est affiché par l'option ",V" de la commande CLOAD. Il vaut #08 pour les fichiers **S**équentiels, #10 pour les fichiers di**R**ects et n'est pas défini pour les fichiers **D**isques. SEDORIC utilise la case mémoire F9 en page zéro pour gérer ce flag lors des opérations de lecture / écriture des fichiers sur disquette. A coté de ces FTYPEs, il existe un flag, situé en 0B en page zéro, qui est utilisé en interne par les routines de gestion de fichiers et qui vaut respectivement #80, #00 et #01 pour les fichiers de type **S**équentiels, di**R**ects et **D**isques. Sauf précision, ce sont ces valeurs qui sont utilisées dans ce qui suit pour désigner les différents types de fichiers.

- Un fichier **S**équentiel (type #80) est une série de variables BASIC écrites les unes à la suite des autres. Ces variables sont appelées des "enregistrements", car elles sont enregistrées une à une. On y accède avec PUT et TAKE qui assurent l'échange entre variables BASIC et "enregistrements" et dont la longueur et le type doivent correspondre. La longueur du fichier croît à chaque fois que l'on ajoute de nouveaux enregistrements.

- Un fichier "**R**" (type #00) est une série de fiches de longueur fixe pour un fichier donné. Chaque fiche est constituée d'une série de "champs" de type et de longueur définis pour un fichier donné. On accède aux fiches avec PUT et TAKE. Ce sont les commandes "<" et ">" qui assurent l'échange entre variables BASIC et "champs", la longueur et le type devant correspondre. La longueur du fichier croît lorsque l'on ajoute de nouvelles fiches.

- Un "pseudo-fichier" d'accès **D**isque (type #01) est en fait une disquette constituée d'une série de secteurs auxquels on accède avec PUT et TAKE. La longueur du "pseudo-fichier" est fixe: c'est la taille de la disquette! Celle de chaque secteur aussi: c'est 256 octets. Chaque secteur peut être "découpé" en une série de "pseudo-champs" (de structure plus simple que ceux des fichiers "**R**") auxquels on accède avec "<" et ">". C'est l'utilisateur qui doit assurer la cohésion entre les variables BASIC utilisées et les fragments de secteurs qu'il "découpe".

Commandes utilisables avec les fichiers "**S**":

&(), APPEND, BUILD, CLOSE, JUMP, LTYPE, OPEN, PUT, REWIND, TAKE et TYPE

Commandes utilisables avec les fichiers "**R**":

&(), >, CLOSE, FIELD, LSET, OPEN, PUT, RSET et TAKE

Commandes utilisables avec les fichiers "**D**":

>, CLOSE, CRESEC, FIELD, FRSEC, LSET, OPEN, PMAP, PUT, RSET, SMAP et TAKE

Variables utilisées

A/F2		LLHH de la taille ou de l'augmentation de taille de **FI**
AX		adresse dans **FI** calculée à partir d'un offset AY
AY		adresse de la valeur du nombre dans le "Channel's own Data Buffer"
		puis cette valeur elle-même (octet, entier ou réel)
		coordonnées du secteur Y de la piste A
00/01		adresse réelle du début du "Channel Buffer" courant
02/03		adresse du début du "Channel's own Data Buffer" courant
04/05		adresse du début du "Descriptor Buffer" du fichier courant
		adresse du début du descripteur courant
		offset du point d'insertion d'un nouveau descripteur
06/07		adresse du début du "General Buffer"
		adresse du début de la fiche dans le "General Buffer"
		adresse du début des data dans le "General Buffer"
08/09		rang du secteur où se trouve la fiche depuis le début du fichier
0A		n° logique NL courant (de 0 à 63)
0B		flag type de fichier courant: "**R**" (#00) ou "**S**" (#80) ou "**D**" (#01)
28		flag de la variable ("chaîne" ou "nombre")
33/34		nombre d'enregistrements à sauter (n° de la fiche à atteindre)
33/34/F2	n° de la fiche (codé sur 3 octets)	
91/92		longueur de la chaîne
9E/9F		adresse de début des tableaux BASIC, c'est à dire, adresse de **FI**
A0/A1		adresse de fin des tableaux BASIC
C7/C8		adresse du haut de cible pour déplacer un bloc vers le haut
C9/CA		adresse du dernier octet du bloc à déplacer vers le haut
CE/CF		adresse du premier octet du bloc à déplacer vers le haut
D0/D4	ACC1	dont:
D0		longueur de la chaîne
D1/D2		adresse de la chaîne
D3/D4		adresse de la variable
F2		indication du n° de secteur libre
		flag "?" présent dans le nom_de_fichier cible sans homologue dans le nom_de_fichier source
		longueur de l'enregistrement (nombre de caractères restant à afficher)

F2/F3 adresse de la paire d'octets correspondant au NL dans la "Table NL" (cette paire d'octet est l'offset du début du "Channel Buffer" du fichier ouvert correspondant, F3 est nul si fichier est fermé)
adresse de l'entrée courante dans le "Field Buffer"
adresse dans le "Channel's own Data Buffer"
en général, adresse dans **FI** calculée à partir d'un offset AY
F3 longueur de la fiche
F4 flag "?" présent dans le nom de fichier source
F4/F5 nombre total de champs déclarés
adresse d'un emplacement libre dans le "Field Buffer"
F5 longueur de la chaîne (échange variable alphanumérique/champ)
index dans le "General Buffer"
F5/F6 coordonnées piste/secteur du secteur libre
F6 longueur de la variable (nombre d'octets à copier)
F7 HH de l'adresse du descripteur où est décrit le secteur contenant la fiche
valeur courante de l'index de lecture dans le tampon
F8 pointeur dans le descripteur courant
longueur d'enregistrement (nombre d'octets à copier)
F9/F3 offset du point d'insertion lors de l'extension de **FI**
F9 FTYPE #08 si "**R**" (b3 à 1) et #10 si "**S**" (b4 à 1) (ce sont les types SEDORIC: les "pseudo-fichiers" d'accès **D**isques n'en ont pas)

C000/C001/C002 DRIVE/PISTE/SECTEUR actifs
C003/C004 RWBUF adresse où sera lu/écrit le secteur
C009 DRVDEF n° du lecteur par défaut
C027 POSNMX position dans le secteur de catalogue
C028/C038 BUFNOM (drive, nom, extension, PSDESP et NSTOTP soit 17 octets)
C052/C053 (DESALO) ici, nombre de fiches d'un fichier à accès direct "**D**"
C054/C055 (FISALO) ici, longueur de fiches d'un fichier à accès direct "**D**"
C058/C059 nombre de secteurs supplémentaires requis
C05F index dans le descripteur
C076/C07F "General Field Buffer" (entrée courante du "Field Buffer") dont:
C076/C07A nom du champ (5 caractères significatifs)
C07B index de l'élément de pseudo-tableau
C07C n° logique pour ce champ
C07D offset entre le début de la fiche et le début de ce champ
index du début du champ dans l'enregistrement
longueur totale des champs du "Field Buffer" déjà explorés
C07E longueur du champ (1 si octet, 2 si entier, 5 si réel, longueur si alphanumérique)
C07F type de champ (#00 réel, #01 entier, #40 octet, #80 alphanumérique)
C080 sauvegarde du NL de la dernière commande FIELD
C081 compteur de longueur totale des champs du "Field Buffer"
puis #01, #40 ou #80 (si CLOSE)
C082 flag mis à #80 lors de CLOSE
flag du point d'entrée du sous-programme F4E6/F4E9/F4EC/F4EF:
#00 pour localiser un nom de champ
#01 pour vérifier qu'un nom de champ particulier existe

		#40 pour trouver une place pour un nouveau nom de champ
		#80 pour supprimer tous les noms de champs associés au fichier
C083		HH de l'adresse de Buffer
		longueur d'une fiche du fichier courant ("**R**") ou #00 ("**S**" ou "**D**")
C084		pointeur dans le dernier descripteur
C085/08/09	nombre d'octets	précédant la fiche dans le fichier (sur 3 octets)
C085		rang de l'octet de début de la fiche dans le secteur
C086		index de lecture dans la liste des coordonnées du descripteur courant
C087		n° du descripteur courant
C088		index dans le buffer lu de ou à écrire sur la disquette
C090/C09C	nom_de_fichier_ambigu	"Source" pour COPY* (drive, nom et extension)
C09D/C0A9	nom_de_fichier_ambigu	"Cible" pour COPY* (drive, nom et extension)
C100/C1FF	BUF1	buffer pour descripteur
C200/C2FF	BUF2	buffer pour bitmap
C300/C3FF	BUF3	buffer pour page de directory
CD25/CD2A		octets d'initialisation de **FI**: C6 C9 88 02 88 02

Structure du "Pseudo-Tableau" **FI** utilisé par les commandes de gestion de fichiers
Voir le schéma récapitulatif situé un peu plus loin.

Ce "Pseudo-Tableau" est placé au début de la zone des tableaux BASIC avec une taille initiale de #288 (648) octets. Son adresse est donc pointée par 9E/9F (début des tableaux BASIC). Dans les explications qui suivent, on parlera "d'offset" d'un point P pour désigner le nombre d'octets qui séparent ce point P du début de **FI** dont l'adresse est gardée en 9E/9F. Chaque octet de **FI** sera désigné par son "rang" dans **FI**: il s'agit de son n° d'ordre. Le premier octet est donc l'octet de rang #00. **FI** est constitué des 5 parties suivantes:

1) "En-tête" 8 octets de rang #00 à #07:

#00/01 **C6 C9** tableau de type "entier" et de nom "**FI**" (pour FIchiers)

#02/03 longueur totale de **FI**, c'est une sorte de "lien" permettant au BASIC de sauter au tableau suivant (ou la fin des tableaux, si aucun autre tableau n'est défini). Il vaut initialement #288. Je parle de "Pseudo-Tableau", car ces 4 premiers octets sont les seuls à correspondre à un tableau BASIC usuel. Après, ça se complique!

#04/05 offset du début du "Field Buffer" (voir plus loin) située juste avant la fin de **FI**. Cet offset permet l'ajout d'un nouveau "Channel Buffer" (voir plus loin). Il vaut initialement #288 et correspond à la fin de **FI** tant qu'une commande FIELD n'a pas été émise.

#06/07 donne le nombre total de champs dans le "Field Buffer". Il contient initialement #0000, comme on peut s'en douter!

2) "Table NL" #80 (soit 64 x 2 = 128) octets de rang #08 à #87 (8 à 135): à chaque n° logique (de 0 à 63) correspond une paire d'octets. L'octet de poids fort HH de chaque paire est à zéro si le fichier correspondant n'est pas ouvert. Cette paire d'octets représente l'offset nécessaire pour atteindre le début du "Channel Buffer" correspondant à ce NL.

3) "General Buffer": #200 (512) octets de rang #88 à #287 (136 à 647): c'est une zone générale tampon de 2 pages. Elle est utilisée par les fichiers de type "**R**" et "**S**", mais pas de type "**D**" qui sont un peu spéciaux.

Pour un fichier de type "**R**", le "General Buffer" est utilisé pour charger 2 secteurs consécutifs du fichier. Le premier de ces secteurs contient le début de la fiche. Une fiche donnée est copiée de ce "General Buffer" dans le "Channel's own Data Buffer" (voir plus loin).

Pour un fichier de type "**S**", le "General Buffer" est utilisé pour construire un enregistrement. Pour la commande TAKE, l'enregistrement ne vient pas directement de la disquette dans ce "General Buffer", mais via le "Channel's own Data Buffer". Pour la commande PUT, l'enregistrement est d'abord assemblé dans le "General Buffer" avant d'être envoyé dans le "Channel's own Data Buffer" puis sur la disquette.

4) "Channel Buffers": il y a un buffer de #121 (289) octets pour chaque fichier ouvert, qu'il soit de type "**S**", "**R**" ou "**D**". Ces buffers sont donc placés dans l'ordre d'ouverture des fichiers. Le rang du premier octet de chacun de ces "Channel Buffers" dans **FI** est indiqué par un offset conservé dans la "Table NL". Dans un "Channel Buffer" donné, chaque octet est défini par son "rang" (voir plus loin les pointeurs correspondants):

#00 flag indiquant le type du fichier: "**R**" (#00) ou "**S**" (#80) ou "**D**" (#01). Cet octet est copié en 0B pour le fichier courant

#01 longueur d'une fiche pour "**R**" et "**D**". Cette longueur est fixée lors de la première ouverture du fichier. Dans le cas de "**D**", la longueur est automatiquement fixée à #00, ce qui signifie #100, soit une page). Pour "**S**", cet octet est toujours #00. La longueur d'une fiche est seulement spécifiée lorsque cette fiche est écrite pour la première fois dans le fichier sur la disquette. La longueur de fiche du fichier courant est indiquée en C083

#02 n° du dernier descripteur du fichier, le n° du premier descripteur étant #00. Initialisé à #FF pour passage immédiat à #00

#03 index de lecture dans la liste des coordonnées du descripteur courant dans le "Descriptor Buffer". Initialisé à #0C (dans ce cas, pointe au début de la liste des coordonnées piste/secteur du premier descripteur)

#04 n° du descripteur courant (initialisé à #00)

#05 index dans le buffer qui vient d'être lu de la disquette ou dans le buffer qui va être écrit sur la disquette (initialisé à #00). Pour un fichier "**R**", ce buffer est le "General Buffer" de #200 octets et l'index est l'offset séparant le début de la fiche du début du champ dans la fiche. Pour un fichier "**S**" ou "**D**", ce buffer" est le "Channel's own Data Buffer" (voir plus loin) et l'index est l'offset séparant le début du buffer du début de l'enregistrement (fichier "**S**") ou du début du champ (fichier "**D**")

#06 n° de drive sur lequel le fichier a été ouvert. Le dernier NL utilisé est indiqué en 0A

#07/16 ces #10 (16) octets sont la copie d'une ligne de catalogue (nom_de_fichier etc...)

#17/116 ces #100 (256) octets correspondent au "**Channel's own Data Buffer**" ou buffer propre à un NL,

c'est à dire à un fichier. Pour les fichiers "**S**" et "**D**", c'est l'endroit où les data arrivent de la disquette et d'où les data vont vers la disquette. Pour un fichier "**D**", l'ensemble du buffer de #100 octets correspond à un secteur complet. Dans le cas d'un fichier "**R**", ce buffer est utilisé pour stocker une fiche donnée et cette information ne va pas ou ne vient pas directement de la disquette, mais est transférée via le "General Buffer"

#117/120 ces 10 octets ne sont pas utilisés, mais... repoussés vers le haut dans le cas des fichiers "**S**" ou "**R**" pour lesquels le ou les descripteur(s) sont insérés au point #117 (création d'un "Descriptor Buffer"). Dans ce cas, les octets de rang #117/216 contiennent le premier descripteur du fichier, les octets de rang #217/316 contiennent le descripteur suivant s'il existe etc... La micro-zone inutilisée #117/120, ainsi que les autres "Channel Buffers", le "Field Buffer" et les tableaux BASIC (s'ils existent) sont ainsi repoussés vers le haut pour charger tous les descripteurs. La zone initiale #117/120 n'est pas utilisée (gâchis!) si le "fichier" ouvert est de type "**D**" (pseudo-fichier sans descripteur).

5) "Field Buffer" lorsque la commande FIELD est utilisée pour la première fois, une zone de 100 octets (pour 10 entrées de 10 octets) est créée, tout à la fin de **FI**. Elle sert à garder les informations fournies par la commande FIELD. Ce "Field Buffer" est étendue si nécessaire (par multiples de 10 entrées) pour recevoir toutes les informations supplémentaires fournies par d'autres commandes FIELD. Les 10 octets d'une entrée, correspondant à un "Field Buffer" sont utilisés comme suit:

#00/04 nom du champ (5 caractères significatifs, les autres sont négligés)

#05 index du champ

#06 NL pour ce champ

#07 offset séparant le début de la fiche du début de ce champ

#08 longueur du champ (1 pour un champ octet **O**, 2 pour un champ entier **%**, 5 pour un champ réel **!** ou longueur réelle pour un champ alphanumérique **$**)

#09 type de champ (#00 pour **!**, #01 pour **%**, #40 pour **O** et #80 pour **$**)

Structure du "Pseudo-Tableau" FI

Zone des variables

(9E/9F) "**Entête**" (8 octets de #00 à #07):
(début des tableaux) **FI**
Lien du tableau suivant
Offset du "Field Buffer"
Nombre total de champs

(9E/9F) + #08 "**Table NL**" (128 octets de #08 à #87):
les 64 offsets des "Channel Buffers"
(2 octets par NL dont le HH est à zéro si le fichier est fermé)

(06/07) ou "**General Buffer**" (512 octets de #88 à #287):
(9E/9F) + #88 2 pages de tampon

(00/01)* Premier "**Channel Buffer**" (de taille variable):
(9E/9F) + #288 pointeurs et flags (7 octets #00/#06)
(02/03)* ligne de catalogue (16 octets #07/#16)
(04/05)* "**Channel's own Data Buffer**" (128 octets #17/#116)
"**Descriptor Buffer**" (#117/#216, #217/#316 etc...)
(un ou plusieurs descripteurs, 128 octets chacun)
10 octets non utilisés (#x17/#z16)

Deuxième "Channel Buffer"
(dans l'ordre des ouvertures)

Troisième "Channel Buffer" etc...

Offset et longueur "**Field Buffer**" (1 ou plusieurs blocs de 100 octets):
indiqués au début de **FI** 10 "entrées" de 10 octets chacune

---

Offset indiqué par Suite de la zone des tableaux
le lien au début de **FI** (tableaux proprement dits)

---

* Ces 3 adresses sont validées selon le NL courant

Pointeurs utilisés pour les buffers

Les octets 00/07 en page zéro de la RAM sont utilisés comme suit:

00/01 adresse réelle du début du "Channel Buffer" du fichier courant

02/03 idem + #17 = adresse du début du "Channel's own Data Buffer" (#100 octets) qui est utilisé comme tampon de lecture/écriture sur la disquette pour les fichiers "**S**" et "**D**" et comme tampon de travail sur la fiche courante pour un fichier "**R**"

04/05 idem + #117 = adresse du début du "Descriptor Buffer"

06/07 adresse du début du "General Buffer", c'est à dire de la zone tampon vraie, utilisée pour les échanges directs avec la disquette

Principe général de la gestion de fichiers

Dès que le premier n° logique (NL) est attribué à un fichier par la commande OPEN (voir page 76 du manuel pour plus d'informations), le "Pseudo-Tableau" **FI** est créé au début de la zone des tableaux BASIC (dont l'adresse est indiquée en 9E/9F) et a une longueur initiale totale de #288 octets. Au fur et à mesure de l'ouverture des fichiers et donc de l'attribution d'autres NL, **FI** est étendu par construction d'un buffer (tampon), le "Channel Buffer" pour chaque fichier (donc chaque NL). De plus lorsque la commande FIELD est utilisée pour la première fois, un buffer spécial, le "Field Buffer" (tampon relatif aux champs) est initialisé juste à la fin de **FI** afin de prévoir le stockage des informations en provenance de la dite commande FIELD. Ce "Field Buffer" sera étendu au fur et à mesure des besoins afin de recueillir davantage d'informations. Toutes ces créations (**FI**, "Channel Buffer" et "Field Buffer") se font par insertions impliquant un décalage des (vrais) tableaux BASIC vers le haut de la RAM.

Utilisation des différents buffers

1) pour un fichier de type "**S**équentiel"

Pour la commande TAKE, les data sont lus d'un secteur de la disquette dans les #100 octets du "Channel's own Data Buffer" pointé par 02/03. L'enregistrement courant est transféré dans le "General Buffer" (s'il tient sur deux secteurs, le deuxième secteur est chargé dans le "Channel's own Data Buffer" et le reste de l'enregistrement est copié dans le "General Buffer"). L'enregistrement complet, reconstitué dans le "General Buffer" est alors copié dans les variables BASIC. Dans ce cas l'octet d'ordre #05 du "Channel Buffer" est l'index dans le "Channel's own Data Buffer".

Pour la commande PUT, chaque variable indiquée est d'abord placée dans le "General Buffer" pour constituer un enregistrement. Le secteur courant du fichier, qui contient l'enregistrement où il faut écrire, est lu de la disquette dans le "Channel's own Data Buffer". L'enregistrement est recopié dans le "Channel's own Data Buffer", sauvé sur la disquette. Si nécessaire l'opération est répétée pour le secteur suivant si l'enregistrement tient sur deux secteurs. Comme précédemment, l'octet d'ordre #05 du "Channel Buffer" est l'index dans le "Channel's own Data Buffer".

2) pour un fichier de type "**R**" (direct)

Pour la commande TAKE, les data sont lus de deux secteurs consécutifs de la disquette (le premier contient le début de la fiche choisie) dans le "General Buffer". La fiche est alors copiée du "General Buffer" dans le "Channel's own Data Buffer". Dans ce cas l'octet d'ordre #05 du "Channel Buffer" est l'index dans le "General Buffer".

Pour la commande PUT, les data sont lus de 2 secteurs consécutifs de la disquette (le premier secteur contient le début de la fiche choisie) dans le "General Buffer". La fiche mise à jour est alors copiée du "Channel's own Data Buffer" dans le "General Buffer". Enfin, les 2 secteurs sont réécrit sur la disquette. Comme précédemment, l'octet d'ordre #05 du "Channel Buffer" est l'index dans le "General Buffer".

3) pour un pseudo-fichier de type "**D**isk access"

Pour la commande TAKE, les data sont lus d'un secteur de la disquette dans le "Channel's own Data Buffer" où il sera directement exploité sans passer par le "General Buffer". Dans ce cas l'octet d'ordre #05 du "Channel Buffer" est l'index dans le "Channel's own Data Buffer".

Pour la commande PUT, le secteur courant du fichier, qui est présent dans le "Channel's own Data Buffer" est sauvé sur la disquette sans passer par le "General Buffer". Comme précédemment, l'octet d'ordre #05 du "Channel Buffer" est l'index dans le "Channel's own Data Buffer".

Informations non documentées

Le système de gestion des fichiers de data utilise un "Pseudo-Tableau" situé tout au début de la zone des tableaux BASIC, de nom **FI** (FIchiers) et de type "Entier". Il est à noter que le manuel SEDORIC ne souffle mot de ce "tableau" **FI** et que si par malheur vous aviez l'idée d'en créer un pour votre propre usage, le système plantera au premier OPEN.

De même, à partir du moment où un OPEN a été utilisé et ce jusqu'au dernier CLOSE, il est absolument interdit d'utiliser la commande BASIC DIM! (ce n'est pas non plus dans le manuel). Ceci est dû au fait que les tableaux sont toujours créés au début de la zone des tableaux BASIC et vont faire "disparaître" le pseudo-tableau **FI**.

Dans le cas de OPEN R, lors de la création du fichier (première ouverture avec OPEN), la longueur de fiche doit être comprise entre #03 et #FF.

Début de l'analyse des commandes de gestion de fichiers

Calcule la valeur du pointeur visant l'offset correspondant à 0A dans la "Table NL"puis calcule l'adresse AX et F2/F3 correspondant à cet offset

(sous-programme F3CF-F3F2, appelé par la commande CLOSE et par les sous-programmes F4A8 et F4E6)

Rappel: ce pointeur vise l'offset du début du "Channel Buffer" correspondant au NL courant indiqué en 0A.

F3CF-	A5 0A	LDA 0A	prend le NL (de 0 à 63), en fait le dernier NL utilisé
F3D1-	0A	ASL	le multiplie par 2 (et force la retenue à zéro)
F3D2-	69 08	ADC #08	et ajoute 8 (varie donc de 8 à 134). NB: l'offset 134 vise le premier octet de la paire LLHH correspondant au NL 63

F3D4- D0 0B BNE F3E1 suite forcée en F3E1 pour calculer l'adresse correspondante, retournera finalement avec Y = #00 (voir OPEN pour explication)

Lit l'offset du début du "Field Buffer", puis calcule l'adresse correspondante AX et F2/F3

F3D6- A0 04 LDY #04 vise l'octet de rang #04 de **FI**

Lit l'offset présent aux octets de rang Y et Y+1, puis calcule l'adresse correspondante AX et F2/F3

F3D8- B1 9E LDA (9E),Y lit l'octet n° Y de **FI**
F3DA- 48 PHA et l'empile
F3DB- C8 INY
F3DC- B1 9E LDA (9E),Y lit l'octet n° Y+1 de **FI**
F3DE- A8 TAY
F3DF- 68 PLA AY est l'offset recherché
F3E0- 2C A0 00 BIT 00A0 continue en F3E3 pour calculer l'adresse correspondante, retournera finalement avec Y = #00 (voir OPEN pour explication)

Calcule l'adresse AX et F2/F3 correspondant à l'offset A et retourne avec Y = #00

F3E1- A0 00 LDY #00 force à zéro le HH de l'offset AY

Calcule l'adresse AX et F2/F3 correspondant à l'offset AY et retourne avec Y = #00

F3E3- 18 CLC prépare une addition
F3E4- 65 9E ADC 9E
F3E6- 85 F2 STA F2 calcule F2/F3 = AY + 9E/9F
F3E8- 48 PHA
F3E9- 98 TYA
F3EA- 65 9F ADC 9F
F3EC- 85 F3 STA F3 et AX = AY + 9E/9F
F3EE- AA TAX
F3EF- 68 PLA
F3F0- A0 00 LDY #00 et Y = #00
F3F2- 60 RTS

Vérifie l'existence de FI et le crée s'il n'existe pas encore
(sous-programme F3F3-F424, appelé par les commandes APPEND, CLOSE, LSET et RSET)

F3F3- A0 00 LDY #00 index pour lecture au début de la zone des tableaux
F3F5- A5 9F LDA 9F le HH de l'adresse de fin des variables BASIC
F3F7- C5 A1 CMP A1 est-il égal au HH de l'adresse de fin des tableaux?
F3F9- F0 07 BEQ F402 si oui (il n'y a pas de "Pseudo-Tableau" **FI** car alors HH serait plus grand d'au moins #02), continue en F402 pour en créer un
F3FB- B1 9E LDA (9E),Y sinon, teste le type du premier tableau:
F3FD- C8 INY le b7 des deux premiers octets...
F3FE- 31 9E AND (9E),Y est-il à 1? (c'est à dire de type "entier")

F400- 30 22 BMI F424 si oui (le buffer existe déjà), simple RTS en F424. Cette vérification est un peu cavalière (bogue)! Que se passe t'il s'il existe déjà un tableau entier? Il aurait été plus sûr de vérifier aussi que les 2 premiers octets indiquent bien le nom "FI" de **FI** et surtout d'indiquer dans le manuel que ce nom est réservé et qu'il est interdit d'utiliser la commande DIM entre le premier CLOSE et le dernier OPEN (voir "Non documenté" en F3CF).

Création de "**FI**" s'il n'existe pas encore

Ce "Pseudo-Tableau" doit être le premier de la zone des tableaux BASIC.

F402- A6 9E LDX 9E
F404- A4 9F LDY 9F XY, adresse du premier octet de la zone des tableaux
F406- A9 02 LDA #02 pour créer un "Pseudo-Tableau" de #288 octets:
F408- 85 F2 STA F2 HH de la taille de **FI** à créer
F40A- A9 88 LDA #88 LL de la taille de **FI** à créer
F40C- 20 56 F4 JSR F456 décale tous les tableaux BASIC qui existent déjà, à partir de l'adresse XY, de #288 octets vers le haut
F40F- A0 00 LDY #00
F411- 8C 81 C0 STY C081 force C081 à zéro (longueur du "Field Buffer")
F414- 98 TYA
F415- 91 9E STA (9E),Y force à zéro les 256 premiers octets de **FI** situés
F417- C8 INY à partir de l'adresse présente en 9E/9F
F418- D0 FB BNE F415 (entre autres, la "Table NL")
F41A- A0 05 LDY #05
F41C- B9 25 CD LDA CD25,Y copie les 6 octets présents en CD25/CD2A
F41F- 91 9E STA (9E),Y à cette adresse, c'est à dire initialise le début
F421- 88 DEY de **FI** avec C6 C9 88 02 88 02
F422- 10 F8 BPL F41C qui représentent un tableau de type "entier", de nom "FI" et de longueur #288 octets (offset pour trouver le tableau suivant)
F424- 60 RTS et retourne

Extension de "**FI**" par insertion de AF2 octets au point d'offset XY
(sous-programme F425-F472, appelé par les sous-programmes F4E6, F684 et FACB)

Ce sous-programme explore toute la "Table NL", ainsi qu'une partie de l'en-tête de **FI** et teste si l'offset indiqué par chaque paire d'octets (correspondant par exemple à un n° logique NL et visant le "Channel Buffer" afférent) est supérieure ou égale à la valeur de XY. Si c'est le cas, le sous-programme ajoute AF2 octets à la valeur indiquée par la paire. En effet, ces offsets permettent de calculer une adresse dans la partie haute de **FI** et toute extension de **FI** à partir du point d'offset XY se fait par décalage vers le haut. Il faut donc augmenter d'autant tous les offsets se référant à des zones qui seront déplacées. En clair tous les offsets supérieurs ou égaux à XY seront incrémentés de la valeur indiquée en AF2. Après cette mise à jour de la "Table NL", le sous-programme termine en décalant de AF2 octets vers le haut, à partir du point d'offset XY, la partie haute de **FI** et tous les tableaux BASIC se trouvant à la suite.

F425- 48 PHA empile A (LL de l'augmentation de taille)
F426- 84 F3 STY F3 garde Y dans F3 (HH de l'offset du point d'insertion)
F428- 86 F9 STX F9 et X dans F9 (LL de l'offset du point d'insertion)
F42A- 18 CLC C = 0 représente une retenue pour une soustraction

F42B- A0 86 LDY #86 vise l'octet n°134, c'est à dire le premier octet de la dernière paire de la "Table NL". Y visera successivement les 64 paires d'octets de la "Table NL", puis les octets d'en-tête à l'exclusion des deux premiers qui représentent le nom **FI** du "Pseudo-Tableau" de type "entier"
F42D- B1 9E LDA (9E),Y lors de la comparaison qui suit, C restera à 0
F42F- C5 F9 CMP F9 (retenue) si l'octet lu (premier octet d'une paire) est inférieur au LL de l'offset de la fin de la "Zone Buffer", sinon C passera à 1 (pas de retenue à reporter lors de la soustraction qui va suivre)
F431- C8 INY vise octet suivant, c'est à dire le deuxième d'une paire
F432- B1 9E LDA (9E),Y de même lors de cette soustraction, C restera à 0
F434- E5 F3 SBC F3 si l'octet lu (deuxième octet d'une paire) est inférieur au HH de l'offset de la fin de la "Zone Buffer"), sinon C passera à 1
F436- 90 0F BCC F447 continue en F447 si paire d'octet < XY (pas de mise à jour, la zone correspondante est en dessous du point d'insertion et ne sera pas déplacée)
F438- 88 DEY sinon, vise à nouveau le premier octet de la paire
F439- 18 CLC prépare une addition
F43A- 68 PLA récupère une copie de A
F43B- 48 PHA (LL de l'augmentation de taille)
F43C- 71 9E ADC (9E),Y calcule A = A + premier octet de la paire
F43E- 91 9E STA (9E),Y réécrit la nouvelle valeur dans **FI**
F440- C8 INY
F441- B1 9E LDA (9E),Y calcule A = F2 + deuxième octet de la paire
F443- 65 F2 ADC F2
F445- 91 9E STA (9E),Y réécrit la nouvelle valeur dans **FI**. Calcule ainsi: offset = offset + amplitude du décalage vers le haut.
F447- 88 DEY vise à nouveau le premier octet de la paire
F448- 88 DEY
F449- 88 DEY vise le premier octet de la paire précédente
F44A- D0 E1 BNE F42D reboucle en F42D tant qu'il n'atteint pas la première paire qui correspond au nom du "Pseudo-Tableau" qui n'est donc pas affecté!

Calcule l'adresse du début de la zone à décaler vers le haut

F44C- 8A TXA en entrée X = LL et
F44D- 65 9E ADC 9E F3 = HH de l'offset du point d'insertion
F44F- AA TAX calcule XY (adresse de la zone à décaler)
F450- A5 F3 LDA F3 = XF3 (offset du point d'insertion)
F452- 65 9F ADC 9F + adresse de début de **FI**
F454- A8 TAY
F455- 68 PLA récupère A (LL de l'augmentation de taille)

Décale à partir de l'adresse XY, de AF2 octets vers le haut

Calcule les adresses nécessaires au déplacement du bloc, puis effectue ce déplacement pour avoir la place nécessaire pour créer "**FI**" ou pour insérer un "Channel Buffer" ou pour créer (ou étendre) le "Field Buffer".

F456- 86 CE STX CE initialise CE/CF avec l'adresse du premier octet situé
F458- 84 CF STY CF au début du bloc à déplacer vers le haut
F45A- 18 CLC prépare une addition

F45B- 65 A0 ADC A0 initialise C7 avec LL de l'adresse haut de cible
F45D- 85 C7 STA C7 (LL de l'adresse de fin des tableaux + A octets)
F45F- 48 PHA
F460- A5 A0 LDA A0 initialise C9/CA avec l'adresse du dernier octet
F462- A4 A1 LDY A1 du haut du bloc à déplacer (fin des tableaux)
F464- 85 C9 STA C9 (bogue: le LDY est inutile car écrasé plus loin!)
F466- A5 A1 LDA A1
F468- 85 CA STA CA initialise C8 avec HH de l'adresse haut de cible
F46A- 65 F2 ADC F2 (HH de l'adresse de fin des tableaux + F2 pages
F46C- 85 C8 STA C8 + éventuelle retenue précédente)
F46E- A8 TAY
F46F- 68 PLA AY, adresse du haut de la cible
F470- 4C 5C D1 JMP D15C JSR C3F4/ROM décale un bloc mémoire vers le haut (CE/CF premier octet du bas, C9/CA dernier octet du haut, C7/C8 et AY cible vers le haut, "OUT_OF_MEMORY_ERROR" si adresse cible > adresse du bas des chaînes A2/A3, revient avec nouveau début - #100 en C7/C8 et nouvelle fin enA0/A1 (haut des tableaux))

Vérifie l'existence de **FI**, la validité du NL présent dans A et si le fichier est bien déjà ouvert (sous-programme F473-F4A4, appelé par les commandes &, CLOSE, FIELD, OPEN, REWIND et TAKE)

F473- 48 PHA sauvegarde A (NL)
F474- 20 F3 F3 JSR F3F3 vérifie l'existence de **FI** et le crée s'il n'existe pas encore
F477- 68 PLA récupère A
F478- AA TAX et le passe dans X
F479- 18 CLC flag "vérifier que le fichier est déjà ouvert"
F47A- 08 PHP sauvegarde les indicateurs 6502 dont C
F47B- 90 0A BCC F487 suite forcée en F487 pour vérifier que le fichier est ouvert

Vérifie l'existence de **FI**, la validité du NL indiqué à TXTPTR et si le fichier est bien déjà ouvert

F47D- 18 CLC flag "vérifier que le fichier est déjà ouvert"
F47E- 24 38 BIT 38 et saute l'instruction suivante

Vérifie l'existence de **FI**, la validité du NL indiqué à TXTPTR et si le fichier n'est pas déjà ouvert

F47F- 38 SEC flag "vérifier que fichier n'est pas encore ouvert"
F480- 08 PHP sauvegarde les indicateurs 6502 dont C
F481- 20 F3 F3 JSR F3F3 vérifie l'existence de **FI** et le crée s'il n'existe pas encore (c'est idiot dans certains cas, "CLOSE" par exemple)
F484- 20 7F D2 JSR D27F CF17/ROM et CF09/ROM et D8CB/ROM évalue un nombre entier à TXTPTR et le retourne dans X (en principe un NL)
F487- E0 40 CPX #40 le NL est-il > 63?
F489- B0 1A BCS F4A5 si oui, "ILLEGAL_QUANTITY_ERROR"
F48B- 86 0A STX 0A sinon, place ce NL dans 0A
F48D- 20 CF F3 JSR F3CF calcule l'adresse F2/F3 de la paire d'octets correspondant au NL

dans la "Table NL" et revient avec Y = #00

F490- C8 INY qui passe donc à 1
F491- 28 PLP récupère C (flag "déjà ouvert ou pas")
F492- B1 F2 LDA (F2),Y lit l'octet qui suit le pointeur (deuxième octet de la paire)
F494- D0 0A BNE F4A0 continue en F4A0 si le fichier est ouvert
F496- B0 0A BCS F4A2 si le fichier est fermé, teste si c'est bien ce que l'on attendait, c'est à dire si C = 1, si c'est le cas, continue en F4A2
F498- A2 0D LDX #0D si ce n'est pas le cas (C = 0), prépare une
F49A- 2C A2 0E BIT 0EA2 "FILE_NOT_OPEN_ERROR" et continue en F49D
F49B- A2 0E LDX #0E pour "FILE_ALREADY_OPEN_ERROR"
F49D- 4C 7E D6 JMP D67E incrémente X et traite l'erreur n° X

F4A0- B0 F9 BCS F49B le fichier est déjà ouvert, teste si on attendait un fichier fermé, c'est à dire si C = 1, si oui, continue en F49B
F4A2- 4C 9E D3 JMP D39E XCRGOT relit le caractère à TXTPTR (sans incrémenter TXTPTR = CHRGOT), puis le convertit en MAJUSCULE, les espaces sont sautés, Z = 1 si fin d'instruction (0 ou :), C = 0 si caractère chiffre 0 à 9 (soit #30 à #39), sinon C = 1, Y et X inchangés. C'est la seule sortie normale (sans erreur) de ce sous-programme

F4A5- 4C 20 DE JMP DE20 "ILLEGAL_QUANTITY_ERROR"

Place l'adresse du début du "Channel Buffer" correspondant au NL en 00/01, celle du début du "Channel's own Data Buffer" en 02/03, celle du "Descriptor Buffer" en 04/05 celle du début du "General Buffer" en 06/07 et enfin, met à jour C083 (longueur d'une fiche ou #00) et 0B (flag "S/R/D") puis retourne avec Y = #00

Place l'adresse du début du "General Buffer" en 06/07

Ce buffer de 2 pages (#200 octets) est utilisé pour les entrées/sorties directes avec la disquette (sous-programme F4A8-F4DB, appelé par les commandes &, >, BUILD, FIELD, LSET, PUT, REWIND, RSET et TAKE).

F4A8- A9 88 LDA #88 AY = #0088 (136 décimal, soit
F4AA- A0 00 LDY #00 en-tête 8 + "Table NL" 64 x 2)
F4AC- 20 E3 F3 JSR F3E3 calcule l'adresse AX (avec copie dans F2/F3) correspondant à l'offset AY dans **FI** et retourne avec Y = #00
F4AF- 85 06 STA 06 résultat qui est stocké en 06/07 (c'est l'adresse
F4B1- 86 07 STX 07 du début du "General Buffer")

Calcule Y qui vise dans la "Table NL" la paire LLHH (offset du "Channel Buffer") correspondant au NL

F4B3- A5 0A LDA 0A NL
F4B5- 0A ASL multiplié par 2
F4B6- 69 08 ADC #08 plus 8, le résultat est copié dans Y
F4B8- A8 TAY

Place l'adresse du début du "Channel Buffer" correspondant au NL en 00/01, celle du début du "Channel's

own Data Buffer" en 02/03 et celle du "Descriptor Buffer" en 04/05

F4B9- 20 D8 F3 JSR F3D8 lit dans la "Table NL" l'offset visant le début du "Channel Buffer" correspondant au NL, calcule l'adresse AX correspondant à cet offset et retourne avec Y = #00
F4BC- 85 00 STA 00 copie cette adresse en 00/01
F4BE- 18 CLC (début du "Channel Buffer")
F4BF- 69 17 ADC #17
F4C1- 85 02 STA 02 copie cette adresse + #17 en 02/03
F4C3- 85 04 STA 04 (début du buffer proprement dit
F4C5- 8A TXA ou "Channel's own Data Buffer")
F4C6- 85 01 STA 01
F4C8- 69 00 ADC #00 copie cette adresse + #117 en 04/05
F4CA- 85 03 STA 03 (début du "Descriptor Buffer")
F4CC- 69 01 ADC #01
F4CE- 85 05 STA 05

Met à jour C083 (longueur d'une fiche ou #00) et 0B (flag "**S**/**R**/**D**") puis retourne avec Y = #00
Cette partie est inutile lorsque le fichier est ouvert pour la première fois

F4D0- C8 INY qui passe donc à 1
F4D1- B1 00 LDA (00),Y lit l'octet de rang #01 dans le "Channel Buffer"
F4D3- 8D 83 C0 STA C083 et le copie en C083 (longueur d'une fiche ou #00)
F4D6- 88 DEY qui repasse à 0
F4D7- B1 00 LDA (00),Y lit l'octet de rang #00 dans le "Channel Buffer"
F4D9- 85 0B STA 0B et le copie en 0B (flag "**S**/**R**/**D**")
F4DB- 60 RTS retourne avec Y = #00

Ajoute A au contenu de 02/03 (appelé de F5D9, F5F4 et FCAA)
(sous-programme F4DC-F4E5, appelé par les commandes >, LSET et RSET)

F4DC- 18 CLC prépare une addition
F4DD- 65 02 ADC 02
F4DF- 85 02 STA 02
F4E1- 90 02 BCC F4E5 02/03 = 02/03 + A
F4E3- E6 03 INC 03
F4E5- 60 RTS

Série de sous-programmes pour générer ou localiser un nom de champ particulier ou pour supprimer tous les noms de champ associés avec le fichier courant

Supprime tous les noms de champ spécifiés pour le fichier courant (appelé de FBB9)
(sous-programmes F4E6-F5B9, appelés par les commandes >, CLOSE, FIELD, LSET, RSET et TAKE)

F4E6- A9 80 LDA #80 pour supprimer tous les noms de champs associés
F4E8- 2C A9 00 BIT 00A9 au fichier courant, continue en F4F1

Localise un nom de champ particulier associé au fichier courant dans le "Field Buffer" (appelé de F5C8 et FC85)

F4E9- A9 00 LDA #00 pour localiser un nom de champ (erreur s'il
F4EB- 2C A9 01 BIT 01A9 n'existe pas), continue en F4F1

Vérifie si le nom de champ existe déjà pour le fichier courant (appelé de FC2B)

F4EC- A9 01 LDA #01 pour vérifier qu'un nom de champ particulier associé au fichier existe
F4EE- 2C A9 40 BIT 40A9 (C = 0 s'il n'existe pas et C = 1 s'il existe déjà), continue en F4F1

Réserve un emplacement disponible pour un nouveau nom de champ associé au fichier courant (appelé de FC30)

F4EF- A9 40 LDA #40 pour trouver une place pour un nouveau nom de champ
F4F1- 8D 82 C0 STA C082 sauve en C082 le flag d'entrée dans cette routine
F4F4- A9 06 LDA #06 pour viser le nombre total de champs déclarés
F4F6- 20 E1 F3 JSR F3E1 calcule l'adresse F2/F3 correspondant à l'octet de rang #06 de **FI** (nombre total de champs) et retourne avec Y = #00
F4F9- B1 F2 LDA (F2),Y lit LL de ce nombre total de champs
F4FB- 85 F4 STA F4 et le copie en F4
F4FD- C8 INY
F4FE- B1 F2 LDA (F2),Y lit HH de ce nombre total de champs
F500- 85 F5 STA F5 et le copie en F5
F502- 20 D6 F3 JSR F3D6 lit l'offset du "Field Buffer" (octets de rang #04 et #05 de **FI**), calcule l'adresse correspondante F2/F3 et retourne avec Y = #00

Lit tous les noms de champ, localise ceux qui sont associés au fichier courant, effectue le travail spécifié par le flag sauvé en C082

F505- A5 F4 LDA F4 teste si tous les bits de F4 et de F5 sont nuls
F507- 05 F5 ORA F5 c'est à dire, si tous les noms de champ ont été
F509- F0 54 BEQ F55F vérifiés: si oui, continue en F55F
F50B- A5 F4 LDA F4 sinon,
F50D- D0 02 BNE F511 décrémente F4/F5
F50F- C6 F5 DEC F5 (le nombre de noms de champ restant
F511- C6 F4 DEC F4 à vérifier)
F513- A0 06 LDY #06 Y = #06 vise le NL de ce nom de champ
F515- 2C 82 C0 BIT C082 teste si les b7 et b6 du flag C082 sont nuls
F518- 10 0C BPL F526 si b7 est nul, continue en F526

Supprime le nom de champ s'il est associé au fichier courant

F51A- 38 SEC si b7 pas nul, prépare une soustraction
F51B- B1 F2 LDA (F2),Y lit l'octet de rang #06 de l'entrée courante dans le "Field Buffer", c'est à dire le NL pour ce champ
F51D- E5 0A SBC 0A est-ce le NL du fichier courant?
F51F- D0 31 BNE F552 sinon, continue en F552 pour le nom suivant
F521- A8 TAY si oui, force Y à zéro

F522- 91 F2 STA (F2),Y ainsi que l'octet de rang #00 de l'entrée courante dans le "Field Buffer", c'est à dire le premier caractère du nom du champ
F524- F0 2C BEQ F552 suite forcée en F552 pour le nom suivant

Suite de l'analyse du flag C082

F526- 50 20 BVC F548 si le b6 est également nul, continue en F548

Localise un emplacement libre pour un nouveau nom de champ

F528- A0 00 LDY #00 si le b6 de C082 n'est pas nul,
F52A- B1 F2 LDA (F2),Y teste l'octet de rang #00 de l'entrée courante dans le "Field Buffer", c'est à dire le premier caractère du nom du champ
F52C- D0 24 BNE F552 si pas #00, continue en F552 pour le nom suivant

Un emplacement libre a été trouvé pour le nouveau nom de champ, copie l'adresse de cette "entrée" libre F2/F3 en F4/F5

F52E- A5 F2 LDA F2 si le premier caractère du nom de champ est nul,
F530- A4 F3 LDY F3 copie F2/F3 en F4/F5
F532- 85 F4 STA F4
F534- 84 F5 STY F5
F536- 60 RTS et retourne

Le nom de champ recherché, associé au fichier courant a été trouvé, lit les 10 octets de "l'entrée" trouvée et les copie en C076/C07F dans le "General Field Buffer"

F537- A0 09 LDY #09 pour copier 10 octets (C = 1 en entrée)
F539- AD 82 C0 LDA C082 teste si l'octet en C082 est différent de zéro
F53C- D0 F0 BNE F52E si oui, termine en F52E: il fallait seulement vérifier l'existence de ce nom de champ, retourne avec l'adresse de l'entrée correspondante dans le "Field Buffer" en F4/F5 et avec C = 1 (flag "trouvé")
F53E- B1 F2 LDA (F2),Y lit l'octet de rang Y de "l'entrée"
F540- 99 76 C0 STA C076,Y et le copie en C076/C07F (travaille par la fin)
F543- 88 DEY vise l'octet précédent
F544- 10 F8 BPL F53E reboucle en F53E tant qu'il en reste à copier
F546- 30 E6 BMI F52E fini: termine en F52E avec l'adresse de l'entrée correspondante dans le "Field Buffer" en F4/F5 et avec C = 1 (flag "trouvé")

Suite de l'analyse du flag C082: il s'agit de localiser ou de vérifier un nom de champ

F548- 88 DEY décrémente Y (passera successivement de #06 à #00)
F549- 30 EC BMI F537 continue en F537 avec C = 1 lorsque Y devient négatif (fini: tous les octets sont identiques) pour y lire les 10 octets de "l'entrée" trouvée et les copier en C076/C07F dans le "General Field Buffer"
F54B- B1 F2 LDA (F2),Y lit l'octet de rang Y de l'entrée courante dans le "Field Buffer", c'est à dire le NL, l'index du champ et enfin son nom
F54D- D9 76 C0 CMP C076,Y et le compare avec l'octet homologue en C076/C07C

F550- F0 F6 BEQ F548 reboucle en F548 s'ils sont identiques, sinon passe à l'examen du nom de champ suivant

Cherche le nom de champ suivant

F552- A9 0A LDA #0A pour incrémenter le pointeur de 10 octets
F554- 18 CLC prépare une addition
F555- 65 F2 ADC F2
F557- 85 F2 STA F2
F559- 90 AA BCC F505 calcule F2/F2 = F2/F3 + #0A
F55B- E6 F3 INC F3
F55D- B0 A6 BCS F505 et reprend d'office en F505

Tous les noms de champ ont été examinés

F55F- 2C 82 C0 BIT C082 teste si le b6 de CO82 est à zéro (pas de réservation
F562- 50 48 BVC F5AC pour un nouveau nom de champ) si oui, continue en F5AC

Il fallait trouver un emplacement libre pour un nouveau nom de champ, comme rien n'a été trouvé, ajoute 100 octets au "Field Buffer" pour loger 10 nouveaux noms de champ

F564- A0 04 LDY #04 vise l'offset du début du "Field Buffer"
F566- B1 9E LDA (9E),Y lit l'octet de rang #04 de **FI**
F568- 48 PHA l'empile
F569- AA TAX et garde une copie dans X
F56A- C8 INY vise l'octet suivant
F56B- B1 9E LDA (9E),Y lit l'octet de rang #05 de **FI**
F56D- 48 PHA l'empile
F56E- A8 TAY et garde une copie dans Y
F56F- 8A TXA récupère le premier des 2 dans A
F570- 20 E3 F3 JSR F3E3 calcule l'adresse F2/F3 correspondant à l'offset AY dans **FI**, c'est à dire l'adresse du "Field Buffer" et retourne avec Y = #00
F573- 20 2E F5 JSR F52E copie F2/F3 en F4/F5 (début du nouveau buffer et sera utilisé plus loin pour mettre à zéro les 10 "entrées" correspondantes)
F576- 68 PLA
F577- A8 TAY récupère Y
F578- 68 PLA
F579- AA TAX récupère X (XY, offset du début du "Field Buffer")
F57A- A9 00 LDA #00
F57C- 85 F2 STA F2 AF2 = #0064 (10 "entrées" de 10 octets = 100)
F57E- A9 64 LDA #64
F580- 20 25 F4 JSR F425 extension du "Field Buffer" par insertion de #64 octets au point d'offset XY, avec mise à jour de la "Table NL"

Ajuste l'offset du début du "Field Buffer" après extension

Cette partie corrige la "correction" effectuée par le sous-programme F425 qui a ajouté #64 à la paire d'octets 04/05 de **FI** alors qu'il ne fallait pas!

```
F583-       38          SEC   prépare une soustraction
F584-       A0 04       LDY #04
F586-       B1 9E       LDA (9E),Y  lit l'octet de rang #04 de FI
F588-       E9 64       SBC #64     en retire #64
F58A-       91 9E       STA (9E),Y  et le remet en place
F58C-       C8          INY
F58D-       B1 9E       LDA (9E),Y  reporte la retenue sur l'octet de rang #05
F58F-       E9 00       SBC #00
F591-       91 9E       STA (9E),Y  et le remet en place
```

Ajoute 10 au nombre total d'emplacements pour noms de champ

```
F593-       A0 06       LDY #06     vise le nombre total d'emplacements possibles
F595-       A9 09       LDA #09     C = 1 en entrée donc ajoute 10 en fait
F597-       71 9E       ADC (9E),Y  incrémente de 10 le nombre d'emplacements
F599-       91 9E       STA (9E),Y  et le remet en place
F59B-       C8          INY
F59C-       B1 9E       LDA (9E),Y  reporte la retenue sur l'octet HH
F59E-       69 00       ADC #00
F5A0-       91 9E       STA (9E),Y  et le remet en place
```

Efface les 10 nouvelles entrées

```
F5A2-       A9 00       LDA #00
F5A4-       A0 63       LDY #63
F5A6-       91 F4       STA (F4),Y  force à zéro les 64 octets situés à partir
F5A8-       88          DEY   de l'adresse indiquée en F4/F5
F5A9-       10 FB       BPL F5A6
F5AB-       60          RTS
```

Rien trouvé, on ne recherchait pas un emplacement libre, il fallait soit localiser ou vérifier l'existence d'un nom de champ, soit supprimer les noms de champs associés au fichier courant:

F5AC- 30 06 BMI F5B4 suite de l'analyse du flag C082: RTS en F5B4 si b7 de C082 est à 1 (retourne car il n'y a pas ou il n'y a plus de nom de champ à supprimer, c'est à dire rien de plus à faire)
F5AE- 4E 82 C0 LSR C082 teste si le b0 est nul (en le poussant dans C)
F5B1- 90 02 BCC F5B5 si oui, continue en F5B5 (localisation: le nom de champ spécifié était requis, mais n'a pas été trouvé, génère une erreur)
F5B3- 18 CLC sinon, force C = 0 (pas trouvé) (vérification: il fallait simplement vérifier si le nom existait, ce qui n'est pas le cas)
F5B4- 60 RTS et retourne
F5B5- A2 13 LDX #13 pour "UNKNOW'N_FIELD_NAME_ERROR"
F5B7- 4C 7E D6 JMP D67E incrémente X et traite l'erreur n° X

EXÉCUTION DE LA COMMANDE SEDORIC ">"

(Fichiers "**R**" et "**D**")

(sous-programme F5BA-F683)

Rappel de la syntaxe

Nom_de_champ(index) > Nom_de_variable

Lit le champ de nom spécifié et en affecte la valeur à la variable indiquée. Le fichier doit évidemment être ouvert, sinon "FILE_NOT_OPEN_ERROR" et être du bon type, sinon "FILE_TYPE_MISMATCH_ERROR". Il faut d'abord utiliser la commande TAKE pour mettre à jour le n° de fiche et le NL. Le nom_de_champ(index) doit avoir été défini pour le NL courant (sinon "UNKNOW'N_FIELD_NAME_ERROR"). Le champ et la variable doivent être du même type (alphanumérique ou numérique) et de la même longueur, sinon "TYPE_MISMATCH_ERROR". En ce qui concerne le type numérique, il y a tolérance entre les types réel, entier ou octet.

Variables utilisées

En voir la liste et les fonctions dans les généralités sur la gestion de fichiers de data situées devant le sous-programme F3CF.

Non ou mal documenté

Cette commande est mal sécurisée: elle ne peut pas être utilisée avec un fichier de type "**S**", mais cela n'est pas vérifié. De plus, il n'est pas testé si un fichier "**R**" à accès direct est ouvert et qu'une fiche à été chargée à l'aide de la commande TAKE NL, n° de fiche ou si un fichier d'accès **D**isque est ouvert et qu'un secteur a été chargé à l'aide de la commande TAKE NL, piste, secteur. La validité de la fiche n'est pas vérifiée et il se peut en fait qu'elle ne contienne aucun data valable.

Rappel: un élément de pseudo-tableau est accepté comme nom de champ valable (voir commande FIELD).

Analyse de la syntaxe et saisie des paramètres

F5BA- 20 40 F6 JSR F640 vide le "General Field Buffer", puis décode le nom de champ et s'il s'agit d'un pseudo-tableau, l'index de cet élément
F5BD- 20 F3 F3 JSR F3F3 vérifie l'existence de **FI** au début des tableaux et le crée s'il n'existe pas encore (!!!)
F5C0- A9 D3 LDA #D3 token ">"
F5C2- 20 2E D2 JSR D22E JSR D067/ROM puis D3A1/RAM overlay demande un ">" à TXTPTR, lit le caractère suivant et le convertit éventuellement en MAJUSCULE
F5C5- 20 2E ED JSR ED2E prend dans AY, dans B8/B9 et dans D3/D4 l'adresse de la variable à TXTPTR
F5C8- 20 E9 F4 JSR F4E9 localise le nom de champ particulier associé au fichier courant dans le "Field Buffer" (sinon localisé, "UNKNOW'N_FIELD_NAME_ERROR")
F5CB- 20 7A F6 JSR F67A compare les types d'enregistrement et de variables ("TYPE_MISMATCH_ERROR" s'ils sont incompatibles)
F5CE- AD 7C C0 LDA C07C lit l'octet en C07C (NL)
F5D1- 85 0A STA 0A et le copie en 0A
F5D3- 20 A8 F4 JSR F4A8 place l'adresse du début du "Channel Buffer" correspondant au NL en 00/01, celle du début du "Channel's own Data Buffer" en 02/03, celle du "Descriptor Buffer" en

04/05, celle du début du "General Buffer" en 06/07 et enfin, met à jour C083 (longueur d'une fiche ou #00) et 0B (flag "**R**/**D**") puis retourne avec Y = #00

F5D6- AD 7D C0 LDA C07D lit l'index de début du champ dans la fiche
F5D9- 20 DC F4 JSR F4DC ajoute A au contenu de 02/03, c'est à dire vise le début du champ dans le "Channel's own Data Buffer"

Teste s'il s'agit d'un fichier de type "**S**", "**R**" ou "**D**"

F5DC- A6 0B LDX 0B flag "**S**/**R**/**D**"
F5DE- CA DEX teste si #01, c'est à dire si passe à zéro
F5DF- D0 08 BNE F5E9 sinon (pas type "**D**"), continue en F5E9

Fichier de type accès **D**isque

F5E1- AE 7F C0 LDX C07F type de champ
F5E4- AC 7E C0 LDY C07E longueur de champ
F5E7- D0 0E BNE F5F7 suite forcée en F5F7

Fichier de type "**R**" (accès direct) ou **S**équentiel

F5E9- A0 00 LDY #00 vise le premier octet du "Channel's own Data Buffer" ("**R**") ou du "General Buffer" ("**S**")
F5EB- B1 02 LDA (02),Y type de champ
F5ED- C8 INY
F5EE- AA TAX
F5EF- B1 02 LDA (02),Y longueur de champ
F5F1- A8 TAY
F5F2- A9 02 LDA #02 ajoute #02 au pointeur 02/03 du "Channel's own
F5F4- 20 DC F4 JSR F4DC Data Buffer" (fichier "**R**") ou du "General Buffer" (ficher "**S**") pour viser la valeur du champ

Met le data dans la variable BASIC

F5F7- 84 F5 STY F5 longueur de champ
F5F9- 8A TXA type de champ
F5FA- 30 29 BMI F625 continue en F625 si champ de type "chaîne"
F5FC- D0 0C BNE F60A continue en F60A si champ de type "entier" et continue en F5FE si champ de type "réel"

Champ de type "réel"

Modification par Ray de cette partie de la commande SEDORIC ">" afin de permettre l'utilisation correcte d'un nombre réel à partir d'un fichier (12 octets différents, indiqués en gras).

F5FE **A5 02** LDA #02 AY, adresse de la valeur du nombre réel
F600 **A4 03** LDY 03 dans le "Channel's own Data Buffer"
F602 **20 BA D2** JSR D2BA DE7B/ROM, place dans ACC1 la valeur pointée par AY
F605 **4C 20 F6** JMP F620 reprise en F620 du cours normal de la commande ">" pour

terminer

F608 **EA EA** NOP NOP la copie de ce nombre réel dans la variable BASIC

Champ de type "entier" ou simple octet

F60A- 0A ASL teste le b6 du type de champ
F60B- 0A ASL C = 0 si "entier" et C = 1 si simple octet
F60C- A0 00 LDY #00 index de lecture dans le "Channel's own Data Buffer"
F60E- B1 02 LDA (02),Y lit le simple octet ou le LL d'un nombre "entier"
F610- A8 TAY le copie dans Y si c'est un simple octet
F611- 85 F2 STA F2 et dans F2 pour le cas où c'est le LL d'un "entier"
F613- A9 00 LDA #00 HH pour le cas où c'est un simple octet
F615- B0 06 BCS F61D continue en F61D si c'est un simple octet
F617- A0 01 LDY #01 index l'octet suivant
F619- B1 02 LDA (02),Y lit le HH d'un nombre "entier"
F61B- A4 F2 LDY F2 récupère le LL du nombre "entier"
F61D- 20 54 D2 JSR D254 JSR D499/ROM convertit le nombre signé AY en nombre réel dans ACC1

Copie ACC1 dans la variable BASIC

F620- A5 29 LDA 29 lit le flag "entier"
F622- 4C FE D1 JMP D1FE JSR CB39/ROM affecter un nombre à une variable

Copie une chaîne dans la variable BASIC

F625- A5 F5 LDA F5 longueur de la chaîne
F627- 20 64 D2 JSR D264 JSR D5AB/ROM réserve une place en mémoire pour la chaîne de longueur A, retourne avec cette longueur en D0 et l'adresse de la chaîne en D1/D2
F62A- A8 TAY longueur de la chaîne
F62B- F0 08 BEQ F635 bon, si la chaîne est vide, rien à copier!
F62D- 88 DEY indexe les octets à copier
F62E- B1 02 LDA (02),Y lit un octet dans le "Channel's own Data Buffer"
F630- 91 D1 STA (D1),Y et le copie dans la zone réservée en mémoire
F632- 98 TYA teste Y
F633- D0 F8 BNE F62D et reboucle s'il en reste à copier
F635- 4C 8E EE JMP EE8E XCSTR copie la longueur et l'adresse d'une chaîne alphanumérique (les 3 octets D0, D1 et D2) "dans" la variable BASIC pointée en B8, B9 et BA

Série de NOP en attente

F638- EA NOP Le sous-programme que Ray avait implanté ici (STX 3115 et JMP E635)
F639- EA NOP a été déplacé dans la BANQUE n°6 (de C513 à C518, afin de libérer
F63A- EA NOP un maximum de place dans le NOYAU. La version 3.0 retrouve donc
F63B- EA NOP la série de 8 NOPs qui était présente dans la version 1.006 et qui sera
F63C- EA NOP précieuse pour de futures implémentations.

F63D- EA NOP
F63E- EA NOP
F63F- EA NOP

Vide le "General Field Buffer", puis décode le nom de champ et s'il s'agit d'un pseudo-tableau, l'index de cet élément (ce sous-programme est aussi appelé par les commandes FIELD, LSET et RSET)

F640- A2 0A LDX #0A pour forcer 10 octets à zéro
F642- A9 00 LDA #00
F644- 9D 75 C0 STA C075,X force à zéro les octets de C076 à C07F
F647- CA DEX c'est à dire le "General field Buffer"
F648- D0 FA BNE F644 reboucle en F644 tant qu'il en reste
F64A- A5 0A LDA 0A lit le NL courant
F64C- 8D 7C C0 STA C07C et le copie en C07C
F64F- 20 9E D3 JSR D39E XCRGOT relit le caractère à TXTPTR (sans incrémenter TXTPTR = CHRGOT), puis le convertit en MAJUSCULE, les espaces sont sautés, Z = 1 si fin d'instruction (0 ou :), C = 0 si caractère chiffre 0 à 9 (soit #30 à #39), sinon C = 1, Y et X inchangés
F652- 4C 58 F6 JMP F658 saute l'instruction suivante (X = 0 cf BNE en F648)

Copie le nom de champ dans le "General Field Buffer"

F655- 20 98 D3 JSR D398 XCRGET incrémente TXTPTR, lit un caractère (CHRGET), les espaces sont sautés, le met dans A, le convertit en MAJUSCULE, Z = 1 si fin d'instruction (0 ou :), C = 0 si chiffre 0 à 9 (soit #30 à #39), sinon C =1. Y et X inchangés
F658- F0 72 BEQ F6CC simple RTS en F6CC (!) si fin de commande atteinte
F65A- C9 80 CMP #80 teste s'il est >= à #80 (token BASIC)

NB: les mots-clés BASIC étant interdits dans les noms de champ, dès qu'un token BASIC est rencontré (codé par #D3), le sous-programme suppose qu'il s'agit d'un ">" et la sortie se fait par un RTS en F6CC (et ben quoi!?). Si une "(" marquant un index d'élément de pseudo-tableau est rencontré, la sortie se fait dans le sous-programme F66C avec le JMP D22E

F65C- B0 6E BCS F6CC si oui, simple RTS en F6CC
F65E- C9 28 CMP #28 teste si c'est une "(" (élément de pseudo-tableau)
F660- F0 0A BEQ F66C si oui, continue en F66C
F662- E0 05 CPX #05 teste si X = 5 (5 caractères maximum pour un nom de champ)
F664- F0 EF BEQ F655 si oui, reprend en F655 (saute les caractères restants, qui ne sont pas significatifs)
F666- 9D 76 C0 STA C076,X sinon, copie les 5 premiers caractères du nom de
F669- E8 INX champ dans le "General Field Buffer"
F66A- D0 E9 BNE F655 reprise forcée en F655 ("<" étant codé par le token #D3, la sortie se fait par F6CC, sauf si c'est un élément de pseudo-tableau:)

Décode l'index d'un élément de pseudo-tableau

Rappel: SEDORIC accepte comme nom de champ un élément de pseudo-tableau (index de 0 à 255) exemple ROBERT(4) qui en fait correspond à ROBER(4).

F66C- 20 98 D3 JSR D398 XCRGET incrémente TXTPTR, lit un caractère (CHRGET), les espaces sont sautés, le met dans A, le convertit en MAJUSCULE, Z = 1 si fin d'instruction (0 ou :), C = 0 si chiffre 0 à 9 (soit #30 à #39), sinon C =1. Y et X inchangés
F66F- 20 7F D2 JSR D27F CF17/ROM et CF09/ROM et D8CB/ROM évalue un nombre entier à TXTPTR et le retourne dans X (index de l'élément du tableau)
F672- 8E 7B C0 STX C07B sauve X dans C07B (index de l'élément du tableau)
F675- A9 29 LDA #29 code ASCII de ")"
F677- 4C 2E D2 JMP D22E JSR D067/ROM puis D3A1/RAM overlay demande une ")" à TXTPTR, lit le caractère suivant, le convertit éventuellement en MAJUSCULE et retourne

Compare les types d'enregistrement et de variables
(ce sous-programme est aussi appelé par les commandes LSET et RSET)

F67A- AD 7F C0 LDA C07F lit l'octet en C07F (type d'enregistrement) (ce sous-programme est aussi appelé par les commandes PUT et TAKE)
F67D- 8D 7F C0 STA C07F écrit l'octet en C07F (type d'enregistrement)
F680- 0A ASL C reçoit le b7 de C07F (flag pour tester le type)
F681- 4C 1C D2 JMP D21C JSR CF09/ROM vérifie si l'expression évaluée à TXTPTR est bien conforme: numérique si C = 0 ou alphanumérique si C = 1, retourne avec valeur numérique dans ACC1 ou adresse de la chaîne dans D3/D4 ou avec une "TYPE_MISMATCH_ERROR" si les types sont incompatibles

Calcule la position de début de la fiche spécifiée dans un fichier "**R**" (accès direct), puis les coordonnées du secteur où commence cette fiche, charge ce secteur et le suivant dans le "General Buffer", place le pointeur 06/07 au début réel de la fiche dans ce buffer et retourne avec Y = #00

(sous-programme F684-F88D, appelé par les commandes PUT et TAKE)

Sachant que les fiches peuvent avoir une longueur comprise entre #03 et #FF (non documenté) et qu'elles sont placées les unes à la suite des autres dans le fichier, elles tombent la plupart du temps à cheval sur 2 secteurs. Ce sous-programme charge dans le "General Buffer" les 2 secteurs contenant la fiche dont le n° est indiqué en 33/34.

F684- A9 00 LDA #00 force F2 à zéro, ce sera l'octet de poids le plus
F686- 85 F2 STA F2 fort du "n° de fiche" codé par 33/34/F2 lors de la multiplication qui suit, à savoir, **nombre d'octets précédant la fiche (codé par C085/08/09) = n° de la fiche (codé par 33/34/F2) que multiplie longueur de fiche (codée par F3)**. Cette opération est effectuée de façon classique, par additions successives dans le totalisateur C085/08/09, pour chaque bit du multiplicateur F3 qui est à 1, d'une quantité croissante 33/34/F2 égale à 1 fois, 2 fois, 4 fois, 8 fois etc... la valeur initiale du n° de fiche (multiplicande). A la fin C085, 08 et 09 donneront la position du début de la fiche dans le fichier. C085, étant l'octet de poids le plus faible, correspondra au rang de l'octet de début de la fiche dans le secteur de rang 08/09 depuis le début du fichier.
F688- 8D 85 C0 STA C085 force C085 à zéro (ce sera le rang de l'octet)
F68B- 85 08 STA 08
F68D- 85 09 STA 09 force 08/09 à zéro (vise le secteur de rang n° #00)
F68F- AD 83 C0 LDA C083 lit la longueur de fiche LF
F692- A2 08 LDX #08 pour 8 rebouclages (nombre de bits de LF à traiter)
F694- 85 F3 STA F3 c'est le multiplicateur LF

F696- 46 F3 LSR F3 teste si le b0 de cet octet est nul
F698- 90 15 BCC F6AF si oui, continue en F6AF, il n'y a pas d'addition à effectuer pour ce bit qui est nul
F69A- 18 CLC sinon, prépare l'addition:
F69B- A5 33 LDA 33 C085/08/09 = C085/08/09 + 33/34/F2
F69D- 6D 85 C0 ADC C085
F6A0- 8D 85 C0 STA C085 C085 = C085 + 33
F6A3- A5 34 LDA 34
F6A5- 65 08 ADC 08
F6A7- 85 08 STA 08 08 = 08 + 34 + retenue précédente
F6A9- A5 F2 LDA F2
F6AB- 65 09 ADC 09
F6AD- 85 09 STA 09 09 = 09 + F2 + retenue précédente
F6AF- 06 33 ASL 33 calcule la valeur de la tranche suivante 33/34/F2
F6B1- 26 34 ROL 34 qu'il faudra ajouter au totalisateur si le bit
F6B3- 26 F2 ROL F2 correspondant du multiplicateur est à 1. Ceci est réalisé par un décalage à gauche portant sur les trois octets 33/34/F2.
F6B5- CA DEX décrémente le nombre de bit restant à traiter
F6B6- D0 DE BNE F696 reboucle tant que X n'est pas nul
F6B8- 20 CD F6 JSR F6CD charge 2 secteurs du fichier de la disquette dans le "General Buffer". Le premier secteur contient le début de la fiche spécifiée. Lorsque le sous-programme est utilisé par la commande PUT, il alloue, si nécessaire, des secteurs supplémentaires sur la disquette à ce fichier de type "**R**".
F6BB- 20 A8 F4 JSR F4A8 place l'adresse du début du "Channel Buffer" correspondant au NL en 00/01, celle du début du "Channel's own Data Buffer" en 02/03, celle du "Descriptor Buffer" en 04/05, celle du début du "General Buffer" en 06/07 et enfin, met à jour C083 (longueur d'une fiche) et 0B (flag "**R**") puis retourne avec Y = #00
F6BE- 18 CLC prépare une addition
F6BF- AD 85 C0 LDA C085
F6C2- 65 06 ADC 06 calcule l'adresse 06/07 du début de la fiche
F6C4- 85 06 STA 06 dans le "General Buffer"
F6C6- 90 02 BCC F6CA
F6C8- E6 07 INC 07 06/07 = 06/07 + C085
F6CA- A0 00 LDY #00
F6CC- 60 RTS et retourne avec Y = #00

Charge 2 secteurs du fichier de la disquette dans le "General Buffer", alloue des secteurs supplémentaires à ce fichier de type "**R**" si nécessaire

F6CD- 20 A8 F4 JSR F4A8 place l'adresse du début du "Channel Buffer" correspondant au NL en 00/01, celle du début du "Channel's own Data Buffer" en 02/03, celle du "Descriptor Buffer" en 04/05, celle du début du "General Buffer" en 06/07 et enfin, met à jour C083 (longueur d'une fiche) et 0B (flag "**R**") puis retourne avec Y = #00
F6D0- 18 CLC prépare une addition
F6D1- A5 08 LDA 08 AX, rang du secteur de la fiche
F6D3- A6 09 LDX 09 AX contient le nombre absolu, en partant de zéro, de secteurs du fichier, nécessaires pour stocker ce nombre de fiches
F6D5- 69 02 ADC #02 AX = AX + #02, ajoute #02 pour normaliser, car

F6D7- 90 01 BCC F6DA un nombre de secteurs de #00 nécessite en fait un
F6D9- E8 INX secteur, plus un autre, car une fiche peut être à cheval sur le secteur suivant. AX contient maintenant la taille minimale (le nombre de secteurs) requise du fichier pour stocker toutes les fiches.

Compare le nombre de secteurs de la fiche avec le nombre total de secteurs de data dans le fichier

F6DA- A0 0A LDY #0A vise le LL du nombre de secteurs de data du fichier
F6DC- 38 SEC prépare la soustraction:
F6DD- F1 04 SBC (04),Y A = A - LL du nombre actuel
F6DF- 48 PHA LL, nombre de secteurs supplémentaires nécessaires
F6E0- C8 INY vise HH du nombre de secteurs de data du fichier
F6E1- 8A TXA pour calculer A = A - HH du nombre actuel
F6E2- F1 04 SBC (04),Y finalement AY = nombre requis - nombre actuel
F6E4- A8 TAY
F6E5- 68 PLA AY, nombre de secteurs supplémentaires nécessaires
F6E6- 90 03 BCC F6EB saute l'instruction suivante si le nombre de secteurs actuel est suffisant pour stocker toutes les fiches dans le fichier
F6E8- 20 5A F7 JSR F75A alloue AY secteurs supplémentaires à ce fichier. Cette allocation complémentaire n'est évidemment exécutée que pour la commande PUT. En effet, si la fiche n'existe pas encore, une "BAD_RECORD_NUMBER_ERROR" est générée au début de la commande TAKE. Conclusion, lors d'une commande TAKE, on perd beaucoup de temps inutilement avec ce sous-programme!

Prépare le calcule du nombre de descripteurs nécessaires

F6EB- A2 FF LDX #FF initialise le compteur de descripteurs
F6ED- 18 CLC prépare une addition
F6EE- A5 08 LDA 08 08/09, rang du secteur depuis le début du fichier,
F6F0- 69 05 ADC #05 c'est à dire, nombre absolu (partant de zéro) de
F6F2- 85 08 STA 08 secteurs de data. Dans le ou les descripteur(s),
F6F4- 90 02 BCC F6F8 chaque secteur de data est repéré par ses
F6F6- E6 09 INC 09 coordonnées piste/secteur (2 octets). Il y a donc autant de paires de coordonnées que de secteurs de data dans le fichier. La liste de ces coordonnées commence en général à l'octet de rang #02 de chaque descripteur (les 2 premiers octets donnent les coordonnées du descripteur suivant), sauf pour le premier descripteur où elle ne commence qu'à l'octet de rang #0C (outre les 2 premiers, 10 autres octets sont utilisés pour d'autres informations). Le sous-programme doit calculer le nombre de descripteurs qu'il faudra pour copier la liste des coordonnées de tous les secteurs du fichier. Afin de simplifier ce calcul, le sous-programme augmente artificiellement le nombre de secteurs de data de #05 (équivalant de 5 paires d'octets piste/secteur), pour compenser les 10 octets déjà requis en plus dans le premier descripteur. Tout se passera alors comme s'il était possible d'écrire 254 / 2 = 127 (#7F) coordonnées par descripteur.

Calcule le nombre de descripteurs nécessaires pour copier la liste des coordonnées de tous les secteurs du fichier

F6F8- 38 SEC prépare une soustraction
F6F9- A5 08 LDA 08 le compteur 08/09 sera décrémenté de #7F

F6FB- A8 TAY Y gardera le reste de la dernière soustraction (nombre de secteurs qui sont répertoriés dans le dernier descripteur)
F6FC- E9 7F SBC #7F nombre de coordonnées stockables dans un descripteur
F6FE- 85 08 STA 08 08/09 = 08/09 - #7F
F700- A5 09 LDA 09
F702- E9 00 SBC #00 répercussion de la retenue
F704- 85 09 STA 09
F706- E8 INX compteur des descripteurs nécessaires, qui passe à zéro au premier tour, c'est à dire pour le premier descripteur. C'est donc en fait un compteur du nombre des descripteurs secondaires nécessaires.
F707- B0 F0 BCS F6F9 reboucle en F6F9 si le reste est supérieur à #7F
F709- C8 INY nombre réel de secteurs répertoriés dans le dernier
F70A- 98 TYA (l'ancien nombre était un rang commençant à zéro)
F70B- 0A ASL le multiplie par 2 (nombre d'octets requis)
F70C- 8D 84 C0 STA C084 pointeur dans le dernier descripteur
F70F- 85 F8 STA F8 idem
F711- 8A TXA
F712- 48 PHA nombre des descripteurs secondaires nécessaires
F713- 20 A8 F4 JSR F4A8 place l'adresse du début du "Channel Buffer" correspondant au NL en 00/01, celle du début du "Channel's own Data Buffer" en 02/03, celle du "Descriptor Buffer" en 04/05, celle du début du "General Buffer" en 06/07 et enfin, met à jour C083 (longueur d'une fiche) et 0B (flag "**R**") puis retourne avec Y = #00
F716- 68 PLA récupère le rang R du dernier descripteur
F717- 18 CLC prépare une addition
F718- 65 05 ADC 05 04/05 adresse du début du "Descriptor Buffer"
F71A- 85 05 STA 05 en augmente le HH de R pages: 04/05 vise maintenant le dernier descripteur où est décrit le secteur de data contenant la fiche
F71C- 85 F7 STA F7 sauve également le résultat dans F7
F71E- AC 84 C0 LDY C084 pointeur dans le dernier descripteur
F721- 20 36 F7 JSR F736 lit le secteur de data de la disquette et l'écrit dans le buffer indiqué en 06/07, c'est à dire la première page du "General Buffer" de **FI** et incrémente le HH de ce pointeur, c'est à dire 07
F724- 4C 36 F7 JMP F736 idem pour le secteur suivant dans la deuxième page. Bien que la taille d'une fiche ne puisse dépasser 256 octets, deux secteurs doivent être lus, car une fiche peut commencer dans un secteur et continuer dans le secteur suivant. Ces deux secteurs existent toujours dans le fichier, car quand le fichier est ouvert pour la première fois, la routine PUT est utilisée pour fixer le nombre de secteurs initiaux et ceci assure que, même si la dernière fiche ne se trouve pas à cheval sur un deuxième secteur, ce secteur supplémentaire soit déjà alloué au fichier. Quand un fichier est étendu pour recevoir des fiches supplémentaires (commande PUT), ce secteur supplémentaire est de la sorte automatiquement alloué au fichier.

Ecrit sur la disquette les 2 secteurs de data contenant la fiche spécifiée

F727- 20 A8 F4 JSR F4A8 place l'adresse du début du "Channel Buffer" correspondant au NL en 00/01, celle du début du "Channel's own Data Buffer" en 02/03, celle du "Descriptor Buffer" en 04/05, celle du début du "General Buffer" en 06/07 et enfin, met à jour C083 (longueur d'une fiche) et 0B (flag "**R**") puis retourne avec Y = #00
F72A- A5 F7 LDA F7 HH de l'adresse du descripteur courant remis en

F72C- 85 05 STA 05 place (04/05, adresse du début du descripteur courant)
F72E- A4 F8 LDY F8 pointeur dans le descripteur courant
F730- 20 33 F7 JSR F733 écrit sur la disquette un secteur de data de la première page du "General Buffer" de **FI** pointé par 06/07 et incrémente 07 pour viser la deuxième page. Continue à la routine suivante qui fait de même pour la deuxième page du "General Buffer".

Ecrit sur la disquette, selon les coordonnées présentes au pointeur Y du descripteur indiqué en 04/05, un secteur de data pointé par 06/07, incrémente 07 pour viser la page suivante, ajuste Y et 04/05 pour viser les coordonnées du secteur suivant

F733- A2 A8 LDX #A8 commande "écriture" pour routine XRWTS
F735- 2C A2 88 BIT 88A2 continue en F738

Lit sur la disquette, selon les coordonnées présentes au pointeur Y du descripteur indiqué en 04/05, un secteur de data, le copie au pointeur 06/07 et incrémente 07 pour viser la page suivante, ajuste Y et 04/05 pour viser les coordonnées du secteur suivant

F736- A2 88 LDX #88 commande "lecture" pour routine XRWTS
F738- B1 04 LDA (04),Y lit le n° de piste
F73A- 8D 01 C0 STA C001 et le copie dans PISTE
F73D- C8 INY
F73E- B1 04 LDA (04),Y lit le n° de secteur
F740- 8D 02 C0 STA C002 et le copie dans SECTEUR
F743- A5 06 LDA 06
F745- 8D 03 C0 STA C003
F748- A5 07 LDA 07
F74A- 8D 04 C0 STA C004 met à jour RWBUF avec 06/07
F74D- E6 07 INC 07 prépare HH suivant (page suivante du "General
F74F- C8 INY Buffer") teste si fin du descripteur atteinte
F750- D0 04 BNE F756 sinon, saute les 2 instructions suivantes
F752- E6 05 INC 05 si oui, incrémente HH (descripteur suivant)
F754- A0 02 LDY #02 et force Y = #02 (vise les premières coordonnées)
F756- 4C 75 DA JMP DA75 suite à la routine XRWTS (gestion drive)

F759- 60 RTS

Alloue des secteurs supplémentaires a un fichier de type "**R**" ou "**S**"
Recherche le descripteur courant (dernier descripteur)

F75A- 8D 58 C0 STA C058 AY, nombre de secteurs supplémentaires requis
F75D- 8C 59 C0 STY C059 sauve AY en C058/C059
F760- 0D 59 C0 ORA C059 teste si le contenu de AY était nul
F763- F0 F4 BEQ F759 si oui, simple RTS en F759
F765- 20 4C DA JSR DA4C sinon, XPMAP prend le premier secteur de bitmap dans BUF2, vérifie le format, met à zéro le b7 de 2F (flag"première bitmap chargée").
F768- 20 A8 F4 JSR F4A8 place l'adresse du début du "Channel Buffer" correspondant au NL en 00/01, celle du début du "Channel's own Data Buffer" en 02/03, celle du "Descriptor Buffer" en 04/05, celle du début du "General Buffer" en 06/07 et enfin, met à jour C083 (longueur d'une fiche ou #00)

et 0B (flag "**S**/**R**") puis retourne avec Y = #00

F76B- A0 02 LDY #02 pour viser le n° du descripteur courant
F76D- B1 00 LDA (00),Y teste si ce n° est nul (cas du premier descripteur)
F76F- F0 14 BEQ F785 si oui, continue en F785

Cas d'un descripteur secondaire

F771- 18 CLC sinon, prépare une addition
F772- 65 05 ADC 05 pour mettre à jour 04/05
F774- 85 05 STA 05 05 = 05 + n° du descripteur courant - #01
F776- C6 05 DEC 05 afin de viser le descripteur précédent qui porte les coordonnées du descripteur recherché
F778- A0 00 LDY #00
F77A- B1 04 LDA (04),Y n° de piste du descripteur recherché
F77C- AA TAX
F77D- C8 INY
F77E- B1 04 LDA (04),Y n° de secteur du descripteur recherché
F780- C8 INY pointe sur les premières coordonnées du descripteur
F781- E6 05 INC 05 04/05 vise le descripteur recherché
F783- D0 0A BNE F78F suite forcée en F78F (05 HH jamais nul)

Cas du premier descripteur

F785- A0 13 LDY #13 vise les coordonnées du descripteur principal dans la ligne de catalogue recopiée dans le "Channel Buffer"
F787- B1 00 LDA (00),Y n° de piste du premier descripteur
F789- AA TAX sauvé dans X
F78A- C8 INY octet suivant
F78B- B1 00 LDA (00),Y n° de secteur du premier descripteur
F78D- A0 0C LDY #0C pointe sur les premières coordonnées du premier descripteur

Actualise PISTE, SECTEUR et RWBUF

F78F- 8E 01 C0 STX C001 PISTE
F792- 8D 02 C0 STA C002 SECTEUR
F795- 20 5F F8 JSR F85F copie dans RWBUF l'adresse présente en 04/05

Recherche la fin de la liste des coordonnées dans le descripteur courant

F798- C8 INY vise l'octet suivant (n° de secteur)
F799- B1 04 LDA (04),Y lit le n° de secteur dans la liste des descripteurs
F79B- F0 05 BEQ F7A2 continue en F7A2 si nul (fin de la liste: il reste au moins une paire d'octets disponible)
F79D- C8 INY vise l'octet suivant (n° de piste)
F79E- D0 F8 BNE F798 reboucle en F798 si la fin du descripteur n'est pas atteinte, jusqu'à trouver la fin de cette liste
F7A0- F0 34 BEQ F7D6 continue en F7D6 lorsque la fin du descripteur est atteinte: il faut allouer un autre secteur pour élaborer un nouveau descripteur

Il y a encore de la place dans ce descripteur:
ajoute les coordonnées du ou des secteurs supplémentaires requis

F7A2- 88 DEY vise l'octet précédent (c'est à dire le n° de piste du dernier secteur du fichier)
F7A3- AD 58 C0 LDA C058 teste si le nombre de secteurs supplémentaires
F7A6- 0D 59 C0 ORA C059 requis est nul
F7A9- F0 57 BEQ F802 si oui, continue en F802 (l'insertion des coordonnées des nouveaux secteurs supplémentaires est terminée)
F7AB- 20 5F F8 JSR F85F sinon, copie dans RWBUF
F7AE- AD 58 C0 LDA C058 l'adresse du descripteur courant et
F7B1- D0 03 BNE F7B6
F7B3- CE 59 C0 DEC C059 décrémente le nombre de secteurs supplémentaires
F7B6- CE 58 C0 DEC C058 à allouer
F7B9- 8C 5F C0 STY C05F index du dernier secteur alloué dans le descripteur
F7BC- 20 38 F8 JSR F838 incrémente le nombre de secteurs de data, puis le nombre de secteurs totaux et enfin cherche un secteur libre AY sur la bitmap
F7BF- 84 F2 STY F2 sauve l'indication de secteur libre dans F2
F7C1- AC 5F C0 LDY C05F récupère l'index dans le descripteur
F7C4- 91 04 STA (04),Y sauve l'indication de piste à la fin de la liste
F7C6- C8 INY de coordonnées dans le descripteur
F7C7- A5 F2 LDA F2 récupère l'indication de secteur
F7C9- 91 04 STA (04),Y sauve l'indication de secteur à la suite
F7CB- C8 INY position suivante dans le descripteur
F7CC- D0 D5 BNE F7A3 reboucle en F7A3 tant que le descripteur n'est pas fini

Génère un autre descripteur, car le précédant est plein

F7CE- AD 58 C0 LDA C058 teste si le nombre de secteurs supplémentaires
F7D1- 0D 59 C0 ORA C059 requis est nul
F7D4- F0 2C BEQ F802 si oui, continue en F802, réflexion faite il n'y a plus de secteurs à allouer!

Alloue un autre secteur pour le descripteur suivant

F7D6- 20 4C F8 JSR F84C incrémente le nombre de secteurs totaux du fichier et cherche un secteur libre AY sur la bitmap
F7D9- 85 F5 STA F5
F7DB- 84 F6 STY F6 sauve les coordonnées piste/secteur en F5/F6
F7DD- A0 00 LDY #00
F7DF- 91 04 STA (04),Y copie l'indication de piste du nouveau descripteur au début du descripteur courant (lien
F7E1- C8 INY indiquant les coordonnées du descripteur suivant) récupère
F7E2- A5 F6 LDA F6 l'indication de secteur du nouveau descripteur
F7E4- 91 04 STA (04),Y sauve l'indication de secteur à la suite
F7E6- 20 A4 DA JSR DAA4 XSVSEC écrit le descripteur courant qui était plein sur la disquette, selon DRIVE, PISTE, SECTEUR et RWBUF
F7E9- A5 F5 LDA F5

F7EB- A4 F6 LDY F6 récupère les coordonnées piste/secteur
F7ED- 8D 01 C0 STA C001 et copie dans PISTE
F7F0- 8C 02 C0 STY C002 et dans SECTEUR pour le prochain descripteur
F7F3- 20 6A F8 JSR F86A génère un autre descripteur dans le "Descriptor Buffer": déplace d'une page vers le haut le pointeur de descripteur 04/05, incrémente l'octet de rang #02 du "Channel Buffer" (nombre de descripteurs), calcule l'offset du point d'insertion 04/05 et augmente le "Channel Buffer" de #100 octets
F7F6- A9 00 LDA #00
F7F8- A8 TAY
F7F9- 91 04 STA (04),Y forcer à zéro toute la page
F7FB- C8 INY de ce nouveau descripteur
F7FC- D0 FB BNE F7F9
F7FE- A0 02 LDY #02 index des premières coordonnées dans le nouveau descripteur
F800- D0 A1 BNE F7A3 suite forcée en F7A3 avec Y = #02 (reprise de l'insertion des coordonnées des nouveaux secteurs supplémentaires)

Ecrit le dernier descripteur sur la disquette,

F802- 20 A4 DA JSR DAA4 XSVSEC écrit la dernière page de descripteur sur la disquette, selon DRIVE, PISTE, SECTEUR et RWBUF

Met à jour, sur la disquette, "l'entrée" de catalogue et le descripteur principal

Ce sous-programme, qui ne concerne que les fichiers de type "**R**" et "**S**", permet de répercuter sur la disquette un changement possible de la taille du fichier.

F805- A0 06 LDY #06 pour copier 17 octets (de Y = #06 à #16) qui incluront le n° du drive, 9 octets de nom, 3 octets d'extension, PSDESP coordonnées du descripteur principal et enfin NSTOTP nombre de secteurs totaux + PROT/UNPROT (soit n° drive + une "entrée" de catalogue)
F807- B1 00 LDA (00),Y lit un octet selon l'adresse en 00/01 + Y
F809- 99 22 C0 STA C022,Y et le copie dans BUFNOM
F80C- C8 INY
F80D- C0 17 CPY #17
F80F- D0 F6 BNE F807 reboucle en F807 tant qu'il en reste à copier
F811- 20 30 DB JSR DB30 XTVNM cherche le fichier BUFNOM revient avec POSNMX POSNMP et POSNMS ou Z = 1 si rien trouvé
F814- D0 03 BNE F819 si trouvé, saute l'instruction suivante
F816- 4C DD E0 JMP E0DD sinon, "FILE_NOT_FOUND_ERROR"

F819- 20 EE DA JSR DAEE XBUCA transfère BUFNOM dans BUF3, à la position POSNMX (pour mise à jour de "l'entrée" de catalogue sur la disquette)
F81C- 20 8A DA JSR DA8A l'entrée de cette routine XSMAP sauve BUF2 (bitmap) sur la disquette a été déportée en DC80 pour adaptation à la double bitmap. Un JSR DC80 plus direct aurait fait gagner un peu de temps.
F81F- 20 82 DA JSR DA82 XSCAT sauve BUF3 selon POSNMP et POSNMS
F822- 20 A8 F4 JSR F4A8 place l'adresse du début du "Channel Buffer" correspondant au NL en 00/01, celle du début du "Channel's own Data Buffer" en 02/03, celle du "Descriptor Buffer" en 04/05, celle du début du "General Buffer" en 06/07 et enfin, met à jour C083 (longueur d'une fiche ou #00)

et 0B (flag "**S**/**R**") puis retourne avec Y = #00

F825- A0 13 LDY #13
F827- B1 00 LDA (00),Y lit le n° de piste du descripteur principal
F829- 8D 01 C0 STA C001 et le copie dans PISTE
F82C- C8 INY
F82D- B1 00 LDA (00),Y lit le n° de secteur correspondant
F82F- 8D 02 C0 STA C002 et le copie dans SECTEUR
F832- 20 5F F8 JSR F85F copie dans RWBUF l'adresse du "Descriptor Buffer"
F835- 4C A4 DA JMP DAA4 XSVSEC écrit le premier descripteur sur la disquette, selon DRIVE, PISTE, SECTEUR et RWBUF

Incrémente le nombre de secteurs de data, puis incrémente le nombre de secteurs totaux et enfin cherche un secteur libre AY sur la bitmap

F838- A0 0A LDY #0A pour viser l'octet #0A du descripteur principal
F83A- E6 03 INC 03 incrémente HH de l'adresse présente en 02/03

NB: le descripteur principal est situé #100 octets après le début du "Channel's own Data Buffer"

F83C- B1 02 LDA (02),Y lit LL du nombre de secteurs de data
F83E- 18 CLC prépare une addition
F83F- 69 01 ADC #01 y ajoute #01
F841- 91 02 STA (02),Y et le remet en place
F843- C8 INY vise l'octet suivant
F844- B1 02 LDA (02),Y lit HH du nombre de secteurs de data
F846- 69 00 ADC #00 reporte la retenue précédente
F848- 91 02 STA (02),Y et le remet en place
F84A- C6 03 DEC 03 HH reprend sa valeur d'origine

Incrémente le nombre de secteurs totaux et enfin cherche un secteur libre AY sur la bitmap

F84C- A0 15 LDY #15 pour viser l'octet de rang #15 du "Channel Buffer"
F84E- B1 00 LDA (00),Y lit LL du nombre de secteurs totaux dans la ligne de catalogue
F850- 18 CLC prépare une addition
F851- 69 01 ADC #01 y ajoute #01
F853- 91 00 STA (00),Y et le remet en place
F855- C8 INY vise l'octet suivant
F856- B1 00 LDA (00),Y lit HH du nombre de secteurs totaux
F858- 69 00 ADC #00 reporte la retenue précédente
F85A- 91 00 STA (00),Y et le remet en place
F85C- 4C 6C DC JMP DC6C XLIBSE cherche un secteur libre dans la bitmap dans BUF2, retourne avec A = n° de piste et Y = n° de secteur (sinon "DISK_FULL_ERROR")

Copie dans RWBUF l'adresse présente en 04/05
(sous-programme F85F-F869)

F85F- A5 04 LDA 04

```
F861-    8D 03 C0    STA C003    LL de RWBUF
F864-    A5 05       LDA 05
F866-    8D 04 C0    STA C004    HH de RWBUF
F869-    60          RTS
```

Déplace d'une page vers le haut le pointeur de descripteur 04/05, incrémente l'octet de rang #02 du "Channel Buffer", (nombre de descripteurs), calcule l'offset du point d'insertion 04/05 et augmente le "Channel Buffer" de #100 octets

F86A- E6 05 INC 05 incrémente 05 (HH de l'adresse du descripteur, qui commence par les coordonnées PISTE et SECTEUR du descripteur suivant s'il existe ou par #00). 04/05 vise le dernier descripteur du "Descriptor Buffer"

```
F86C-    18          CLC         prépare une addition
F86D-    A0 02       LDY #02     lit l'octet de rang 02 du "Channel
F86F-    B1 00       LDA (00),Y  Buffer", y ajoute #01
F871-    69 01       ADC #01     et le remet en place (compteur du nombre de
F873-    91 00       STA (00),Y  descripteurs, incrémenté à chaque appel du sous-programme)
F875-    A5 04       LDA 04      AY est l'adresse de début du nouveau
F877-    A4 05       LDY 05      descripteur
F879-    20 85 F8    JSR F885    calcule l'offset XY du pointeur d'adresse AY (nombre d'octets
```

entre le début de **FI** et le point où sera créé le prochain descripteur, c'est à dire à la fin du "Descriptor Buffer")

```
F87C-    A9 01       LDA #01
F87E-    85 F2       STA F2      AF2 = #0100
F880-    A9 00       LDA #00     (nombre d'octets à insérer à l'adresse 04/05)
F882-    4C 25 F4    JMP F425    extension de FI par insertion de #100 octets au point d'offset
```

XY, avec mise à jour de la "Table NL"

Calcule l'offset XY du pointeur AY par rapport au début de **FI**

```
F885-    38          SEC         prépare une soustraction
F886-    E5 9E       SBC 9E
F888-    AA          TAX
F889-    98          TYA
F88A-    E5 9F       SBC 9F      calcule XY = AY - 9E/9F
F88C-    A8          TAY
F88D-    60          RTS
```

EXÉCUTION DE LA COMMANDE SEDORIC &()

(Fichiers "**S**" et "**R**")
(sous-programme F88E-F8DE)

Rappel de la syntaxe

& (NL) et & (-NL)

Attention, les parenthèses n'indiquent pas une option facultative, mais sont obligatoires. Cette commande

retourne les informations suivantes qui diffèrent selon le signe du paramètre et le type de fichier:

Informations non ou mal documentées

Pour les fichiers de type "**D**", cette commande n'est pas utilisable.

Pour les fichiers de type "**R**", cette commande retourne le nombre de fiches si &(+n°) ou la longueur de fiche si &(-n°).

Pour les fichiers de type "**S**", cette commande retourne -1 (vrai) dans tous les cas (±n°) si la fin du fichier est atteinte et si ce n'est pas le cas, retourne soit 0 (false) si &(+n°) soit le type d'enregistrement si &(-n°). C'est le contraire de ce qui est indiqué dans le manuel, page 81 (bogue du manuel).

Variables utilisées

En voir la liste et les fonctions dans les généralités sur la gestion de fichiers de data situées devant le sous-programme F3CF.

Analyse de syntaxe et saisie du paramètre

Le paramètre entre parenthèses est décodé par la ROM du BASIC, et placé en virgule flottante dans ACC1 (D0/D4). Ce sous-programme est appelé à partir du vecteur 0461. SEDORIC vérifie que le paramètre se situe bien entre -63 et +63, puisqu'il s'agit d'un NL (qui en plus doit être attribué, sinon "FILE_NOT_OPEN_ERROR"). Si le NL est attribué à un fichier de type "**D**", il en résulte une "FILE_TYPE_MISMATCH_ERROR".

Convertit le nombre entier signé en un octet non signé (NL)

F88E-	20 4C D2	JSR D24C	JSR D2A9/ROM nombre en ACC1 -> #D4-#D3 (signé)
F891-	A5 D4	LDA D4	
F893-	A6 D3	LDX D3	A = LL et X = HH
F895-	10 0C	BPL F8A3	continue en F8A3 si c'est un nombre entier positif

C'est un nombre entier négatif

F897-	49 FF	EOR #FF	inverse tous les bits du LL et ajoute 1
F899-	18	CLC	(calcule le complément à 2 du LL,
F89A-	69 01	ADC #01	c'est à dire le NL du fichier à tester)
F89C-	E0 FF	CPX #FF	teste si HH est bien égal à #FF (D3 garde le signe)
F89E-	F0 07	BEQ F8A7	si oui, tout va bien, continue en F8A7
F8A0-	4C 20 DE	JMP DE20	sinon "ILLEGAL_QUANTITY_ERROR"

C'est un nombre entier positif

F8A3-	E0 00	CPX #00	teste si HH est bien égal à #00 (D3 garde le signe)
F8A5-	D0 F9	BNE F8A0	si ce n'est pas le cas, "ILLEGAL_QUANTITY_ERROR"

Ce nombre (NL présent dans A) est un octet signé

F8A7- 20 73 F4 JSR F473 vérifie l'existence de **FI**, la validité du NL présent dans A et si le fichier est bien déjà ouvert
F8AA- 20 A8 F4 JSR F4A8 place l'adresse du début du "Channel Buffer" correspondant au NL en 00/01, celle du début du "Channel's own Data Buffer" en 02/03, celle du "Descriptor Buffer" en 04/05, celle du début du "General Buffer" en 06/07 et enfin, met à jour C083 (longueur d'une fiche ou #00) et 0B (flag "**S**/**R**/**D**") puis retourne avec Y = #00
F8AD- 30 23 BMI F8D2 si le fichier est de type "**S**"
F8AF- D0 1E BNE F8CF "FILE_TYPE_MISMATCH_ERROR" si le fichier est de type "**D**"

C'est un fichier de type "**R**"

F8B1- AD 83 C0 LDA C083 longueur de fiche
F8B4- 24 D3 BIT D3 teste si &(-NL)
F8B6- 30 0B BMI F8C3 si oui, suite en F8C3 (retourne la longueur de fiche)
F8B8- A0 04 LDY #04 sinon, retourne le nombre de fiches:
F8BA- B1 04 LDA (04),Y
F8BC- 48 PHA LL du nombre de fiches
F8BD- C8 INY
F8BE- B1 04 LDA (04),Y HH du nombre de fiches
F8C0- A8 TAY
F8C1- 68 PLA AY nombre de fiches
F8C2- 2C A0 00 BIT 00A0 continue en F8C7

Transfère A dans ACC1

F8C3- A0 00 LDY #00 le HH d'un nombre entier sur un octet est nul
F8C5- 24 A8 BIT A8 continue en F8C7

Transfère #FFFF dans ACC1

F8C6- A8 TAY #FF copié aussi dans HH

Intervertit AY (LLHH) en AY (LLHH) pour sous-programme D254

F8C7- 85 F2 STA F2 LL du nombre entier mis en réserve
F8C9- 98 TYA HH du nombre entier repris dans A
F8CA- A4 F2 LDY F2 LL du nombre entier repris dans Y

Transfère AY (LLHH) dans ACC1

F8CC- 4C 54 D2 JMP D254 JSR D499/ROM nombre en AY (LLHH) -> ACC1 (signé)

&() n'est pas utilisable avec un fichier de type "**D**"

F8CF- 4C E0 E0 JMP E0E0 "FILE_TYPE_MISMATCH_ERROR"

&() pour un fichier de type "**S**"

F8D2- 20 0E FD JSR FD0E re-initialise 00/01, 02/03, 04/05 et 06/07, sauvegarde les octets de rang #03, #04 et #05 du "Channel Buffer" en C086, C087, C088 et teste si la fin du fichier est atteinte (A = #FF et Z = 1)
F8D5- F0 EF BEQ F8C6 si fin de fichier, suite en F8C6 avec A = #FF
F8D7- 24 D3 BIT D3 sinon, teste si &(-NL)
F8D9- 30 E8 BMI F8C3 si oui, suite en F8C3 avec A = type d'enregistrement
F8DB- A9 00 LDA #00 si pas fin de fichier et &(NL),
F8DD- F0 E4 BEQ F8C3 retourne avec A = #00

EXÉCUTION DE LA COMMANDE SEDORIC TAKE

(Fichiers "**S**","**R**"et "**D**")
(sous-programme F8DF-F98F)

Rappel de la syntaxe

Pour un fichier **S**équentiel: **TAKE NL,liste_de_variables**. Lit dans le fichier de n° logique NL les variables indiquées dans la liste. Ces variables doivent être du même type qu'à l'écriture et doivent être accessibles avant la fin du fichier (sinon "END_OF_FILE_ERROR").

Pour un fichier **R**andom (à accès direct): **TAKE NL,n°_de_fiche**. Charge en mémoire la fiche indiquée qui doit bien sûr exister (sinon "BAD_RECORD_NUMBER_ERROR").

Pour un "fichier" d'accès **D**isque: **TAKE NL,piste,secteur(,lecteur)**. Charge en mémoire le secteur indiqué s'il existe (sinon "TYPE 10 I/O ERROR").

Dans les 3 cas, le fichier doit être préalablement ouvert à l'aide de la commande OPEN (sinon "FILE_NOT_OPEN_ERROR"). Le fichier ouvert avec OPEN et celui indiqué (NL) par TAKE doivent être du même type (sinon "FILE_TYPE_MISMATCH_ERROR"). Voir le préambule sur "l'utilisation des différents buffers" situé juste devant le sous-programme F3CF.

Variables utilisées

En voir la liste et les fonctions dans les généralités sur la gestion de fichiers de data situées devant le sous-programme F3CF.

Analyse de la syntaxe et saisie des paramètres

F8DF- 20 56 F9 JSR F956 vérifie l'existence de **FI**, la validité du NL, si le fichier est bien ouvert, demande la virgule suivante, initialise les pointeurs 00/01, 02/03, 04/05, 06/07, 0B, C083 et DRIVE, retourne avec Z = 1 si fichier "**D**", C = 0 si "**R**" et C = 1 si "**S**"
F8E2- D0 06 BNE F8EA continue en F8EA si ce n'est pas un fichier "**D**"

TAKE pour un fichier d'accès **D**isque

F8E4- 20 6B F9 JSR F96B lit à TXTPTR piste, secteur et (si indiqué) drive, initialise RWBUF au début du "Channel's own Data Buffer"
F8E7- 4C 73 DA JMP DA73 XPRSEC lit ce secteur selon DRIVE, PISTE, SECTEUR et

RWBUF dans le "Channel's own Data Buffer" correspondant au NL indiqué et retourne

Suite de l'analyse de la syntaxe et saisie des paramètres

F8EA- B0 11 BCS F8FD continue en F8FD si c'est un fichier **S**équentiel

TAKE pour un fichier "**R**" d'accès direct

Si tout est correct, 2 secteurs de la disquette sont lus dans le "General Buffer" (le premier secteur contient le début de la fiche choisie). La fiche proprement dite est alors copiée du "General Buffer" dans le "Channel's own Data Buffer".

F8EC- 20 1F F9 JSR F91F lit le n° de fiche indiqué à TXTPTR, le copie en 33/34 et teste si cette fiche existe (sinon, "BAD_RECORD_NUMBER_ERROR")
F8EF- 08 PHP sauvegarde les indicateurs 6502
F8F0- 78 SEI interdit les interruptions
F8F1- 20 84 F6 JSR F684 sachant que les fiches peuvent avoir une longueur comprise entre #03 et #FF (non documenté) et qu'elles sont placées les unes à la suite des autres dans le fichier, elles tombent la plupart du temps à cheval sur 2 secteurs. Le sous-programme F684 charge les 2 secteurs contenant la fiche spécifiée dans le "General Buffer" de **FI**, positionne le pointeur 06/07 au début de la fiche (qui se trouve dans le premier secteur) et retourne avec Y = 0
F8F4- B1 06 LDA (06),Y la fiche proprement dite (en fait un bloc de 256
F8F6- 91 02 STA (02),Y octets situés à partir du début de la fiche, ce
F8F8- C8 INY qui, la plupart du temps, inclut des octets supplémentaires) est alors copiée dans le "Channel's own Data Buffer" dont l'adresse est pointée par 02/03. Cette solution représente en fait une belle optimisation (code programme compact et rapide): cela aurait été en effet beaucoup plus compliqué avec une taille de fiches excédant 256 octets. Dans cette boucle Y varie de #00 à #FF
F8F9- D0 F9 BNE F8F4 reboucle en F8F4 tant qu'il en reste à copier
F8FB- 28 PLP récupère les indicateurs
F8FC- 60 RTS et retourne

TAKE pour un fichier **S**équentiel

F8FD- 20 2E ED JSR ED2E place l'adresse de la valeur de la variable indiquée à TXTPTR dans AY, D3/D4 et B8/B9 (ce sous-programme appartient à la commande LINPUT)
F900- 20 D9 FD JSR FDD9 si la fin du fichier n'est pas atteinte, copie un enregistrement complet (type, longueur et valeur de la variable) du "Channel's own Data Buffer" (charge un second secteur dans le buffer si nécessaire), vers le "General Buffer", 06/07 pointe sur le type, la longueur et la valeur du data
F903- 8A TXA type du data
F904- 20 7D F6 JSR F67D vérifie que type de data = type de variable
F907- A5 06 LDA 06
F909- A4 07 LDY 07 AY, pointeur dans l'enregistrement
F90B- 85 02 STA 02
F90D- 84 03 STY 03 02/03, pointeur dans le "General Buffer"
F90F- 20 DC F5 JSR F5DC place le data (la donnée) dans la variable BASIC
F912- 20 9E D3 JSR D39E XCRGOT relit le caractère à TXTPTR (sans incrémenter TXTPTR = CHRGOT), puis le convertit en MAJUSCULE, les espaces sont sautés, Z = 1 si fin

d'instruction (0 ou :), C = 0 si caractère chiffre 0 à 9 (soit #30 à #39), sinon C = 1, Y et X inchangés

F915- F0 E5 BEQ F8FC RTS en F8FC s'il n'y a plus de variable à pourvoir
F917- 20 2C D2 JSR D22C D067/ROM exige une "," place TXTPTR sur l'octet suivant
F91A- 4C FD F8 JMP F8FD reprend en F8FD pour la variable suivante

Lit le n° de fiche indiqué à TXTPTR et vérifie si la fiche existe
(sous-programme appelé par la commande PUT)

F91D- 18 CLC point d'entrée pour la commande PUT
F91E- 24 38 BIT 38 continue en F920

Lit le n° de fiche indiqué à TXTPTR et vérifie si la fiche existe

F91F- 38 SEC point d'entrée pour la commande TAKE
F920- 08 PHP sauvegarde les indicateurs 6502 dont C
F921- 20 FA D2 JSR D2FA E853/ROM évalue un nombre non signé à TXTPTR et revient avec ce nombre dans YA, 33/34 et D3/D4 et mise à jour de TXTPTR sur l'octet suivant ce nombre (C = 0, N et Z positionnés selon l'octet de poids faible)
F924- A0 04 LDY #04 index visant le nombre de fiches dans le descripteur
F926- B1 04 LDA (04),Y
F928- C5 33 CMP 33 la fiche demandée existe t-elle?
F92A- C8 INY compare le nombre de fiches actuelles
F92B- B1 04 LDA (04),Y indiqué dans le "Descriptor Buffer"
F92D- E5 34 SBC 34 et le n° de la fiche demandée
F92F- B0 08 BCS F939 la fiche existe déjà, termine en F939
F931- 28 PLP sinon, récupère les indicateurs dont C
F932- 90 07 BCC F93B si commande PUT, continue en F93B (crée la fiche)
F934- A2 10 LDX #10 si commande TAKE, "BAD_RECORD_NUMBER_ERROR"
F936- 4C 7E D6 JMP D67E incrémente X et traite l'erreur n° X

F939- 28 PLP la fiche existe, récupère les indicateurs
F93A- 60 RTS dont C et retourne au programme appelant

Crée une ou des fiches pour la commande PUT
(sous-programme F93B-F955, appelé par la commande PUT)

F93B- A0 04 LDY #04 index visant le nombre de fiches dans le descripteur
F93D- A5 33 LDA 33
F93F- 91 04 STA (04),Y LL du nombre de fiches requis
F941- C8 INY
F942- A5 34 LDA 34 HH du nombre de fiches requis
F944- 91 04 STA (04),Y (mise à jour du descripteur)
F946- 20 5F F8 JSR F85F copie dans RWBUF l'adresse présente en 04/05, c'est à dire l'adresse de début du "Descriptor Buffer"
F949- A0 13 LDY #13
F94B- B1 00 LDA (00),Y n° de piste où il faut écrire le descripteur
F94D- 48 PHA
F94E- C8 INY

F94F- B1 00 LDA (00),Y n° de secteur où il faut écrire le descripteur
F951- A8 TAY
F952- 68 PLA AY vise piste A secteur Y
F953- 4C 9E DA JMP DA9E XSAY sauve le descripteur indiqué par RWBUF et AY

Vérifie l'existence de **FI**, la validité du NL, si le fichier est bien ouvert, demande la virgule suivante, initialise les pointeurs 00/01, 02/03, 04/05, 06/07, 0B, C083 et DRIVE, retourne avec Z = 1 si fichier "**D**", C = 0 si "**R**" et C = 1 si "**S**"

(sous-programme appelé par les commandes PUT et TAKE)

F956- 20 7D F4 JSR F47D vérifie l'existence de **FI**, la validité du NL s'il existe et si le fichier est bien déjà ouvert
F959- 20 2C D2 JSR D22C D067/ROM exige une "," place TXTPTR sur l'octet suivant
F95C- 20 A8 F4 JSR F4A8 place l'adresse du début du "Channel Buffer" correspondant au NL en 00/01, celle du début du "Channel's own Data Buffer" en 02/03, celle du "Descriptor Buffer" en 04/05, celle du début du "General Buffer" en 06/07 et enfin, met à jour C083 (longueur d'une fiche ou #00) et 0B (flag "**S**/**R**/**D**") puis retourne avec Y = #00
F95F- 48 PHA sauve A temporairement (flag 0B, "**S**/**R**/**D**")
F960- A0 06 LDY #06
F962- B1 00 LDA (00),Y met DRIVE à jour avec le n° du drive sur lequel le
F964- 8D 00 C0 STA C000 fichier est ouvert (prépare le prochain accès)
F967- 68 PLA récupère A (type de fichier)
F968- C9 01 CMP #01 teste b0 du flag "**S**/**R**/**D**"
F96A- 60 RTS retourne avec Z = 1 s'il s'agit d'un fichier "**D**", avec C = 0 si fichier "**R**" et avec C = 1 si fichier "**S**"

Pour "fichier" **D**isque, lit à TXTPTR piste, secteur et (si indiqué) drive, initialise RWBUF au début du "Channel's own Data Buffer"

(sous-programme appelé par les commandes PUT et TAKE)

F96B- 20 7F D2 JSR D27F CF17/ROM et CF09/ROM et D8CB/ROM évalue un nombre entier à TXTPTR et le retourne dans X (numéro de piste)
F96E- 8E 01 C0 STX C001 et le copie dans PISTE
F971- 20 2C D2 JSR D22C D067/ROM exige une "," place TXTPTR sur l'octet suivant
F974- 20 7F D2 JSR D27F CF17/ROM et CF09/ROM et D8CB/ROM évalue un nombre entier à TXTPTR et le retourne dans X (numéro de secteur)
F977- 8E 02 C0 STX C002 et le copie dans SECTEUR
F97A- F0 06 BEQ F982 saute les 2 instructions suivantes si la fin des paramètres est atteinte
F97C- 20 2C D2 JSR D22C D067/ROM exige une "," lit le caractère suivant et le convertit en MAJUSCULE
F97F- 20 0D E6 JSR E60D valide le drive si celui-ci est indiqué, sinon valide DRVDEF
F982- 20 A8 F4 JSR F4A8 place l'adresse du début du "Channel Buffer" correspondant au NL en 00/01, celle du début du "Channel's own Data Buffer" en 02/03, celle du "Descriptor Buffer" en 04/05, celle du début du "General Buffer" en 06/07 et enfin, met à jour C083 (#00) et 0B (flag "**D**") puis retourne avec Y = #00

```
F985-    A5 02       LDA 02
F987-    A4 03       LDY 03      AY, début du "Channel's own Data Buffer"
F989-    8D 03 C0    STA C003
F98C-    8C 04 C0    STY C004    copie AY dans RWBUF
F98F-    60          RTS   et retourne
```

EXÉCUTION DE LA COMMANDE SEDORIC PMAP

(Fichiers "**D**")
(sous-programme F990-F995)

Rappel de la syntaxe

PMAP lecteur

Lit la bitmap de la disquette présente dans le lecteur indiqué et la copie dans BUF2. NB: PMAP signifie "**P**rend la bit**MAP**".

Variables utilisées

En voir la liste et les fonctions dans les généralités sur la gestion de fichiers de data situées devant le sous-programme F3CF.

Analyse de syntaxe et saisie du paramètre

F990- 20 0D E6 JSR E60D valide drive si indiqué, sinon valide DRVDEF
F993- 4C 4C DA JMP DA4C XPMAP prend le premier secteur de bitmap dans BUF2, vérifie le format, met à zéro le b7 de 2F (flag"première bitmap chargée").

EXÉCUTION DE LA COMMANDE SEDORIC SMAP

(Fichiers "**D**")
(sous-programme F996-F99B)

Rappel de la syntaxe

SMAP lecteur

Copie BUF2 dans la bitmap de la disquette présente dans le lecteur indiqué. Attention, il ne doit y avoir eu aucun LOAD, SAVE ou DIR entre les commandes PMAP et SMAP, sous peine de rendre la bitmap incohérente. NB: SMAP signifie "**S**auve la bit**MAP**".

Variables utilisées

En voir la liste et les fonctions dans les généralités sur la gestion de fichiers de data situées devant le sous-programme F3CF.

Analyse de syntaxe et saisie du paramètre

F996- 20 0D E6 JSR E60D valide drive si indiqué, sinon valide DRVDEF
F999- 4C 8A DA JMP DA8A l'entrée de cette routine XSMAP sauve BUF2 (bitmap) sur la disquette a été déportée en DC80 pour adaptation à la double bitmap. Un JSR DC80 plus direct aurait fait gagner un peu de temps.

EXÉCUTION DE LA COMMANDE SEDORIC FRSEC

(Fichiers "**D**")
(sous-programme F99C-F9BB)

Rappel de la syntaxe

FRSEC n°_de_piste,n°_de_secteur

Libère le secteur indiqué et incrémente le nombre de secteurs libres. Il ne se passe rien si ce secteur était déjà libre. Attention, la bitmap doit évidemment être mise à jour avec PMAP avant d'utiliser la commande FRSEC et sauvegardée après avec SMAP (non documenté). NB: FRSEC signifie libère (**FR**ee) un **SEC**teur.

Variables utilisées

En voir la liste et les fonctions dans les généralités sur la gestion de fichiers de data situées devant le sous-programme F3CF.

Analyse de syntaxe et saisie du paramètre

F99C- 20 7F D2 JSR D27F CF17/ROM et CF09/ROM et D8CB/ROM évalue un nombre entier à TXTPTR et le retourne dans X (le n° de piste où se trouve le secteur à libérer)
F99F- 8A TXA
F9A0- 48 PHA empile ce n° de piste
F9A1- 29 7F AND #7F force à zéro le n° de face
F9A3- CD 06 C2 CMP C206 vérifie que le n° de piste indiqué est valide
F9A6- B0 20 BCS F9C8 sinon, "ILLEGAL_QUANTITY_ERROR"
F9A8- 20 2C D2 JSR D22C D067/ROM exige une "," place TXTPTR sur l'octet suivant
F9AB- 20 7F D2 JSR D27F CF17/ROM et CF09/ROM et D8CB/ROM évalue un nombre entier à TXTPTR et le retourne dans X (le n° de secteur à libérer)
F9AE- 8A TXA garde ce n° de secteur dans A
F9AF- CA DEX vérifie que ce n° n'est pas #00
F9B0- 30 16 BMI F9C8 sinon "ILLEGAL_QUANTITY_ERROR"
F9B2- EC 07 C2 CPX C207 vérifie que le n° de piste indiqué est valide
F9B5- B0 11 BCS F9C8 sinon "ILLEGAL_QUANTITY_ERROR"
F9B7- A8 TAY
F9B8- 68 PLA AY coordonnées secteur Y de la piste A
F9B9- 4C 15 DD JMP DD15 XDETSE libère le secteur AY et incrémente le nombre de secteurs libres dans la bitmap courante dans BUF2. Retourne avec C = 1 si ce secteur était déjà libre. Ne pas oublier de sauver le plus tôt possible cette nouvelle bitmap avec SMAP.

Il serait intéressant d'écrire une nouvelle commande FXSEC (**F**i**X**e un **SEC**teur) de syntaxe: FXSEC n°_de_piste,n°_de_secteur(,drive) qui ferait comme FRSEC, mais marquerait "occupé" le secteur indiqué afin qu'on ne puisse plus y écrire (utile après avoir reçu une "_WRITE_FAULT_" ou une "_READ_FAULT_" ERROR). Pour cela, le masque obtenu en DD15 doit être inversé et le ORA en DD18 remplacé par un AND.

EXÉCUTION DE LA COMMANDE SEDORIC CRESEC

(Fichiers "**D**")
(sous-programme F9BC-F9CA)

Rappel de la syntaxe

CRESEC

Cherche un secteur libre dans la bitmap courante, présente dans BUF2. S'il en trouve un, réserve ce secteur, décrémente le nombre de secteurs libres et retourne les coordonnées AY de ce secteur qui sont également copiées dans les variables FP et FS. "DISK_FULL_ERROR" s'il ne trouve pas de secteur libre. Attention, la bitmap doit être mise à jour avec un PMAP avant d'exécuter la commande CRESEC et sauvegardée après avec SMAP (non documenté).

NB: CRESEC signifie **CRE**e un **SEC**teur (ce qui n'est pas très bien choisi), FP signifie **F**ree **P**iste (n° de piste où se trouve le secteur libre) et FS **F**ree **S**ector (n° du secteur libre). Tout çà c'est trouvé comme franglais!

Variables utilisées

En voir la liste et les fonctions dans les généralités sur la gestion de fichiers de data situées devant le sous-programme F3CF.

Entrée de la commande CRESEC

F9BC- 20 6C DC JSR DC6C XLIBSE cherche un secteur libre dans la bitmap dans BUF2, retourne avec A = n° de piste et Y = n° de secteur (sinon "DISK_FULL_ERROR")
F9BF- 48 PHA empile le n° de piste (et n° de face)
F9C0- 98 TYA garde le n° de secteur dans Y
F9C1- 20 ED D7 JSR D7ED assigne le n° de secteur à la variable FS
F9C4- 68 PLA récupère les n° de piste et de face
F9C5- 4C EA D7 JMP D7EA assigne les n° de piste et de face à la variable FP et retourne

F9C8- 4C 20 DE JMP DE20 "ILLEGAL_QUANTITY_ERROR"

EXÉCUTION DE LA COMMANDE SEDORIC PUT

(Fichiers "**S**", "**R**" et "**D**")
(sous-programme F9CB-FA4F)

Rappel de la syntaxe

Pour un fichier **S**équentiel: **PUT NL,liste_de_variables**. Ecrit dans le fichier de n° logique NL les variables indiquées dans la liste. Il doit y avoir concordance de type ("réel", "entier" ou "chaîne") entre les variables et le fichier, sauf si la fin du fichier est atteinte, auquel cas le fichier est allongé sans contrainte de type ni de longueur, puisqu'un nouvel enregistrement est créé (s'il y a encore de la place sur la disquette). Les chaînes sont "alignées" à gauche (à droite, elles sont tronquées ou complétées avec des espaces).

Pour un fichier **R**andom (à accès direct): **PUT NL,n°_de_fiche**. Copie sur la disquette, dans le fichier de n° logique NL, la fiche indiquée, présente en mémoire. Si la fiche n'existe pas, elle sera créée (s'il y a encore de la place sur la disquette), ainsi que toutes les fiches de n° inférieur qui n'existent pas encore (conclusion, pour épargner de la place, il vaut mieux créer des fiches de n° consécutifs). Si tout se déroule sans problème, les data sont lus de 2 secteurs consécutifs de la disquette (le premier secteur contient le début de la fiche choisie) dans le "General Buffer". La fiche est alors recopiée, à sa place exacte, du "Channel's own Data Buffer" dans le "General Buffer". Contrairement à ce qui se passe pour la commande TAKE, la longueur réelle de la fiche sert ici pour le décompte du nombre d'octets recopiés, sous peine d'écraser le début de la fiche suivante. Enfin, les 2 secteurs sont réécrits, à leur place, sur la disquette.

Pour un "fichier" d'accès **D**isque: **PUT NL,piste,secteur(,lecteur)**. Ecrit sur la disquette dans le secteur indiqué les 256 octets présents dans le tampon en mémoire dans le "Channel's own Data Buffer".

Dans les 3 cas, le fichier doit être préalablement ouvert à l'aide de la commande OPEN (sinon "FILE_NOT_OPEN_ERROR"). Le fichier ouvert avec OPEN et celui indiqué (NL) par PUT doivent être du même type (sinon "FILE_TYPE_MISMATCH_ERROR"). L'écriture sur la disquette se fait immédiatement après chaque PUT. Voir le préambule sur "l'utilisation des différents buffers" situé juste devant le sous-programme F3CF.

Variables utilisées

En voir la liste et les fonctions dans les généralités sur la gestion de fichiers de data situées devant le sous-programme F3CF.

Informations non documentées

ATTENTION: dans le cas du PUT pour un fichier de type accès **D**isque, il n'y a pas de vérification de la validité des n° de piste et de secteur indiqués.

Analyse de la syntaxe et saisie des paramètres

F9CB- 20 56 F9 JSR F956 vérifie l'existence de **FI**, la validité du NL, si le fichier est bien ouvert, demande la virgule suivante, initialise les pointeurs 00/01, 02/03, 04/05, 06/07, 0B, C083 et DRIVE, retourne avec Z = 1 si fichier "**D**", C = 0 si "**R**" et C = 1 si "**S**"
F9CE- D0 06 BNE F9D6 continue en F9D6 si c'est un fichier "**R**" ou "**S**"

Commande PUT pour fichier de type "**D**":

F9D0- 20 6B F9 JSR F96B lit les coordonnées indiquées (secteur, piste et éventuellement drive), initialise RWBUF au début du "Channel's own Data Buffer"

F9D3- 4C A4 DA JMP DAA4 XSVSEC écrit un secteur selon DRIVE, PISTE, SECTEUR et RWBUF

Suite commune pour fichiers "**R**" ou "**S**":

F9D6- B0 17 BCS F9EF continue en F9EF si c'est un fichier **S**équentiel.
Sinon, c'est donc un fichier "**R**" (à accès direct)

Commande PUT pour fichier de type "**R**": Lit le n° de fiche indiqué, vérifie que la fiche existe dans le fichier, sinon la crée (ainsi que celles qui la précèdent si nécessaire) et la sauve

F9D8- 20 1D F9 JSR F91D lit le n° de fiche indiqué à TXTPTR et vérifie si la fiche existe dans le fichier. La crée si ce n'est pas le cas.
F9DB- 08 PHP sauvegarde les indicateurs 6502
F9DC- 78 SEI interdit les interruptions
F9DD- 20 84 F6 JSR F684 sachant que les fiches peuvent avoir une longueur comprise entre #03 et #FF (non documenté) et qu'elles sont placées les unes à la suite des autres dans le fichier, elles tombent la plupart du temps à cheval sur 2 secteurs. Le sous-programme F684 charge ces 2 secteurs contenant la fiche spécifiée dans le "General Buffer" de **FI**, positionne le pointeur 06/07 au début de la fiche (qui se trouve dans le premier secteur) et retourne avec Y = 0
F9E0- B1 02 LDA (02),Y lit un octet dans le "Channel's own Data Buffer"
F9E2- 91 06 STA (06),Y et l'écrit dans le "General Buffer"
F9E4- C8 INY vise le suivant
F9E5- CC 83 C0 CPY C083 compare avec la longueur de la fiche
F9E8- D0 F6 BNE F9E0 et reboucle s'il en reste à copier
F9EA- 20 27 F7 JSR F727 écrit sur la disquette les 2 secteurs de data ("General Buffer") contenant la fiche spécifiée
F9ED- 28 PLP récupère les indicateurs
F9EE- 60 RTS

Commande PUT pour fichier de type "**S**":

F9EF- 20 24 D2 JSR D224 JSR CF17/ROM évalue une expression dans la liste des variables à TXTPTR, retourne avec une valeur numérique dans ACC1 (et #00 dans 28) ou l'adresse d'une chaîne dans D3/D4 (et #FF dans 28) et A, N et Z positionnés selon l'exposant (résultat numérique) ou la longueur de chaîne
F9F2- 20 0E FD JSR FD0E re-initialise 00/01, 02/03, 04/05 et 06/07, sauvegarde les octets de rang #03, #04 et #05 du "Channel Buffer" en C086, C087, C088 et teste si la fin du fichier est atteinte (Z = 1)
F9F5- D0 24 BNE FA1B si ce n'est pas le cas, continue en FA1B pour mettre à jour l'enregistrement existant

Fin de fichier atteinte: ajoute le nouvel enregistrement au bout

F9F7- A2 05 LDX #05 longueur d'enregistrement par défaut (nombre réel)
F9F9- A0 00 LDY #00 type d'enregistrement par défaut (nombre réel)
F9FB- 24 28 BIT 28 teste la variable indiquée ("chaîne" ou "nombre")
F9FD- 10 0D BPL FA0C continue en FA0C si c'est un nombre (on a tous les paramètres)

sinon, c'est une chaîne et il faut compléter les paramètres

La variable indiquée est une chaîne

F9FF-	A5 D3	LDA D3	
FA01-	A6 D4	LDX D4	
FA03-	85 91	STA 91	adresse de cette chaîne
FA05-	86 92	STX 92	
FA07-	B1 91	LDA (91),Y	lit la longueur de la chaîne
FA09-	AA	TAX	et garde cette information dans X
FA0A-	A0 80	LDY #80	type d'enregistrement pour une chaîne

Suite commune chaîne/nombre

FA0C-	8C 7F C0	STY C07F	type d'enregistrement

FA0F- 20 39 FA JSR FA39 écrit l'enregistrement dans le "Channel's own Data Buffer" en utilisant le secteur suivant du fichier si nécessaire. Les data présents sur la disquette sont mis à jour par l'intermédiaire du "Channel's own Data Buffer": ils sont copiés du "General Buffer" au point indiqué dans le "Channel's own Data Buffer" jusqu'à ce que la fin de celui-ci soit atteinte. Le "Channel's own Data Buffer" est alors sauvé et rechargé avec le secteur suivant contenant la suite de l'enregistrement à mettre à jour.

FA12-	A9 FF	LDA #FF	flag "fin de fichier **S**équentiel"
FA14-	20 CC FD	JSR FDCC	lit l'index Y du "Channel's own Data Buffer"
FA17-	91 02	STA (02),Y	écrit le flag "fin de fichier **S**équentiel"
FA19-	30 10	BMI FA2B	suite forcée en FA2B

La fin du fichier n'était pas encore atteinte: remplace l'ancien enregistrement par le nouveau

FA1B- 20 D9 FD JSR FDD9 copie un enregistrement complet (type, longueur et valeur de la variable) du "Channel's own Data Buffer" (charge un second secteur dans le buffer si nécessaire), vers le "General Buffer", 06/07 pointe sur le type, la longueur et la valeur du data

FA1E-	48	PHA	empile la longueur de l'enregistrement
FA1F-	8A	TXA	garde le type de l'enregistrement dans X
FA20-	20 7D F6	JSR F67D	vérifie que type de data = type de variable

FA23- 20 2A FD JSR FD2A re-initialise 00/01, 02/03, 04/05 et 06/07, restaure les octets C088, C087 et C086 dans les octets de rang #05, #04 et #03 du "Channel Buffer", l'octet lu en C086 est comparé avant écriture avec l'octet qu'il écrasera dans cette zone, RTS en FD29 s'ils sont identiques, sinon charge un secteur du fichier ouvert

FA26-	68	PLA	récupère la longueur de l'enregistrement
FA27-	AA	TAX	récupère le type de l'enregistrement

FA28- 20 39 FA JSR FA39 écrit l'enregistrement dans le "Channel's own Data Buffer" en utilisant le secteur suivant du fichier si nécessaire

FA2B- 20 9E D3 JSR D39E XCRGOT relit le caractère à TXTPTR (sans incrémenter TXTPTR = CHRGOT), puis le convertit en MAJUSCULE, les espaces sont sautés, Z = 1 si fin d'instruction (0 ou :), C = 0 si caractère chiffre 0 à 9 (soit #30 à #39), sinon C = 1, Y et X inchangés

FA2E- F0 06 BEQ FA36 saute les 2 instructions suivantes s'il n'y a plus de paramètre, c'est à dire, sauve le secteur sur la disquette

FA30-	20 2C D2	JSR D22C	D067/ROM exige une "," place TXTPTR sur l'octet suivant
FA33-	4C EF F9	JMP F9EF	reprend au début en F9EF

FA36- 4C 46 FD JMP FD46 sauve sur la disquette le secteur qui est présent dans le "Channel's own Data Buffer" du "Channel Buffer"

Copie l'enregistrement sur la disquette

Ecrit l'enregistrement dans le "General Buffer", puis les data présents sur la disquette sont mis à jour par l'intermédiaire du "Channel's own Data Buffer": ils sont copiés du "General Buffer" au point indiqué dans le "Channel's own Data Buffer" jusqu'à ce que la fin de celui-ci soit atteinte. Le "Channel's own Data Buffer" est alors sauvé et rechargé avec le secteur suivant contenant la suite de l'enregistrement à mettre à jour.

FA39-	8E 7E C0	STX C07E	longueur de l'enregistrement
FA3C-	A5 06	LDA 06	
FA3E-	A4 07	LDY 07	prend l'adresse du début du "General Buffer"
FA40-	85 02	STA 02	et la copie en 02/03 (adresse utilisée en FC9E)
FA42-	84 03	STY 03	
FA44-	18	CLC	pour LSET (alignement à gauche)
FA45-	A0 00	LDY #00	vise au début du "General Buffer"
FA47-	20 9E FC	JSR FC9E	copie l'enregistrement dans le "General Buffer"
FA4A-	20 A8 F4	JSR F4A8	place l'adresse du début du "Channel Buffer" correspondant au NL en 00/01, celle du début du "Channel's own Data Buffer" en 02/03, celle du "Descriptor Buffer" en 04/05, celle du début du "General Buffer" en 06/07 et enfin, met à jour C083 (#00) et 0B (flag "**S**") puis retourne avec Y = #00
FA4D-	4C 38 FE	JMP FE38	les data déjà présents sur la disquette sont mis à jour par l'intermédiaire du "Channel's own Data Buffer": ils sont copiés du "General Buffer" au point indiqué dans le "Channel's own Data Buffer" jusqu'à ce que la fin de celui-ci soit atteinte. Le "Channel's own Data Buffer" est alors sauvé et rechargé avec le secteur suivant contenant la suite de l'enregistrement à mettre à jour.

EXÉCUTION DE LA COMMANDE SEDORIC OPEN

(Fichiers "**S**", "**R**" et "**D**")
(sous-programme FA50-FABA)

Rappel de la syntaxe (trois possibilités):

OPEN D,NL(,L) pour un accès **D**isque de n° logique **NL** sur le **L**ecteur **L**. Cet accès disque, un peu inhabituel, permet de lire ou d'écrire un secteur de la disquette présente dans le lecteur **L** dans le tampon de 256 octets créé à cet usage par la commande OPEN D (avec un espace obligatoire entre OPEN et D)

OPEN R,nom_de_fichier_non_ambigu,NL(,LF ,NR) pour ouvrir un fichier à accès direct (**R**andom) de nom **nom_de_fichier_non_ambigu**, de n° logique **NL**, comportant un **N**ombre de fiches à **R**éserver **NR**, la **L**ongueur de chaque **F**iche étant de **LF** caractères. Attention **LF** et **NR** ne sont à préciser que la première fois, lors de la création du ficher à accès direct où ils sont alors obligatoires. L'extension au-delà de ce nombre de fiches est automatique en cas de dépassement (dans la limite de la place disponible sur la disquette). La place occupée sur la disquette par le fichier nom_de_fichier_non_ambigu est égale au nombre d'octets de data, soit NR x LF / 256 secteurs.

OPEN S,nom_de_fichier_non_ambigu,NL pour ouvrir un fichier **S**équentiel de nom

nom_de_fichier_non_ambigu et de n° logique **NL**. Cette commande réserve un tampon en mémoire et place le pointeur au début du fichier.

Dans les 3 cas, le fichier est créé s'il n'existe pas. OPEN associe le **nom_de_fichier_non_ambigu** à un numéro logique **NL**. et **NL** peut être n'importe quel nombre de 0 à 63. Il peut donc y avoir 64 fichiers ouverts simultanément en mémoire. Le fichier doit évidemment ne pas être déjà ouvert, sinon "FILE_ALREADY_OPEN_ERROR" et être du bon type, sinon "FILE_TYPE_MISMATCH_ERROR".

Variables utilisées

En voir la liste et les fonctions dans les généralités sur la gestion de fichiers de data situées devant le sous-programme F3CF.

Non documenté

A partir du moment où un OPEN a été utilisé et ce jusqu'au dernier CLOSE, il est absolument interdit d'utiliser la commande BASIC DIM! (ce n'est pas dans le manuel !!!). Ceci est dû au fait que les tableaux sont toujours créés au début de la zone des tableaux BASIC et vont faire "disparaître" le pseudo-tableau **FI**.

Dans le cas de OPEN R, lors de la création du fichier (première ouverture avec OPEN), la longueur de fiche doit être comprise entre #03 et #FF.

Organigramme de la commande OPEN **S** pour un nouveau fichier

FA50 analyse du premier paramètre "**D**", "**R**" ou "**S**". C'est "**S**", continue en FA96.
FA96 JSR FB08 ce long sous-programme:

- initialise 0B avec #80 (flag "**S**") et F9 avec #10 (FTYPE "séquentiel"),

- saisit et vérifie nom_de_fichier_non_ambigu présent à TXTPTR,

- demande la virgule qui doit suivre,

- vérifie l'existence de "**FI**", le crée s'il n'existe pas encore,

- saisit et vérifie le NL indiqué à TXTPTR, le copie en 0A,

- vérifie si le fichier n'est pas déjà ouvert (ici pas de "FILE_ALREADY_OPEN_ERROR" possible car c'est un nouveau fichier),

- cherche le fichier nom_de_fichier_non_ambigu dans le catalogue (empile Z = 1 car pas trouvé),

- crée un fichier fictif de un secteur de data,

- vérifie FTYPE (ici pas de "FILE_TYPE_MISMATCH_ERROR", car nouveau fichier),

- insère un "Channel Buffer" de #121 octets à la fin du "General Buffer",

- place l'offset de ce "Channel Buffer" dans la "Table NL",

- initialise 00/01, 02/03, 04/05, 06/07,

- place le flag #80 (fichier "**S**") au début du "Channel Buffer" (rang #00),

- place #00 (la longueur de fiche fictive) au début du "Channel Buffer" (octet de rang #01) et en C083,

- copie "l'entrée" de catalogue dans le "Channel Buffer" (rangs #07 à #16),

- initialise le nombre de descripteurs chargés à #FF et l'adresse de chargement à l'octet de rang #117 (début du "Descriptor Buffer"),

- charge le descripteur dans le "Channel Buffer" à partir de l'offset #117, c'est à dire dans le "Descriptor Buffer",

- copie DRIVE dans l'octet de rang #06 du "Channel Buffer",

- dépile Z (à 1 car le fichier vient d'être créé)

suite FA9D:

- place #0C, #00 et #00 dans les octets de rang #03, #04 et #05 du "Channel Buffer" (index de lecture dans la liste des coordonnées, n° du descripteur courant et index dans le buffer),

- écrit #FF au début du "Channel's own Data Buffer" (marque de fin de fichier) et continue en FD66

JMP FD66:

- sauve sur la disquette ce premier secteur data du fichier qui est présent dans le "Channel's own Data Buffer". En fait, ce fichier, qui venait d'être créé, ne contenait qu'un secteur de data fictif, qui est remplacé par le secteur préparé ci-dessus, dont le marqueur de fin de fichier est placé au premier octet du tampon (fichier vide).

Dans tous les cas, le début du premier enregistrement est maintenant dans le "Channel's own Data Buffer" prêt pour TAKE ou PUT et l'octet de rang #05 du "Channel Buffer" est l'index de valeur initiale zéro pointant sur le premier octet de l'enregistrement dans le "Channel's own Data Buffer".

Organigramme de la commande OPEN **R** pour un nouveau fichier

FA50 analyse du premier paramètre "**D**", "**R**" ou "**S**". C'est "**R**", continue en FA80
FA7C suite de l'analyse de syntaxe
FA80 JSR FB08 ce long sous-programme:

- initialise 0B avec #00 (flag "**R**") et F9 avec #08 (FTYPE "accès direct"),

- saisit et vérifie nom_de_fichier_non_ambigu présent à TXTPTR,

- demande la virgule qui doit suivre,

- vérifie l'existence de "**FI**", le crée s'il n'existe pas encore,

- saisit et vérifie le NL indiqué à TXTPTR, le copie en 0A,

- vérifie si le fichier n'est pas déjà ouvert ("FILE_ALREADY_OPEN_ERROR"),

- cherche le fichier nom_de_fichier_non_ambigu dans le catalogue (empile Z = 1 car pas trouvé),

- crée un fichier d'une fiche selon la longueur de fiche indiquée à TXTPTR,

- vérifie FTYPE (ici pas de "FILE_TYPE_MISMATCH_ERROR", car nouveau fichier),

- insère un "Channel Buffer" de #121 octets à la fin du "General Buffer",

- place l'offset de ce "Channel Buffer" dans la "Table NL",

- initialise 00/01, 02/03, 04/05, 06/07,

- place le flag "**R**" (#00) au début du "Channel Buffer" (octet de rang #00),

- place la longueur de fiche dans le "Channel Buffer" (rang #01) et en C083,

- copie "l'entrée" de catalogue dans le "Channel Buffer" (rangs #06 à #17),

- initialise le nombre de descripteurs déjà chargés avec la valeur #FF et l'adresse de chargement à l'octet de rang #117 du "Channel Buffer",

- charge le descripteur dans le "Channel Buffer" à partir de l'offset #117, c'est à dire dans le "Descriptor Buffer",

- copie DRIVE dans l'octet de rang #06 du "Channel Buffer",

- dépile Z (à 1 car le fichier vient d'être créé)

FA87 simple RTS si le fichier existait déjà, ce qui n'est pas le cas

FA89 demande une virgule
et positionne TXTPTR sur le nombre de fiches à réserver qui doit se trouver à la fin des paramètres

FA8C JMP F9D8:

- lit le nombre de fiches à réserver et crée celles qui n'existent pas encore (lors de la création du nouveau fichier une seule fiche a été créée). En fait, une sorte de "réservation" est faite en mettant simplement à jour le nombre de fiches dans le descripteur présent en mémoire et sur la disquette,

- par appel au sous-programme F684, alloue autant de secteurs que nécessaire pour toutes ces fiches et termine en chargeant dans le "General Buffer" les deux secteurs correspondant à la dernière fiche réservée

avec 06/07 pointant sur le début de la fiche,

- effectue une pseudo mise à jour de la fiche en copiant une fiche fictive du "Channel's own Data Buffer" dans le "General Buffer" et sauve les deux secteurs de ce dernier sur la disquette.

Organigramme de la commande OPEN **D**

FA50 analyse du premier paramètre "**D**", "**R**" ou "**S**". C'est "**D**", continue en FA5C:

- valide DRVDEF

- vérifie l'existence de "**FI**", le crée s'il n'existe pas encore,

- saisit et vérifie le NL indiqué à TXTPTR, le copie en 0A,

- vérifie si le fichier n'est pas déjà ouvert ("FILE_ALREADY_OPEN_ERROR"),

- s'il y a une indication de drive à TXTPTR, valide celui-ci,

- insère un "Channel Buffer" de #121 octets à la fin du "General Buffer",

- place l'offset de ce "Channel Buffer" dans la "Table NL",

- initialise 00/01, 02/03, 04/05, 06/07,

- place le flag "**D**" (#01) au début du "Channel Buffer" (octet de rang #00),

- place #(1)00 dans le "Channel Buffer" (rang #01) et en C083 (LF),

- copie DRIVE dans l'octet de rang #06 du "Channel Buffer" et retourne

NB: il n'existe pas de FTYPE pour les "pseudo-fichiers" d'accès disquette

Entrée de la commande OPEN

Analyse de la syntaxe et saisie des paramètres

FA50- 48 PHA sauve le paramètre D, R ou S
FA51- 20 98 D3 JSR D398 XCRGET incrémente TXTPTR, lit un caractère (CHRGET), les espaces sont sautés, le met dans A, le convertit en MAJUSCULE, Z = 1 si fin d'instruction (0 ou :), C = 0 si chiffre 0 à 9 (soit #30 à #39), sinon C =1. Y et X inchangés (positionne TXTPTR)
FA54- 20 2C D2 JSR D22C JSR D067/ROM demande une "," lit le caractère suivant, le convertit en MAJUSCULE et l'écrase avec le PLA suivant (il faudra donc le relire)!
FA57- 68 PLA récupère le paramètre D, R ou S
FA58- C9 44 CMP #44 est-ce un "D"?
FA5A- D0 20 BNE FA7C sinon, poursuit l'analyse en FA7C

OPEN D

FA5C- AD 09 C0 LDA C009 si oui, copie DRVDEF (n° du lecteur par défaut)
FA5F- 8D 00 C0 STA C000 dans DRIVE (lecteur à utiliser)
FA62- 20 7F F4 JSR F47F vérifie l'existence de **FI**, la validité du NL indiqué à TXTPTR, si le fichier n'est pas déjà ouvert
FA65- F0 06 BEQ FA6D si Z = 1 (fin d'instruction), continue en FA6D
FA67- 20 2C D2 JSR D22C D067/ROM exige une "," lit le caractère suivant et le convertit en MAJUSCULE
FA6A- 20 0D E6 JSR E60D valide drive si indiqué, sinon re-valide DRVDEF (!)
FA6D- A9 00 LDA #00 initialise A avec #00, pour une longueur d'enregistrement de #100 octets (un secteur) par défaut
FA6F- A0 01 LDY #01 initialise Y avec #01 (flag "**D**")
FA71- 20 CB FA JSR FACB augmente de #121 octets la taille de **FI**, place l'offset de début de ce nouveau "Channel Buffer" dans la "Table NL", initialise 00/01, 02/03, 04/05, 06/07, place le flag "**D**" et la longueur de secteur au début du "Channel Buffer" (et en C083 pour la longueur)

Copie DRIVE dans l'octet de rang #06 du "Channel Buffer"

FA74- A0 06 LDY #06 index pour écriture
FA76- AD 00 C0 LDA C000 copie DRIVE (lecteur à utiliser)
FA79- 91 00 STA (00),Y dans l'octet de rang #06 du "Channel Buffer"
FA7B- 60 RTS

Suite de l'analyse des paramètres

FA7C- C9 52 CMP #52 est-ce un "R"?
FA7E- D0 12 BNE FA92 sinon, poursuit l'analyse en FA92

OPEN R

FA80- A9 00 LDA #00 si oui, initialise A avec #00 (flag "**R**")
FA82- A0 08 LDY #08 et initialise Y avec #08 (FTYPE "fichier direct")
FA84- 20 08 FB JSR FB08 ce long sous-programme initialise 0B (flag "**R**") et F9 (FTYPE), saisit et vérifie nom_de_fichier_non_ambigu présent à TXTPTR et demande la virgule qui doit suivre. Il vérifie l'existence de **FI**, le crée si besoin, saisit et vérifie le NL indiqué à TXTPTR, le copie en 0A, vérifie si le fichier n'est pas déjà ouvert (sinon "FILE_ALREADY_OPEN_ERROR"), cherche le fichier nom_de_fichier_non_ambigu dans le catalogue (empile Z = 1 si pas trouvé), le crée si besoin (une seule fiche selon la longueur indiquée à TXTPTR). Il vérifie FTYPE ("FILE_TYPE_MISMATCH_ERROR" si incompatibilité), insère un "Channel Buffer" de #121 octets à la fin du "General Buffer", place l'offset de ce "Channel Buffer" dans la "Table NL", initialise 00/01, 02/03, 04/05, 06/07, place le flag "**R**" et la longueur de fiche LF au début du "Channel Buffer" (et en C083 pour LF). Il copie "l'entrée" de catalogue dans le "Channel Buffer", initialise le nombre de descripteurs et l'adresse de chargement (début du "Descriptor Buffer"). Enfin, il charge le ou les descripteurs dans le "Channel Buffer" à partir de l'offset #117, c'est à dire dans le "Descriptor Buffer", copie DRIVE dans l'octet de rang #06 et dépile Z (qui est à 1 si le fichier vient d'être créé).

FA87- D0 F2 BNE FA7B simple RTS si le fichier existait déjà
FA89- 20 2C D2 JSR D22C D067/ROM exige une "," place TXTPTR sur l'octet suivant
FA8C- 4C D8 F9 JMP F9D8 lit le nombre de fiches, vérifie que la place existe, sinon

augmente la taille du "Channel's own Buffer". Lors de la création du fichier le nombre de fichier indiqué à TXTPTR sert ainsi a créer les fiches requises.

FA8F- 4C 23 DE JMP DE23 "SYNTAX_ERROR"

Suite de l'analyse des paramètres

FA92- C9 53 CMP #53 est-ce un "S"? (dernière possibilité)
FA94- D0 F9 BNE FA8F sinon, "SYNTAX_ERROR"

OPEN S

FA96- A9 80 LDA #80 si oui, A = #80 (flag "**S**" pour initialiser 0B)
FA98- A0 10 LDY #10 et Y = #10 (FTYPE "séquentiel" pour initialiser F9)
FA9A- 20 08 FB JSR FB08 ce long sous-programme initialise 0B (flag "**S**" #80) et F9 (FTYPE séquentiel #10), saisit et vérifie le nom de fichier nom_de_fichier_non_ambigu présent à TXTPTR et demande la virgule qui doit suivre. Il vérifie l'existence de **FI**, le crée si besoin, saisit et vérifie le NL indiqué à TXTPTR, le copie en 0A, vérifie si le fichier n'est pas déjà ouvert (sinon "FILE_ALREADY_OPEN_ERROR"), cherche le fichier nom_de_fichier_non_ambigu dans le catalogue (empile Z = 1 si pas trouvé), le crée si besoin (un seul secteur vide). Il vérifie FTYPE ("FILE_TYPE_MISMATCH_ERROR" si incompatibilité), insère un "Channel Buffer" de #121 octets à la fin du "General Buffer", place l'offset de ce "Channel Buffer" dans la "Table NL", initialise 00/01, 02/03, 04/05, 06/07, place #80 (flag "**S**") et #00 (LF) au début du "Channel Buffer" (et #00 en C083 pour LF). Il copie "l'entrée" de catalogue dans le "Channel Buffer", initialise le nombre de descripteurs et l'adresse de chargement. Enfin, il charge le ou les descripteurs dans le "Channel Buffer" à partir de l'offset #117, c'est à dire dans le "Descriptor Buffer", copie DRIVE dans l'octet de rang #06 et dépile Z (à 1 si le fichier vient d'être créé).

Place #0C, #00 et #00 dans les octets de rang #03, #04 et #05 du "Channel Buffer", continue en FD44 si fichier existant (charge le premier secteur du fichier), sinon écrit #FF au début du "Channel's own Data Buffer" et continue en FD66 (sauve le premier secteur du fichier)

FA9D- 08 PHP sauvegarde Z (à zéro si le fichier existait déjà)
FA9E- A0 03 LDY #03 index pour écriture dans le "Channel Buffer"
FAA0- A9 0C LDA #0C écrit #0C (index de lecture dans le premier descripteur)
FAA2- 91 00 STA (00),Y dans l'octet de rang #03 du "Channel Buffer"
FAA4- C8 INY
FAA5- A9 00 LDA #00 écrit #00 dans l'octet de rang #04 du
FAA7- 91 00 STA (00),Y "Channel Buffer" (n° du descripteur courant)
FAA9- C8 INY écrit #00 dans l'octet de rang #05 du
FAAA- 91 00 STA (00),Y "Channel Buffer" (index dans le tampon)
FAAC- 28 PLP récupère les indicateurs dont Z (à 1 si nouvellement créé)
FAAD- D0 09 BNE FAB8 continue en FD44 si Z = 0 (le fichier existait déjà)

Ecrit un flag "fin de fichier **S**équentiel" au début du nouveau fichier

FAAF- A0 00 LDY #00 index pour écriture
FAB1- A9 FF LDA #FF écrit #FF au début du "Channel's own Data Buffer", pour

marquer la fin des data (le fichier est vide)
FAB3- 91 02 STA (02),Y dont l'adresse est indiquée en 02/03 (commence à l'octet de rang #17 du "Channel Buffer")
FAB5- 4C 46 FD JMP FD46 sauve sur la disquette le secteur qui est présent dans le "Channel's own Data Buffer" du "Channel Buffer". En fait, ce fichier, qui venait d'être créé, ne contenait qu'un secteur de data fictif, qui est remplacé par le secteur préparé en FAB1, dont le marqueur de fin de fichier est placé au premier octet du tampon. A ce stade, le nouveau fichier (vide) est ouvert, pointeur visant le premier octet.

Ouvre le fichier qui existait déjà

FAB8- 4C 44 FD JMP FD44 charge le premier secteur du fichier qui se trouvait déjà sur la disquette et le place dans le "Channel's own Data Buffer". A ce stade, le fichier est ouvert, pointeur visant le premier octet.

EXÉCUTION DE LA COMMANDE SEDORIC REWIND

(Fichiers "**S**")
(sous-programme FABB-FACA)

Rappel de la syntaxe

REWIND NL

Place le pointeur de fichier au début du fichier de n° logique NL. Le fichier doit évidemment être ouvert, sinon "FILE_NOT_OPEN_ERROR" et être du bon type, sinon "FILE_TYPE_MISMATCH_ERROR".

Variables utilisées

En voir la liste et les fonctions dans les généralités sur la gestion de fichiers de data situées devant le sous-programme F3CF.

Analyse de la syntaxe et saisie du paramètre

FABB- 20 C0 FA JSR FAC0 ce sous-programme vérifie l'existence de **FI**, la validité du NL et si le fichier est bien déjà ouvert. Puis il re-initialise les adresses 00/01, 02/03, 04/05, 06/07, 0B et C083 et retourne avec Y = #00. Ce faisant, si tout se passe bien, Z = 0 et N = 1 (fichier "**S**").
FABE- 30 DD BMI FA9D suite forcée en FA9D, car au retour du sous-programme FACO l'indicateur N est forcément à 1 sinon on aurait eu droit à une "FILE_TYPE_MISMATCH_ERROR"! C'est ce qui s'appelle de la programmation optimisée!

Ce sous-programme place #0C, #00 et #00 dans les octets de rang #03, #04 et #05 du "Channel Buffer" puis continue en FD44 car Z = 0.

Vérifie l'existence de **FI**, la validité du NL et si le fichier est déjà ouvert, re-initialise les adresses 00/01, 02/03, 04/05, 06/07, 0B et C083 et retourne avec Y = 0, Z = 0 et N = 1

(ce sous-programme FAC0-FACA, est aussi appelé par les commandes APPEND, JUMP, LTYPE et

TYPE)

FAC0- 20 7D F4 JSR F47D vérifie l'existence de **FI**, la validité du NL s'il existe et si le fichier est bien déjà ouvert
FAC3- 20 A8 F4 JSR F4A8 place l'adresse du début du "Channel Buffer" correspondant au NL en 00/01, celle du début du "Channel's own Data Buffer" en 02/03, celle du "Descriptor Buffer" en 04/05, celle du début du "General Buffer" en 06/07 et enfin, met à jour C083 (#00) et 0B (flag "**S**") puis retourne avec Y = #00
FAC6- 30 B3 BMI FA7B le b7 de 0B a été testé. Il doit être à 1 car REWIND ne marche qu'avec un fichier séquentiel. Si c'est le cas, RTS en FA7B.
FAC8- 4C E0 E0 JMP E0E0 sinon "FILE_TYPE_MISMATCH_ERROR"

Génère un "Channel Buffer" supplémentaire pour fichier "**D**", "**R**" ou "**S**"
Augmente la taille de **FI** de #121 octets, place l'offset de début de ce nouveau "Channel Buffer" dans la "Table NL", initialise 00/01, 02/03, 04/05, 06/07 place le flag "**S**/**R**/**D**" et la longueur de fiche LF ou #00 au début du "Channel Buffer" (et en C083 pour LF ou #00)

(sous-programme FACB-FB07, appelé par les commandes OPEN)

Dans le cas d'un fichier "**R**" ou "**S**", le ou les descripteur(s) sont chargés plus tard et le "Channel Buffer" est alors éventuellement étendu pour recevoir tous les descripteurs. S'il existe un "Field Buffer", il est déplacé vers le haut, sinon le pseudo-tableau **FI** est simplement étendu vers le haut. La longueur et le type de fiche ou d'enregistrement sont copiés dans le "Channel's own Data Buffer".

FACB- 48 PHA sauvegarde A puis Y sur la pile (rappel: A est la
FACC- 98 TYA longueur de fiche si "**R**" sinon A = #00; Y = #00
FACD- 48 PHA si "**R**" ou #80 si "**S**" ou #01 si "**D**")
FACE- A0 05 LDY #05 index pour lecture dans en-tête de **FI**
FAD0- B1 9E LDA (9E),Y lit HH de l'offset du début du "Field Buffer"
FAD2- D0 02 BNE FAD6 continue en FAD6 si cet octet est différent de 0. S'il est nul, il n'y a pas besoin de déplacer le "Field Buffer", car il n'existe pas, lit alors les octets de rang #02/03 (offset de la fin de **FI**).
FAD4- A0 03 LDY #03 index pour lecture dans l'en-tête de **FI**
FAD6- 88 DEY pointe le précédent (donc octet de rang #04 ou #02)
FAD7- B1 9E LDA (9E),Y à l'issue de cette partie, XY (et sa copie sur
FAD9- AA TAX la pile) contient une valeur d'offset qui se
FADA- 48 PHA trouvait soit dans les octets de rang #04/05
FADB- C8 INY et qui représente le début du "Field Buffer"
FADC- B1 9E LDA (9E),Y soit dans ceux de rang #02/03,
FADE- 48 PHA qui représentent la fin actuelle de **FI**
FADF- A8 TAY XY (et sa copie sur la pile) pointe sur le nouveau
FAE0- A9 01 LDA #01 "Channel Buffer", c'est aussi l'adresse de base
FAE2- 85 F2 STA F2 du bloc à déplacer vers le haut de AF2 octets
FAE4- A9 21 LDA #21 AF2 = #0121
FAE6- 20 25 F4 JSR F425 extension de **FI** par insertion de #121 octets au point d'offset XY, avec mise à jour de la "Table NL"
FAE9- 20 CF F3 JSR F3CF calcule l'adresse F2/F3 de la paire d'octets correspondant au n° logique dans la "Table NL" et revient avec Y = #00

FAEC- C8 INY Y = #01
FAED- 68 PLA récupère deux octets de la pile
FAEE- 91 F2 STA (F2),Y (ancien offset XY du point d'insertion)
FAF0- 88 DEY et les copie dans la "Table NL"
FAF1- 68 PLA (offset de début du "Channel Buffer"
FAF2- 91 F2 STA (F2),Y correspondant au NL)
FAF4- 20 A8 F4 JSR F4A8 place l'adresse du début du "Channel Buffer" correspondant au NL en 00/01, celle du début du "Channel's own Data Buffer" en 02/03, celle du "Descriptor Buffer" en 04/05, celle du début du "General Buffer" en 06/07 et enfin, met à jour C083 (longueur d'une fiche ou #00) et 0B (flag "**S**/**R**/**D**") puis retourne avec Y = #00. Ce sous-programme, qui est d'usage très général, effectue inutilement ces 2 dernières mises à jour, basées sur des valeurs fausses, lues dans le nouveau "Channel Buffer".
FAF7- 68 PLA récupère deux octets de la pile
FAF8- 91 00 STA (00),Y (d'abord l'ancien 0B, c'est à dire #00 si "**R**"
FAFA- 68 PLA ou #80 si "**S**" ou #01 si "**D**", puis la longueur
FAFB- C8 INY de fiche si "**D**" ou #00 dans les autres cas)
FAFC- 91 00 STA (00),Y et les place au début du "Channel Buffer"
FAFE- 8D 83 C0 STA C083 ainsi qu'en C083 pour la longueur de fiche LF
FB01- 60 RTS voilà, ça va mieux comme çà!

FB02- 4C E0 E0 JMP E0E0 "FILE_TYPE_MISMATCH_ERROR"

FB05- 4C 20 DE JMP DE20 "ILLEGAL_QUANTITY_ERROR"

Saisit le nom de fichier nom_de_fichier_non_ambigu et n° logique NL, crée si besoin et ouvre le fichier correspondant, en charge le ou les descripteurs dans le "Descriptor Buffer"

(sous-programme FB08-FB8C, appelé par la commande OPEN S ou R)

Le sous-programme FB08 initialise 0B (flag "**S**/**R**") et F9 (FTYPE), saisit et vérifie nom_de_fichier_non_ambigu présent à TXTPTR et demande la virgule qui doit suivre. Il vérifie l'existence de **FI**, le crée si besoin, saisit et vérifie le NL indiqué à TXTPTR, le copie en 0A, vérifie si le fichier n'est pas déjà ouvert (sinon "FILE_ALREADY_OPEN_ERROR"), cherche le fichier nom_de_fichier_non_ambigu dans le catalogue (empile Z = 1 si pas trouvé), le crée si besoin. Il vérifie FTYPE ("FILE_TYPE_MISMATCH_ERROR" si incompatibilité), insère un "Channel Buffer" de #121 octets à la fin du "General Buffer", place l'offset de ce "Channel Buffer" dans la "Table NL", initialise 00/01, 02/03, 04/05, 06/07, place le flag "**S**/**R**" et la longueur de fiche LF au début du "Channel Buffer" (et en C083 pour LF). Il copie "l'entrée" de catalogue dans le "Channel Buffer", initialise le nombre de descripteurs et l'adresse de chargement. Enfin, il charge le ou les descripteurs dans le "Channel Buffer" à partir de l'offset #117, c'est à dire dans le "Descriptor Buffer", copie DRIVE dans l'octet de rang #06 et dépile Z (à 1 si le fichier vient d'être créé).

Initialise 0B (flag "**S**/**R**") et F9 (FTYPE), saisit et vérifie le nom_de_fichier_non_ambigu présent à TXTPTR et demande la virgule qui doit suivre

FB08- 85 0B STA 0B #00 si "**R**" et #80 si "**S**"
FB0A- 84 F9 STY F9 #08 si "direct" et #10 si "séquentiel". C'est le type de fichier (b3 à 1 si direct ou b4 à 1 si séquentiel cf manuel page 100)

FB0C- 20 4F D4 JSR D44F XNF lit nom de fichier non-ambigu à TXTPTR et l'écrit dans BUFNOM
FB0F- 20 9E D7 JSR D79E XNOWILD recherche "?" dans BUFNOM revient avec C = 1 si pas trouvé ou génère une "WILDCARD(S)_NOT_ALLOWED_ERROR" si trouvé
FB12- 20 2C D2 JSR D22C D067/ROM exige une "," place TXTPTR sur l'octet suivant

Vérifie l'existence de **FI**, le crée si besoin, saisit et vérifie le NL indiqué à TXTPTR, le copie en 0A, vérifie si le fichier n'est pas déjà ouvert, cherche le fichier nom_de_fichier_non_ambigu dans le catalogue, le crée si besoin

FB15- 20 7F F4 JSR F47F vérifie l'existence de **FI**, la validité du NL indiqué à TXTPTR et si le fichier n'est pas déjà ouvert
FB18- 20 2D DB JSR DB2D vérifie que la disquette en place est bien une disquette SEDORIC, cherche le fichier dans le catalogue, revient avec le bon secteur de catalogue en place (de coordonnées POSNMP et POSNMS) et avec X = POSNMX, pointant sur "l'entrée" cherchée ou avec Z = 1 si le fichier n'a pas été trouvé
FB1B- 08 PHP sauvegarde les indicateurs 6502 dont Z
FB1C- D0 22 BNE FB40 continue en FB40 si le fichier a été trouvé (pas besoin d'obtenir d'information sur les fiches d'un fichier "**R**" ou de créer une entrée dans le catalogue)
FB1E- A2 00 LDX #00 si pas trouvé, il faut créer ce fichier avec les indications de longueur de fiche et du nombre de fiches à réserver, si elles existent (cas d'un fichier "**R**"). Prend X = #00 par défaut pour un fichier "**S**"
FB20- 24 0B BIT 0B teste le b7 de 0B (à 1 si fichier "**S**", sinon à 0)
FB22- 30 0A BMI FB2E continue en FB2E s'il s'agit d'un fichier "**S**"

Fichier de type "**R**":

FB24- 20 2C D2 JSR D22C D067/ROM exige une "," place TXTPTR sur l'octet suivant
FB27- 20 7F D2 JSR D27F CF17/ROM et CF09/ROM et D8CB/ROM évalue un nombre entier à TXTPTR et le retourne dans X (longueur de la fiche type)
FB2A- E0 03 CPX #03 teste si X < #03 Non documenté: la longueur de fiche LF doit être d'au moins 3 caractères et d'au plus #FF caractères!
FB2C- 90 D7 BCC FB05 si oui, "ILLEGAL_QUANTITY_ERROR"
FB2E- A9 00 LDA #00
FB30- 8D 52 C0 STA C052 force DESALO à #0000
FB33- 8D 53 C0 STA C053 (adresse fictive de début de fichier)
FB36- A8 TAY force Y à zéro et copie X dans A
FB37- 8A TXA (c'est à dire la longueur de fiche ou #00)
FB38- A6 F9 LDX F9 type de fichier (#08 si "**R**", #10 si "**S**")
FB3A- 20 00 DE JSR DE00 copie AY dans FISALO (qui représente en fait la longueur de fiche LF et dont le HH est toujours nul), X dans FTYPE, #0000 dans EXSALO, #CO dans VSALO0 (code de type SAVEU) et sauve le fichier selon BUFNOM. C'est un fichier fictif de un secteur qui est créé (de 0000 à LF).
FB3D- 20 30 DB JSR DB30 XTVNM cherche le fichier dans le catalogue, revient avec X = POSNMX, POSNMP et POSNMS (cette fois, il est trouvé!)

Vérifie FTYPE, insère un "Channel Buffer" de #121 octets à la fin du "General Buffer", place l'offset de ce "Channel Buffer" dans la "Table NL", initialise 00/01, 02/03, 04/05, 06/07, place le flag "**S**/**R**/**D**" et la

longueur de fiche LF au début du "Channel Buffer" (et en C083 pour LF)

FB40- BD 0C C3 LDA C30C,X coordonnées A = piste et Y = secteur du
FB43- BC 0D C3 LDY C30D,X descripteur principal de ce fichier
FB46- 20 5D DA JSR DA5D XPBUF1 charge dans BUF1 le descripteur AY
FB49- AD 03 C1 LDA C103 type de fichier
FB4C- C5 F9 CMP F9 teste si différent de celui indiqué par F9
FB4E- D0 B2 BNE FB02 si oui, "FILE_TYPE_MISMATCH_ERROR"
FB50- AD 06 C1 LDA C106 longueur d'une fiche si fichier à accès direct
FB53- A4 0B LDY 0B #00 si "**R**" et #80 si "**S**"
FB55- 20 CB FA JSR FACB augmente de #121 octets la taille de **FI**, place l'offset de début de ce nouveau "Channel Buffer" dans la "Table NL", initialise 00/01, 02/03, 04/05, 06/07, place le flag "**S**/**R**/**D**" et la longueur de fiche LF au début du "Channel Buffer" (et en C083 pour LF)

Copie "l'entrée" de catalogue dans le "Channel Buffer", initialise le nombre de descripteurs chargés et l'adresse de chargement (celle du "Descriptor Buffer")

FB58- A0 07 LDY #07 index pour écriture (ira de #07 à #16)
FB5A- AE 27 C0 LDX C027 POSNMX, position dans le secteur de catalogue
FB5D- BD 00 C3 LDA C300,X lit un octet d'une "entrée" du catalogue
FB60- 91 00 STA (00),Y et le copie selon Y dans le "Channel Buffer" dans les octets de rang #07 à #16 (nom de fichier etc...)
FB62- E8 INX suivant en lecture
FB63- C8 INY suivant en écriture
FB64- C0 17 CPY #17 teste si 16 octets (une ligne) ont été copiés
FB66- D0 F5 BNE FB5D sinon, reboucle en FB5D
FB68- A9 FF LDA #FF qui passera à #00 au début de la routine suivante
FB6A- A0 02 LDY #02 copie un #FF dans l'octet de rang #02
FB6C- 91 00 STA (00),Y du "Channel Buffer"
FB6E- C6 05 DEC 05 décrémente 05. Le sous-programme suivant commencera par incrémenter ces 2 octets, qui prendront respectivement la valeur #00 et la valeur initiale de 05 (04/05 vise le "Descriptor Buffer")

Charge le ou les descripteurs dans le "Descriptor Buffer" à partir de l'offset #117, copie DRIVE dans l'octet de rang #06

FB70- 20 6A F8 JSR F86A déplace d'une page vers le haut le pointeur de descripteur 04/05, incrémente l'octet de rang #02 du "Channel Buffer" (nombre de descripteurs), calcule l'offset du point d'insertion 04/05 et augmente le "Channel Buffer" de #100 octets pour recevoir le prochain descripteur
FB73- 20 5F F8 JSR F85F copie dans RWBUF l'adresse présente en 04/05, c'est à dire le début de la nouvelle zone de "Descriptor Buffer" insérée à l'offset #117 du "Channel Buffer"
FB76- 20 73 DA JSR DA73 XPRSEC lit un secteur selon DRIVE, PISTE, SECTEUR et RWBUF. Au premier tour, il s'agit des valeurs initialisées plus haut (en FB46 et en FB73), c'est à dire celles correspondant au descripteur principal
FB79- A0 00 LDY #00 pour viser le premier octet du descripteur chargé
FB7B- B1 04 LDA (04),Y lit cet octet (piste du descripteur suivant)
FB7D- 8D 01 C0 STA C001 et le copie dans PISTE
FB80- C8 INY pour viser l'octet suivant du descripteur chargé

FB81- B1 04 LDA (04),Y lit cet octet (secteur du descripteur suivant)
FB83- 8D 02 C0 STA C002 et le copie dans SECTEUR
FB86- D0 E8 BNE FB70 reboucle en FB70, pour charger un à un tous les descripteurs du fichier, si SECTEUR est différent de zéro. Cet octet est nul lorsqu'il n'y a plus de descripteur suivant.
FB88- 20 74 FA JSR FA74 copie DRIVE dans l'octet de rang #06 du "Channel Buffer"
FB8B- 28 PLP récupère les indicateurs dont Z
FB8C- 60 RTS (à 1 si le fichier a été nouvellement créé)

EXÉCUTION DE LA COMMANDE SEDORIC CLOSE

(Fichiers "**S**", "**R**" et "**D**")
(sous-programme FB8D-FBBE)

Rappel de la syntaxe

CLOSE (NL)(,NL)(,NL) etc...

Libère le ou les n° logique(s) indiqué(s) (tous les n° logiques en absence d'indication), ferme le ou les fichier(s) correspondant(s) et récupère (en théorie!) le ou les buffer(s) devenu(s) inutilisé(s). Lorsqu'un NL indiqué n'est pas attribué, on obtient une "FILE_NOT_OPEN_ERROR".

Informations non documentées

ATTENTION à la bogue: ne pas utiliser un CLOSE sans paramètre (fermeture de tous les fichiers en mémoire) en milieu de programme (**FI** devient inutilisable). Même si la fermeture a bien lieu, celle-ci s'effectue à partir du NL 99 (#63) et non 63 (#3F). Cette erreur de programmation peut être facilement réparée en remplaçant LDA #63 par LDA #3F en FBA3.

Variables utilisées

En voir la liste et les fonctions dans les généralités sur la gestion de fichiers de data situées devant le sous-programme F3CF.

Principe de la commande CLOSE

Pour chaque NL devant être "fermé", l'octet de poids fort de l'offset correspondant à ce NL dans la "Table NL" est remis à zéro pour indiquer que ce NL est désormais libre et que le fichier correspondant est fermé.

Chaque champ relatif au NL devant être "fermé" est libéré en plaçant un zéro au début de l'entrée correspondante dans le "Field Buffer", afin d'indiquer que cet emplacement est désormais disponible. Cette disponibilité sera utilisée lors d'une prochaine utilisation de la commande FIELD.

Par contre (bogue), le "Channel Buffer" correspondant au NL (#121 octets de long au minimum, plus si le fichier comporte plusieurs secteurs descripteurs) ne sera pas de nouveau utilisable, même si le NL demandé est identique), d'où une perte de place (rapide si l'on ouvre et si l'on ferme beaucoup de fichiers) pouvant occasionner une "OUT_OF_MEMORY_ERROR". Il ne semble pas trop difficile de reprogrammer SEDORIC pour que dorénavant la partie la plus haute de **FI** soit redescendue en mémoire lors de la fermeture d'un fichier.

Analyse de la syntaxe et saisie des paramètres

FB8D- F0 11 BEQ FBA0 continue en FBA0 s'il n'y a pas de paramètre

Ferme le fichier spécifié

FB8F- 20 7D F4 JSR F47D vérifie l'existence de **FI** (le crée au besoin!!!), la validité du NL s'il existe et si le fichier est bien déjà ouvert
FB92- 20 AF FB JSR FBAF ferme le fichier, c'est à dire force à 0 l'octet HH correspondant au NL 0A et supprime le "Field Buffer" éventuellement associé à ce fichier, mais hélas pas le "Channel Buffer"!!!
FB95- 20 9E D3 JSR D39E XCRGOT relit le caractère à TXTPTR (sans incrémenter TXTPTR = CHRGOT), puis le convertit en MAJUSCULE, les espaces sont sautés, Z = 1 si fin d'instruction (0 ou :), C = 0 si caractère chiffre 0 à 9 (soit #30 à #39), sinon C = 1, Y et X inchangés
FB98- F0 14 BEQ FBAE simple RTS en FBAE s'il n'y a plus de paramètre
FB9A- 20 2C D2 JSR D22C D067/ROM exige une "," place TXTPTR sur l'octet suivant
FB9D- 4C 8F FB JMP FB8F reprise forcée en FB8F

Il n'y a pas de paramètre: ferme tous les fichiers ouverts

FBA0- 20 F3 F3 JSR F3F3 vérifie l'existence de **FI** au début des tableaux et le crée s'il n'existe pas encore (ça c'est optimisé!)
FBA3- A9 63 LDA #63 dernier NL possible. Il y a ici une bogue magnifique, car le dernier NL possible n'est pas #63 mais #3F (63 en décimal!). Cette bogue conduit bien sûr à quelques problèmes dans certains cas où un CLOSE sans paramètre est utilisé en milieu de programme.
FBA5- 85 0A STA 0A place #63 dans 0A pour #64 rebouclages!
FBA7- 20 AF FB JSR FBAF ferme le fichier, c'est à dire force à 0 l'octet HH correspondant au NL 0A et supprime le "Field Buffer" éventuellement associé à ce fichier... çà doit pas être triste!
FBAA- C6 0A DEC 0A NL précédent (travaille par la fin)
FBAC- 10 F9 BPL FBA7 reboucle en FBA7 tant que 0A est positif
FBAE- 60 RTS et voilà, les dégâts sont terminés!

Ferme le fichier, c'est à dire force à 0 l'octet HH correspondant au NL 0A dans la "Table NL" et supprime le "Field Buffer" éventuellement associé à ce fichier

FBAF- 20 CF F3 JSR F3CF calcule l'adresse F2/F3 de la paire d'octets correspondant au NL dans la "Table NL" et revient avec Y = #00
FBB2- 98 TYA force A à zéro
FBB3- C8 INY et Y à 1
FBB4- 91 F2 STA (F2),Y force à zéro le deuxième octet de la paire d'octets correspondant au NL (HH de l'offset qui ne peut jamais être nul)
FBB6- 4C E6 F4 JMP F4E6 suite forcée en F4E6 pour supprimer tous les noms de champ spécifiés pour ce fichier. En fait les emplacements correspondants sont rendus disponibles pour définir de nouveaux champs, mais la mémoire occupée n'est pas libérée pour un autre usage!

FBB9- 4C E0 E0 JMP E0E0 "FILE_TYPE_MISMATCH_ERROR"

FBBC- 4C 23 DE JMP DE23 "SYNTAX_ERROR"

EXÉCUTION DE LA COMMANDE SEDORIC FIELD

(Fichiers "**R**" et "**D**")
(sous-programme FBBF-FC72)

Rappel de la syntaxe

FIELD NL,nom_de_champ TO type (,nom_de_champ TO type ,...)

Cette commande est utilisée pour les fichiers "**R**" à accès direct, afin de définir le ou les différents champs d'une fiche modèle. Pour le fichier de n° logique NL, elle associe donc à chaque nom de champ indiqué un type qui peut être alphanumérique (type $), entier (type %), réel (type !) ou octet (type O).

Pour définir les différents champs d'une fiche modèle, il faut soit indiquer ces différents champs dans la même commande FIELD, soit utiliser plusieurs commandes FIELD, mais alors chacune de ces commandes (sauf la dernière) devra se terminer par une virgule, afin d'indiquer que l'on se réfère toujours au même NL.

Dans le cas d'un champ alphanumérique, le signe $ doit alors être suivi de la longueur maximale qui ne peut évidement excéder la place restant disponible dans la fiche et en tout état de cause 254 octets. La longueur d'un champ réel est de 5 octets, celle d'un champ entier 2 octets, celle d'un champ octet 1 octet. De plus, il faut rajouter 2 octets par champ pour les informations de gestion interne. Attention donc à la place disponible dans une fiche!

Cette commande est aussi utilisable pour les "fichiers" d'accès **D**isque. La pseudo-fiche est alors en fait constituée d'un secteur de disquette soit 256 octets qui peuvent être "découpés" en "pseudo-champs" selon les modalités de longueur indiquées ci-dessus. Mais dans ce cas, les champs sont définis sans la séparation de 2 octets entre eux.

Le fichier doit évidemment être ouvert, sinon "FILE_NOT_OPEN_ERROR" et être du bon type, sinon "FILE_TYPE_MISMATCH_ERROR".

Principe de la commande FIELD

Les informations concernant les champs d'une fiche modèle sont gardées dans le "Field Buffer" et la place totale occupée par tous ces champs est calculée lorsque de nouveaux champs sont définis pour la fiche modèle du fichier. Si la commande FIELD s'achève par une virgule, la prochaine commande FIELD, se référant au même NL, révisera la place totale occupée par les champs. Si ce nombre devient plus grand que celui spécifié pour ce NL, on a une "FIELD_OVERFLOW_ERROR". En absence de virgule terminale, la commande FIELD suivante, pour le même NL, remet ce compteur à zéro (re-definition de la fiche modèle).

Non ou mal documenté

1) Attention le "TO" de la commande FIELD, comme celui des autres commandes SEDORIC doit être en MAJUSCULE, sinon il n'est pas reconnu. Je rappelle que l'usage des minuscules n'est plus supporté par la version 3.0 de SEDORIC (trop de bogues).

2) Il est possible d'utiliser n'importe quoi (sauf les deux points) comme délimiteur entre les définitions successives des noms de champ et pas seulement une virgule! (bogue située en FC48 et qui pourrait être

réparée par un JSR D22C).

3) Il est possible d'indexer un nom de champ, comme s'il s'agissait d'un élément de tableau, NOM(2) par exemple, sans que cela crée pour autant les autres éléments du tableau: NOM(0) et NOM(1). En effet, il s'agit seulement d'un système de notation qui permet de compléter le nom du champ (5 caractères maximum) et non de DIMensionner un vrai tableau. Dans mon exemple, FIELD 1, NOM(2) TO $8 ne défini que le seul champ NOM(2) comme étant un champ alphanumérique de 8 caractères de long. Pour définir aussi NOM(0) et NOM(1), il faudrait taper:

FIELD 1, NOM(0) TO $8, NOM(1) TO $8, NOM(2) TO $8 ou plus rapidement:
FOR I=0 TO 2:FIELD 1, NOM(I) TO $8,:NEXT I

4) Il semblerait, au vu du code en FC2E, que le même nom de champ puisse être défini deux fois pour deux champs différents au sein d'une même fiche. En fait, ceci a pour conséquence d'effacer les informations de la première définition et les data contenus dans le premier champ qui deviennent donc inaccessibles. Ce phénomène semble voulu, mais (bogue potentielle) il serait peut-être mieux de reprogrammer cette partie de manière à prévenir ("DUPLICATED FIELD").

5) Autre bogue: Il est également possible d'avoir le même "nom_de_champ(index)" dans plusieurs fichiers. Cependant, peut-être à cause d'une bogue de SEDORIC, lors d'un transfert de ou vers un champ, grâce aux commandes LSET ou RSET, seul le premier "nom_de_champ(index)" est accessible. Ceci est dû au fait que SEDORIC ne compare pas le NL pour lequel le "nom_de_champ(index)" a été définis avec le NL courant. La vérification du NL et du "nom_de_champ(index)" peut être obtenue en changeant un octet de SEDORIC: remplacer BVC F548 (50 20) par BVC F54B (50 23) en F526.

En temps normal, il n'y a pas de vérification du NL: il en résulte que la première occurrence du "nom_de_champ(index)" rencontrée est acceptée comme la bonne, sans tenir compte du NL. Si le même "nom_de_champ(index)" est utilisé par plusieurs NL, alors l'adressage trouvé risque de ne pas être le bon. Après modification de SEDORIC en F526, le NL courant sera utilisé pour la vérification et les deux instructions en F5CE et F5D1 seront redondantes.

Cependant, avec le code en vigueur (SEDORIC non modifié), si tous les "noms_de_champ(index)" sont différents, on peut obtenir sans problème le champ de la fiche courante quel que soit le fichier, à n'importe quel moment, du moment que le fichier soit ouvert et que la bonne fiche soit en place. Il en sera de même pour les commandes LSET et RSET.

Attention, avec les fichiers "**D**", on ne peut hélas pas exploiter l'ensemble d'un secteur (256 octets) comme un seul champ alphanumérique, car la longueur d'un champ alphanumérique BASIC ne peut dépasser 255 octets.

Variables utilisées

En voir la liste et les fonctions dans les généralités sur la gestion de fichiers de data situées devant le sous-programme F3CF.

Analyse de syntaxe et saisie des paramètres

FBBF- 20 7D F4 JSR F47D vérifie l'existence de **FI**, la validité du NL s'il existe et si le

fichier est bien déjà ouvert
FBC2- 20 2C D2 JSR D22C D067/ROM exige une "," place TXTPTR sur l'octet suivant
FBC5- 20 A8 F4 JSR F4A8 place l'adresse du début du "Channel Buffer" correspondant au NL en 00/01, celle du début du "Channel's own Data Buffer" en 02/03, celle du "Descriptor Buffer" en 04/05, celle du début du "General Buffer" en 06/07 et enfin, met à jour C083 (longueur d'une fiche ou #00) et 0B (flag "**R/D**") puis retourne avec Y = #00 et N = 1 si "**S**"
FBC8- 30 EF BMI FBB9 "SYNTAX_ERROR" car fichier de type "**S**" ne sont pas autorisés
FBCA- AD 80 C0 LDA C080 sauvegarde du NL de la dernière commande FIELD
FBCD- C5 0A CMP 0A NL du fichier courant
FBCF- F0 05 BEQ FBD6 continue en FBD6 si c'est le même (continue d'incrémenter le même compteur de longueur totale des champs en C081)

Mise à zéro d'un nouveau totalisateur de la longueur des champs

FBD1- A9 00 LDA #00
FBD3- 8D 81 C0 STA C081 initialise le compteur de longueur totale

Suite de l'analyse de syntaxe et paramètres

FBD6- 20 40 F6 JSR F640 vide le "General Field Buffer" en C076/CC07F, puis décode le nom de champ présent à TXTPTR (5 caractères significatifs, les suivants sont ignorés) et s'il s'agit d'un pseudo-tableau, lit l'index de cet élément de pseudo-tableau (cet index peut commencer à 0 comme pour DIM et sa valeur maximale est stockée en C07B). Enfin met le NL courant en C07C.
FBD9- A9 C3 LDA #C3 token BASIC de TO (il doit être en MAJUSCULES)
FBDB- 20 2E D2 JSR D22E JSR D067/ROM puis D3A1/RAM overlay demande "TO" à TXTPTR, lit le caractère suivant et le convertit éventuellement en MAJUSCULE
FBDE- F0 DC BEQ FBBC "SYNTAX_ERROR" si fin de commande
FBE0- 48 PHA sauve le type de champ qui vient d'être lu à TXTPTR
FBE1- 20 98 D3 JSR D398 XCRGET incrémente TXTPTR, lit un caractère (CHRGET), les espaces sont sautés, le met dans A, le convertit en MAJUSCULE, Z = 1 si fin d'instruction (0 ou :), C = 0 si chiffre 0 à 9 (soit #30 à #39), sinon C =1. Y et X inchangés (sert à positionner TXTPTR sur le caractère suivant)
FBE4- 68 PLA récupère le type de champ
FBE5- A0 00 LDY #00 par défaut, code de type réel
FBE7- A2 05 LDX #05 par défaut, longueur de champ réel
FBE9- C9 C0 CMP #C0 le type lu est-il celui de champ réel? (token "!")
FBEB- F0 1A BEQ FC07 si oui, continue en FC07 avec les bons paramètres
FBED- A2 02 LDX #02 sinon, longueur de champ entier
FBEF- C8 INY et code de champ entier
FBF0- C9 25 CMP #25 le type lu est-il celui de champ entier? ("%")
FBF2- F0 13 BEQ FC07 si oui, continue en FC07 avec les bons paramètres
FBF4- CA DEX longueur de champ pour type octet (c'est à dire #01)
FBF5- A0 40 LDY #40 code pour type octet
FBF7- C9 4F CMP #4F le type lu est-il celui de champ octet? ("O")
FBF9- F0 0C BEQ FC07 si oui, continue en FC07 avec les bons paramètres
FBFB- C9 24 CMP #24 le type lu est-il celui de champ alphanumérique? ("$")
FBFD- D0 BD BNE FBBC sinon, "SYNTAX_ERROR": il n'y a pas d'autre type

Cas particulier d'un champ de type alphanumérique

FBFF- 20 7F D2 JSR D27F CF17/ROM et CF09/ROM et D8CB/ROM évalue un nombre entier à TXTPTR et le retourne dans X (longueur du champ alphanumérique)
FC02- 8A TXA copie la longueur évaluée dans A
FC03- F0 4F BEQ FC54 "ILLEGAL QUANTITY" si longueur nulle
FC05- A0 80 LDY #80 code pour type alphanumérique

FC07- 8C 7F C0 STY C07F sauve le type de champ
FC0A- 8E 7E C0 STX C07E sauve la longueur du champ
FC0D- AD 81 C0 LDA C081 longueur totale des champs jusqu'ici
FC10- A4 0A LDY 0A NL du fichier courant
FC12- 8D 7D C0 STA C07D longueur totale des champs jusqu'ici
FC15- 8C 7C C0 STY C07C NL du fichier courant
FC18- 18 CLC prépare les additions en FC1D et FC22
FC19- A6 0B LDX 0B type "**S/R/D**" du fichier courant
FC1B- D0 05 BNE FC22 saute les 2 instructions suivantes si fichier de type "**D**"

Cas d'un fichier de type "**R**" à accès direct

FC1D- 69 02 ADC #02 ajoute déjà les 2 octets séparateurs
FC1F- 20 57 FC JSR FC57 vérifie si la longueur totale des champs jusqu'ici est compatible avec la longueur de fiche, C = 0 au retour

Calcule et vérifie la longueur totale des champs

FC22- 6D 7E C0 ADC C07E ajoute la longueur du champ
FC25- 20 57 FC JSR FC57 vérifie si la longueur totale des champs jusqu'ici est compatible avec la longueur de fiche, C = 0 au retour
FC28- 8D 81 C0 STA C081 sauve la longueur totale des champs jusqu'ici

Met à jour l'emplacement correspondant du "Field Buffer"

FC2B- 20 EC F4 JSR F4EC vérifie si le nom de champ existe déjà pour ce fichier
FC2E- B0 03 BCS FC33 si oui, saute l'instruction suivante (utilise le même emplacement)
FC30- 20 EF F4 JSR F4EF sinon, réserve une place disponible pour un nouveau nom de champ associé au fichier courant
FC33- A0 09 LDY #09 pour copier les #10 octets de nom de champ etc...
FC35- B9 76 C0 LDA C076,Y du "General Field Buffer"
FC38- 91 F4 STA (F4),Y dans le "Field Buffer"
FC3A- 88 DEY en travaillant par la fin
FC3B- 10 F8 BPL FC35 reboucle en FC35 tant qu'il en reste à copier

Y a t-il encore des paramètres?

FC3D- A2 00 LDX #00 pour éventuelle mise à zéro de la longueur totale
FC3F- 20 9E D3 JSR D39E XCRGOT relit le caractère à TXTPTR (sans incrémenter TXTPTR = CHRGOT), puis le convertit en MAJUSCULE, les espaces sont sautés, Z = 1 si fin

d'instruction (0 ou :), C = 0 si caractère chiffre 0 à 9 (soit #30 à #39), sinon C = 1, Y et X inchangés
FC42- D0 04 BNE FC48 saute les 2 instructions suivantes si pas fini
FC44- 8E 81 C0 STX C081 il n'y a plus de paramètre et notamment pas de virgule finale, remet donc la longueur totale à zéro pour la prochaine
FC47- 60 RTS commande FIELD et retourne
FC48- 20 2E D2 JSR D22E JSR D067/ROM puis D3A1/RAM overlay demande à TXTPTR un caractère ou un code identique à A, lit le caractère suivant et le convertit éventuellement en MAJUSCULE. Cela veut dire que le caractère lu en FC3F doit être égal à lui même! En fait saute le séparateur (normalement une virgule). Il est donc possible d'utiliser n'importe quoi comme délimiteur... (encore une bogue)
FC4B- D0 89 BNE FBD6 reboucle en FBD6 s'il reste des paramètres

La commande se termine par une virgule (ou autre délimiteur!) il faut permettre de reprendre là où on en est lors de la prochaine commande FIELD concernant le même fichier

FC4D- A5 0A LDA 0A NL du fichier courant
FC4F- 8D 80 C0 STA C080 sauvegarde du NL de la dernière commande FIELD
FC52- 18 CLC force C = 0
FC53- 60 RTS et retourne
FC54- 4C 20 DE JMP DE20 "ILLEGAL_QUANTITY_ERROR"

Vérifie si la longueur totale des champs atteinte jusqu'ici est compatible avec la longueur de fiche, C = 0 au retour

Ce sous-programme est un bel exemple de programmation approximative: les instructions FC57 à FC5F et FC67 à FC6D sont inutiles, car la valeur spécifiée en C083 a déjà été vérifiée et ne peut dépasser 255 pour un fichier de type "**R**" et 256 pour un fichier de type "**D**". Il y a donc ici 16 octets potentiellement disponibles!

FC57- F0 10 BEQ FC69 si le résultat de l'addition précédente atteint #00 (en réalité #100 soit 256) continue en FC69 pour vérifier si OK (fichier "**D**")
FC59- B0 13 BCS FC6E si le résultat de l'addition précédente dépasse 255 pour un fichier "**R**", génère une "FIELD_OVERFLOW_ERROR"
FC5B- AE 83 C0 LDX C083 longueur de fiche
FC5E- F0 F2 BEQ FC52 si la longueur de fiche spécifiée est #00 (en réalité #100 soit 256, cas d'un fichier "**D**") continue en FC52
FC60- CD 83 C0 CMP C083 compare la longueur totale atteinte jusqu'ici à la longueur de fiche spécifiée qui représente le maximum autorisé
FC63- F0 ED BEQ FC52 pas encore trop long, termine en FC52
FC65- 90 EB BCC FC52 en dessous de la limite, termine en FC52
FC67- B0 05 BCS FC6E déjà trop long, "FIELD_OVERFLOW_ERROR" en FC6E
FC69- AE 83 C0 LDX C083 longueur de fiche
FC6C- F0 E4 BEQ FC52 si la longueur de fiche spécifiée est #00 (en réalité #100 soit 256, cas d'un fichier "**D**") continue en FC52
FC6E- A2 11 LDX #11 pour "FIELD_OVERFLOW_ERROR"
FC70- 4C 7E D6 JMP D67E incrémente X et traite l'erreur n° X

EXÉCUTION DE LA COMMANDE SEDORIC LSET

(Fichiers "**R**" et "**D**")
(sous-programme FC73-FC75 et suite à RSET)

Rappel de la syntaxe

LSET nom_de_champ(index) < expression

Cette commande transfère une expression ou une variable dans le champ indiqué. Le fichier doit évidemment être ouvert, sinon "FILE_NOT_OPEN_ERROR" et être du bon type, sinon "FILE_TYPE_MISMATCH_ERROR". L'expression doit correspondre au type du champ indiqué (alphanumérique, réel, entier ou octet), sinon "TYPE_MISMATCH_ERROR". Les valeurs alphanumériques sont placées à gauche dans le champ et justifiées à droite avec des espaces (ou tronquées si trop longues). Pour les autres types de champs, la longueur de l'objet étant définie par le type de l'objet, LSET et RSET donnent le même résultat.

Non documenté

LSET signifie "**L**eft **SET**", c'est à dire "place à gauche". Il faut utiliser la commande TAKE pour mettre à jour le NL et le n° de fiche. Le nom_de_champ(index), qui doit avoir été défini pour le NL courant (sinon "UNKNOW'N_FIELD_NAME_ERROR"), est décodé dans le "General Field Buffer", qui se trouve en C076/C07F. Lors du transfert d'une expression alphanumérique vers un champ, celui-ci est d'abord effacé avec des espaces puis l'expression est copiée dans le champ de gauche à droite, en limitant le nombre de caractères copiés à la longueur du champ.

Variables utilisées

En voir la liste et les fonctions dans les généralités sur la gestion de fichiers de data situées devant le sous-programme F3CF.

Entrée de la commande LSET

FC73-	18	CLC flag pour LSET
FC74-	24 38	BIT 38 continue en FC76

EXÉCUTION DE LA COMMANDE SEDORIC RSET

(Fichiers "**R**" et "**D**")
(sous-programme FC75-FD0D)

Rappel de la syntaxe

RSET nom_de_champ(index) < expression

Cette commande transfère une expression ou une variable dans le champ indiqué. Le fichier doit évidemment être ouvert, sinon "FILE_NOT_OPEN_ERROR" et être du bon type, sinon "FILE_TYPE_MISMATCH_ERROR". L'expression doit correspondre au type du champ indiqué

(alphanumérique, réel, entier ou octet), sinon "TYPE_MISMATCH_ERROR". Les valeurs alphanumériques sont placées à droite dans le champ et justifiées à gauche avec des espaces (ou tronquées à gauche si trop longues). Pour les autres types de champs, la longueur de l'objet étant définie par le type de champ, LSET et RSET donnent le même résultat.

Non documenté

RSET signifie "**R**ight **SET**", c'est à dire "place à droite". Il faut utiliser la commande TAKE pour mettre à jour le NL et le n° de fiche. Le nom_de_champ(index), qui doit avoir été défini pour le NL courant (sinon "UNKNOW'N_FIELD_NAME_ERROR"), est décodé dans le "General Field Buffer", qui se trouve en C076/C07F. Lors du transfert d'une expression alphanumérique vers un champ, celui-ci est d'abord effacé avec des espaces puis l'expression est copiée dans le champ de droite à gauche, en limitant le nombre de caractères copiés à la longueur du champ.

Variables utilisées

En voir la liste et les fonctions dans les généralités sur la gestion de fichiers de data situées devant le sous-programme F3CF.

Entrée de la commande RSET

FC75- 38 SEC flag pour RSET

Suite commune pour les commandes LSET et RSET

FC76- 08 PHP sauvegarde C (flag "LSET/RSET")
FC77- 20 F3 F3 JSR F3F3 Vérifie l'existence de **FI** au début des tableaux et le crée s'il n'existe pas encore (il est bien temps !!!)
FC7A- 20 40 F6 JSR F640 vide le "General Field Buffer" (10 octets de C076 à C07F), puis décode le nom de champ présent à TXTPTR, le copie en C076/C07A (5 caractères significatifs maximum), s'il s'agit d'un pseudo-tableau, copie l'index de cet élément en C07B) et enfin copie le NL courant en C07C
FC7D- A9 D5 LDA #D5 token de "<"
FC7F- 20 2E D2 JSR D22E JSR D067/ROM puis D3A1/RAM overlay demande "<" à TXTPTR, lit le caractère suivant et le convertit éventuellement en MAJUSCULE
FC82- 20 24 D2 JSR D224 JSR CF17/ROM évalue une expression à TXTPTR, retourne avec une valeur numérique dans ACC1 (et #00 dans 28) ou l'adresse d'une chaîne dans D3/D4 (et #FF dans 28) et A, N et Z positionnés selon l'exposant (résultat numérique) ou la longueur de chaîne
FC85- 20 E9 F4 JSR F4E9 localise un nom de champ particulier associé au fichier courant dans le "Field Buffer" ("UNKNOW'N_FIELD_NAME_ERROR" s'il n'existe pas), copie "l'entrée" correspondante dans le "General Field Buffer"
FC88- 20 7A F6 JSR F67A compare le type du champ et celui de l'expression
FC8B- 20 A8 F4 JSR F4A8 place l'adresse du début du "Channel Buffer" correspondant au NL en 00/01, celle du début du "Channel's own Data Buffer" en 02/03, celle du "Descriptor Buffer" en 04/05, celle du début du "General Buffer" en 06/07 et enfin, met à jour C083 (longueur d'une fiche ou #00) et 0B (flag "**R**/**D**") puis retourne avec Y = #00
FC8E- AD 7C C0 LDA C07C NL du champ courant
FC91- 85 0A STA 0A n°logique courant
FC93- AC 7D C0 LDY C07D index du début du champ dans l'enregistrement

```
FC96-    AE 7E C0    LDX C07E   longueur du champ
FC99-    A5 0B       LDA 0B     type "R/D" du fichier
FC9B-    D0 0C       BNE FCA9   si fichier de type "D", continue en FCA9
FC9D-    28          PLP    récupère C (flag "LSET/RSET")
```

Copie le champ directement dans le "Channel's own Data Buffer" (pour un fichier "**R**"), copie l'enregistrement directement dans le "General Buffer" (pour un fichier "**S**")

(ce sous-programme est aussi appelé de FA47 par la commande PUT).

Dans les deux cas ("**R**" ou "**S**"), bien que les buffers impliqués soient différents, l'information nouvelle est copiée dans un tampon qui est le "Channel's own Data Buffer" pour un fichier "**R**" et le "General Buffer" pour un fichier "**S**".

Pour un "fichier" de type "**D**", le sous-programme commence en FCA9, le tampon est le "Channel's own Data Buffer" qui constitue en lui-même un secteur complet.

Ce tampon est pointé par 02/03 qui contient soit l'adresse du "Channel's own Data Buffer" ("**R**/**D**") (02/03 inchangée), soit l'adresse du "General Buffer" ("**S**") (préalablement recopiée de 06/07). NB: Y = #00 en entrée et pointe au début du tampon.

```
FC9E-    08          PHP    sauvegarde C (LSET /RSET) pour usage ultérieur
FC9F-    AD 7F C0    LDA C07F   type d'enregistrement ou de champ
FCA2-    91 02       STA (02),Y stocké au début du tampon
FCA4-    C8          INY    Y = #01
FCA5-    8A          TXA    longueur de l'enregistrement ou du champ
FCA6-    91 02       STA (02),Y stocké à la suite
FCA8-    C8          INY    Y = #02
```

Entrée pour fichier de type "**D**":

FCA9- 98 TYA vise le début réel du secteur (Y = #00). Dans le cas d'un "fichier" de type "**D**", il n'y a pas d'indication de type et de longueur au début du tampon qui commence directement par les data réels

```
FCAA-    20 DC F4    JSR F4DC   ajoute A au contenu de 02/03 qui vise donc directement les data
impliqués
FCAD-    28          PLP    récupère C (flag LSET /RSET pour chaîne seulement)
FCAE-    A0 00       LDY #00    nouvel index visant le début des data
FCB0-    AD 7F C0    LDA C07F   teste le type d'enregistrement ou de champ
FCB3-    30 22       BMI FCD7   continue en FCD7 si c'est une chaîne
FCB5-    F0 19       BEQ FCD0   continue en FCD0 si c'est un nombre réel
                                continue en FCB7 si c'est un nombre entier
```

Enregistrement de type "octet" ou "entier"

```
FCB7-    20 4C D2    JSR D24C   JSR D2A9/ROM nombre en ACC1 -> #D4-#D3 (entier)
FCBA-    A0 00       LDY #00    encore! (index visant le début des data)
FCBC-    A5 D4       LDA D4     LL du nombre entier ou "octet"
```

FCBE- 91 02 STA (02),Y stocké dans le tampon
FCC0- 2C 7F C0 BIT C07F teste le b6 du type d'enregistrement ou de champ
FCC3- 50 05 BVC FCCA continue en FCCA si c'est un "entier"
FCC5- A5 D3 LDA D3 c'est un "octet" dont le HH doit être nul
FCC7- D0 8B BNE FC54 sinon "ILLEGAL_QUANTITY_ERROR"
FCC9- 60 RTS tout fini bien pour le type "octet"
FCCA- C8 INY vise l'octet suivant du tampon
FCCB- A5 D3 LDA D3 HH du nombre entier
FCCD- 91 02 STA (02),Y stocké dans le tampon
FCCF- 60 RTS tout fini bien pour le type "entier"

Enregistrement de type "réel"

FCD0- A6 02 LDX 02 adresse du début du tampon
FCD2- A4 03 LDY 03 JSR DEAD/ROM recopie les 5 octets de ACC1 de
FCD4- 4C C2 D2 JMP D2C2 l'adresse XY à l'adresse XY + 4 et retourne (tout fini bien pour le type "réel")

Enregistrement de type "chaîne"

FCD7- 08 PHP sauvegarde C (LSET /RSET) pour usage ultérieur
FCD8- 20 74 D2 JSR D274 JSR D7D0 et CF09/ROM vérifie si l'expression évaluée à TXTPTR est bien alphanumérique, retourne l'adresse de la chaîne dans XY et 91/92 ainsi que sa longueur dans A
FCDB- 85 D0 STA D0 longueur de la chaîne
FCDD- AC 7E C0 LDY C07E récupère la longueur de champ spécifiée dans la commande FIELD pour un ficher "**R**" ou la longueur d'enregistrement déterminée lors du premier PUT pour un fichier "**S**". Cette valeur est indispensable lors de la mise à jour d'un champ ou d'un enregistrement existant afin de ne pas écraser le champ ou l'enregistrement suivant
FCE0- F0 07 BEQ FCE9 continue en FCE9 s'il s'agit d'un "fichier" de type "**D**" pour lequel la longueur d'enregistrement est nulle (en fait automatiquement 256 octets, c'est à dire une "page" ou un secteur)
FCE2- A9 20 LDA #20 "espace" pour vider le champ ou l'enregistrement
FCE4- 88 DEY copie des "espaces" dans tout le champ ou
FCE5- 91 02 STA (02),Y l'enregistrement, selon la longueur voulue
FCE7- D0 FB BNE FCE4 reboucle en FCE4 tant qu'il en reste
FCE9- 28 PLP récupère C (flag LSET/RSET... patience récompensée)
FCEA- B0 0E BCS FCFA continue en FCFA si RSET
FCEC- EA NOP
FCED- EA NOP

LSET en cours

FCEE- C4 D0 CPY D0 teste si l'index a atteint la longueur de chaîne
FCF0- B0 07 BCS FCF9 si oui, termine en FCF9
FCF2- B1 91 LDA (91),Y sinon, lit un octet de la chaîne
FCF4- 91 02 STA (02),Y et le copie dans le tampon
FCF6- C8 INY vise l'octet suivant

```
FCF7-     D0 F5       BNE FCEE   rebouclage forcé en FCEE
FCF9-     60          RTS   tout fini bien pour le type "chaîne" LSET
```

RSET en cours

La copie va se faire par la fin. D0 (longueur de la chaîne) servira à compter le nombre d'octets restant à copier.

```
FCFA-     A4 D0       LDY D0     teste s'il reste des octets à copier (et valide Y)
FCFC-     F0 0F       BEQ FD0D   sinon, termine en FD0D
FCFE-     88          DEY  l'index de lecture variera de D0-1 à 0
FCFF-     C6 D0       DEC D0     un octet de copié (par anticipation!)
FD01-     B1 91       LDA (91),Y lit un octet par la fin de la chaîne
FD03-     CE 7E C0    DEC C07E   longueur de champ ou d'enregistrement
FD06-     AC 7E C0    LDY C07E   index d'écriture dans le tampon
FD09-     91 02       STA (02),Y écrit l'octet dans le tampon (par la fin)
FD0B-     D0 ED       BNE FCFA   reboucle en FCFA, tant qu'il en reste à copier
FD0D-     60          RTS   tout fini bien pour le type "chaîne" RSET
```

NB: le test est effectué deux fois, en FD0B et en FCFC afin d'être plus sûr!!!

Réinitialise 00/01, 02/03, 04/05 et 06/07, sauvegarde les octets de rang #03, #04 et #05 du "Channel Buffer" en C086, C087, C088 et teste si la fin du fichier est atteinte (Z = 1)

(sous-programme FD0E-FD29, appelé par les commandes &, JUMP, LTYPE, PUT et TYPE)
(pour fichiers de type **S**équentiel seulement)

FD0E- 20 A8 F4 JSR F4A8 place l'adresse du début du "Channel Buffer" correspondant au NL en 00/01, celle du début du "Channel's own Data Buffer" en 02/03, celle du "Descriptor Buffer" en 04/05, celle du début du "General Buffer" en 06/07 et enfin, met à jour C083 (longueur d'une fiche) et 0B (flag "**S**") puis retourne avec Y = #00

```
FD11-     A0 03       LDY #03    vise l'octet de rang #03 du "Channel Buffer"
FD13-     B1 00       LDA (00),Y lit l'index de lecture dans le descripteur courant
FD15-     8D 86 C0    STA C086   et le sauve en C086
FD18-     C8          INY  vise l'octet de rang #04 du "Channel Buffer"
FD19-     B1 00       LDA (00),Y lit le n° du descripteur courant
FD1B-     8D 87 C0    STA C087   et le sauve en C087
FD1E-     C8          INY  vise l'octet de rang #05 du "Channel Buffer"
FD1F-     B1 00       LDA (00),Y lit l'index dans le "Channel's own Data Buffer"
FD21-     8D 88 C0    STA C088   et le sauve en C088
FD24-     A8          TAY  ce dernier sert d'index pour lire un octet
FD25-     B1 02       LDA (02),Y dans le "Channel's own Data Buffer"
FD27-     C9 FF       CMP #FF    l'octet lu est comparé avec #FF
FD29-     60          RTS   retourne avec Z = 1 si la fin du fichier est atteinte
```

Réinitialise 00/01, 02/03, 04/05, 06/07, 0B et C083, restaure les octets C088, C087 et C086 dans les octets de rang #05, #04 et #03 du "Channel Buffer", l'octet lu en C086 est comparé avant écriture avec l'octet qu'il écrasera dans cette zone, RTS en FD29 s'ils sont identiques, sinon charge un secteur du fichier ouvert

(sous-programme FD2A-FD72, appelé par les commandes BUILD et PUT)

FD2A- 20 A8 F4 JSR F4A8 place l'adresse du début du "Channel Buffer" correspondant au NL en 00/01, celle du début du "Channel's own Data Buffer" en 02/03, celle du "Descriptor Buffer" en 04/05, celle du début du "General Buffer" en 06/07 et enfin, met à jour C083 (longueur d'une fiche ou #00) et 0B (flag "**S**/**R**/**D**") puis retourne avec Y = #00
FD2D- A0 05 LDY #05 dans le "Channel Buffer", vise l'octet de rang #05 (index dans le tampon qui vient d'être lu sur la disquette)
FD2F- AD 88 C0 LDA C088 lit la valeur antérieure de cet index
FD32- 91 00 STA (00),Y et la remet en place
FD34- 88 DEY vise l'octet de rang #04 du "Channel Buffer" (n° du descripteur courant)
FD35- AD 87 C0 LDA C087 lit la valeur antérieure de ce n° de descripteur
FD38- 91 00 STA (00),Y et la remet en place
FD3A- 88 DEY vise l'octet de rang #04 du "Channel Buffer" (index de lecture dans la liste des coordonnées du descripteur courant)
FD3B- AD 86 C0 LDA C086 lit la valeur antérieure de cet index et la compare
FD3E- D1 00 CMP (00),Y avant écriture avec l'octet qu'il écrasera dans le
FD40- 91 00 STA (00),Y "Channel Buffer"
FD42- F0 E5 BEQ FD29 RTS en FD29 s'ils sont identiques (il n'y a pas besoin de relire ce secteur de la disquette), sinon...

Charge un secteur de data du fichier qui se trouve sur la disquette dans le "Channel's own Data Buffer"

FD44- 18 CLC C = 0 pour charger un secteur de la disquette
FD45- 24 38 BIT 38 continue en FD47

Sauve sur la disquette le secteur qui est présent dans le "Channel's own Data Buffer"

FD46- 38 SEC C = 1 pour sauver un secteur sur la disquette
FD47- 08 PHP sauvegarde les indicateurs 6502 dont C
FD48- 20 A8 F4 JSR F4A8 place l'adresse du début du "Channel Buffer" correspondant au NL en 00/01, celle du début du "Channel's own Data Buffer" en 02/03, celle du "Descriptor Buffer" en 04/05, celle du début du "General Buffer" en 06/07 et enfin, met à jour C083 (longueur d'une fiche ou #00) et 0B (flag "**S**/**R**/**D**") puis retourne avec Y = #00
FD4B- A5 02 LDA 02 AY = adresse de début du "Channel's own Data Buffer"
FD4D- A4 03 LDY 03 correspondant au NL traité
FD4F- 8D 03 C0 STA C003 mise à jour de RWBUF avec
FD52- 8C 04 C0 STY C004 cette adresse
FD55- A0 04 LDY #04 vise l'octet de rang #04 du "Channel Buffer"
FD57- B1 00 LDA (00),Y lit le n° du descripteur courant
FD59- 18 CLC prépare une addition
FD5A- 65 05 ADC 05 calcule HH de l'adresse du début du descripteur
FD5C- 85 05 STA 05 = HH de l'adresse du début du "Descriptor Buffer"
FD5E- 88 DEY + n° du descripteur courant
FD5F- B1 00 LDA (00),Y lit l'index de lecture dans le descripteur courant
FD61- A8 TAY pointe dans la liste des coordonnées piste/secteur

```
FD62-    B1 04      LDA (04),Y  lit un n° de PISTE dans cette liste de coordonnées
FD64-    48         PHA  et l'empile
FD65-    C8         INY  index octet suivant
FD66-    B1 04      LDA (04),Y  lit maintenant le n° de SECTEUR qui suit
FD68-    A8         TAY  et le garde dans Y
FD69-    68         PLA  récupère le premier (AY = cordonnées piste/secteur)
FD6A-    28         PLP  récupère les indicateurs dont C initial
FD6B-    90 03      BCC FD70  saute l'instruction suivante si C = 0
FD6D-    4C 9E DA   JMP DA9E  XSAY sauve secteur RWBUF à la piste A, secteur Y

FD70-    4C 6D DA   JMP DA6D  XPAY charge dans RWBUF le secteur Y de piste A
```

Appelé par PUT pour un fichier de type **S**équentiel, écrit un octet dans le "Channel's own Data Buffer" et avance dans le buffer (sauve le buffer sur la disquette et charge le secteur de data suivant si nécessaire)

(sous-programme FD73-FDD8, appelé par les commandes BUILD et PUT, et par le sous-programme FE38)

```
FD73-    20 CC FD   JSR FDCC  Y = index dans le "Channel's own Data Buffer"
FD76-    91 02      STA (02),Y  écrit A dans le "Channel's own Data Buffer"
FD78-    38         SEC  C = 1 flag "PUT" et
FD79-    24 18      BIT 18  continue en FD7B (avance dans le buffer, sauve le buffer sur
la disquette et charge le secteur de data suivant si nécessaire)
```

Appelé par TAKE pour un fichier de type **S**équentiel, avance dans le "Channel's own Data Buffer", charge un second secteur si nécessaire, et lit un octet

```
FD7A-    18         CLC  C = 0 flag "TAKE"
FD7B-    20 CC FD   JSR FDCC  lit Y = index dans le "Channel's own Data Buffer"
FD7E-    C8         INY  indexe l'octet suivant
FD7F-    D0 42      BNE FDC3  continue en FDC3 si fin du buffer pas atteinte
FD81-    90 21      BCC FDA4  continue en FDA4 si C = 0 (sous-programme appelé par TAKE)
```

Pour PUT: écriture puis lecture du secteur de data suivant

```
FD83-    20 46 FD   JSR FD46  sauve sur la disquette le secteur du fichier qui est présent dans
le "Channel's own Data Buffer"
FD86-    A0 02      LDY #02  vise l'octet de rang #02 du "Channel Buffer"
FD88-    B1 00      LDA (00),Y  lit le n° du dernier descripteur
FD8A-    A0 04      LDY #04  vise l'octet de rang #04 du "Channel Buffer"
FD8C-    D1 00      CMP (00),Y  compare avec le n° du descripteur courant
FD8E-    D0 14      BNE FDA4  continue en FDA4, car le dernier descripteur n'est pas encore
atteint, c'est à dire, pas besoin d'en créer un nouveau
```

Dernier descripteur atteint, vérifie si un nouveau descripteur est nécessaire

```
FD90-    88         DEY  vise l'octet de rang #03 du "Channel Buffer"
FD91-    B1 00      LDA (00),Y  lit l'index dans le descripteur courant
```

FD93- A8 TAY
FD94- C8 INY
FD95- C8 INY teste s'il y a encore place pour 2 octets
FD96- F0 05 BEQ FD9D si la fin du descripteur est atteinte, il faut créer un secteur de data supplémentaire et aussi une nouvelle page de descripteur, continue en FD9D, sinon...
FD98- C8 INY vise un éventuel n° de secteur
FD99- B1 04 LDA (04),Y lit ce n° de secteur potentiel
FD9B- D0 07 BNE FDA4 non nul (le secteur de data existe), continue en FDA4
FD9D- A9 03 LDA #03 alloue 3 secteurs de data supplémentaires
FD9F- A0 00 LDY #00 ajoute un descripteur dans le "Channel Buffer"
FDA1- 20 5A F7 JSR F75A et sur la disquette si nécessaire

Charge le secteur de data suivant dans le "Channel's own Data Buffer" et y lit un octet

FDA4- 20 A8 F4 JSR F4A8 place l'adresse du début du "Channel Buffer" correspondant au NL en 00/01, celle du début du "Channel's own Data Buffer" en 02/03, celle du "Descriptor Buffer" en 04/05, celle du début du "General Buffer" en 06/07 et enfin, met à jour C083 (longueur d'une fiche ou #00) et 0B (flag "**S**/**R**/**D**") puis retourne avec Y = #00
FDA7- A0 03 LDY #03 vise l'octet de rang #03 du "Channel Buffer"
FDA9- B1 00 LDA (00),Y lit l'index de lecture dans le descripteur courant
FDAB- 18 CLC prépare une addition
FDAC- 69 02 ADC #02 teste si la fin du descripteur est atteinte
FDAE- D0 0A BNE FDBA sinon, continue en FDBA
FDB0- A0 04 LDY #04 si oui, vise octet de rang #04 du "Channel Buffer"
FDB2- B1 00 LDA (00),Y lit le n° du descripteur courant
FDB4- 69 00 ADC #00 l'incrémente (car la retenue C est mise à 1 en FDAC)
FDB6- 91 00 STA (00),Y et le remet en place
FDB8- A9 02 LDA #02 vise début de liste des coordonnées piste/secteur
FDBA- A0 03 LDY #03 vise l'octet de rang #03 du "Channel Buffer"
FDBC- 91 00 STA (00),Y lit l'index de lecture dans le descripteur courant
FDBE- 20 44 FD JSR FD44 charge un secteur de data du fichier qui se trouve sur la disquette et le place dans le "Channel's own Data Buffer"
FDC1- A0 00 LDY #00 index de lecture dans le "Channel's own Data Buffer"

Suite commune tant que Y n'est pas nul

FDC3- 98 TYA index de lecture dans le "Channel's own Data Buffer"
FDC4- A0 05 LDY #05 vise l'octet de rang #05 du "Channel Buffer"
FDC6- 91 00 STA (00),Y met à jour l'index du "Channel's own Data Buffer"
FDC8- A8 TAY index de lecture dans le "Channel's own Data Buffer"
FDC9- B1 02 LDA (02),Y lit un octet dans le "Channel's own Data Buffer"
FDCB- 60 RTS

Lit Y = index dans le "Channel's own Data Buffer"

FDCC- A0 05 LDY #05 vise l'octet de rang #05 du "Channel Buffer"
FDCE- 48 PHA sauvegarde A sur la pile
FDCF- B1 00 LDA (00),Y lit l'index du "Channel's own Data Buffer"

```
FDD1-   A8          TAY  sauve cet octet dans Y
FDD2-   68          PLA  et récupère A d'origine
FDD3-   60          RTS
```

Traite l'erreur

```
FDD4-   A2 0F       LDX #0F     pour "END_OF_FILE_ERROR"
FDD6-   4C 7E D6    JMP D67E    incrémente X et traite l'erreur n° X
```

Lit l'enregistrement suivant du fichier

(sous-programme FDD9-FE06, appelé par les commandes APPEND, JUMP, LTYPE, PUT, TAKE et TYPE)

Si la fin du fichier n'est pas atteinte, copie un enregistrement complet (type, longueur et valeur de la variable) du "Channel's own Data Buffer" (charge un second secteur dans le buffer si nécessaire), vers le "General Buffer", au pointeur 06/07.

FDD9- 20 0E FD JSR FD0E réinitialise 00/01, 02/03, 04/05 et 06/07, sauvegarde les octets de rang #03, #04 et #05 du "Channel Buffer" en C086, C087, C088 et teste si la fin du fichier est atteinte (Z = 1)

```
FDDC-   F0 F6       BEQ FDD4    si oui, "END_OF_FILE_ERROR"
FDDE-   A0 00       LDY #00     sinon, copie cet octet (type de variable)
FDE0-   91 06       STA (06),Y  dans le "General Buffer"
```

FDE2- 20 7A FD JSR FD7A avance dans le "Channel's own Data Buffer", charge un second secteur si nécessaire, et lit un octet (longueur de la variable)

```
FDE5-   A0 01       LDY #01     copie cet octet à la suite
FDE7-   91 06       STA (06),Y  dans le "General Buffer"
FDE9-   C8          INY  qui passe donc à #02
FDEA-   84 F5       STY F5      et le copie en F5 (index dans le "General Buffer")
FDEC-   85 F6       STA F6      copie aussi le dernier octet lu en F6 (longueur)
FDEE-   E6 F6       INC F6      prépare un compteur d'octets à copier
```

FDF0- 20 7A FD JSR FD7A avance dans le "Channel's own Data Buffer", charge un second secteur si nécessaire, et lit un octet (valeur de la variable)

```
FDF3-   A4 F5       LDY F5      récupère l'index d'écriture en cours
FDF5-   E6 F5       INC F5      et incrémente F5 pour le tour suivant
FDF7-   91 06       STA (06),Y  copie dans le "General Buffer" l'octet lu
FDF9-   C6 F6       DEC F6      décompte F6
FDFB-   D0 F3       BNE FDF0    reboucle en FDF0 tant qu'il en reste à copier
FDFD-   A0 00       LDY #00     vise le début du "General Buffer"
FDFF-   B1 06       LDA (06),Y  retourne avec les deux premiers octets du
FE01-   AA          TAX  "General Buffer" dans XA
FE02-   C8          INY  X = type de variable
FE03-   B1 06       LDA (06),Y  A = longueur de la variable
FE05-   60          RTS
FE06-   EA          NOP
```

EXÉCUTION DE LA COMMANDE SEDORIC APPEND

(Fichiers "**S**")
(sous-programme FE07-FE11 et suite à la commande JUMP)
(appelé aussi par la commande BUILD)

Rappel de la syntaxe

APPEND NL

Place le pointeur de fichier à la fin du fichier séquentiel correspondant au n° logique NL. Le fichier doit évidemment être ouvert, sinon "FILE_NOT_OPEN_ERROR" et être du bon type, sinon "FILE_TYPE_MISMATCH_ERROR".

Variables utilisées

En voir la liste et les fonctions dans les généralités sur la gestion de fichiers de data situées devant le sous-programme F3CF.

Analyse de la syntaxe et saisie du paramètre

FE07- 20 C0 FA JSR FAC0 vérifie l'existence de **FI**, la validité du NL et si le fichier est déjà ouvert, réinitialise les adresses 00/01, 02/03, 04/05, 06/07, 0B et C083 et retourne avec Y = 0, Z = 0 et N = 1 ("FILE_TYPE_MISMATCH_ERROR" si NL ne correspond pas à un fichier **S**équentiel)
FE0A- A9 FF LDA #FF fin de fichier
FE0C- 85 33 STA 33 place #FF en 33/34
FE0E- 85 34 STA 34
FE10- 30 09 BMI FE1B suite forcée en FE1B pour effectuer un JMP de #FFFF enregistrements, c'est à dire en pratique... jusqu'à la fin du fichier!

EXÉCUTION DE LA COMMANDE SEDORIC JUMP

(Fichiers **S**")
(sous-programme FE12-FE37)

Rappel de la syntaxe

JUMP NL,nombre_d'enregistrements

Déplace le pointeur de fichier du nombre d'enregistrement indiqué. Cette commande équivaut à APPEND si ce nombre est trop grand. Le fichier doit évidemment être ouvert, sinon "FILE_NOT_OPEN_ERROR" et être du bon type, sinon "FILE_TYPE_MISMATCH_ERROR".

Variables utilisées

En voir la liste et les fonctions dans les généralités sur la gestion de fichiers de data situées devant le sous-programme F3CF.

Analyse de la syntaxe et saisie des paramètres

FE12- 20 C0 FA JSR FAC0 vérifie l'existence de **FI**, la validité du NL et si le fichier est déjà ouvert, réinitialise les adresses 00/01, 02/03, 04/05, 06/07, 0B et C083 et retourne avec Y = 0, Z = 0 et N = 1 ("FILE_TYPE_MISMATCH_ERROR" si NL ne correspond pas à un fichier **S**équentiel)
FE15- 20 2C D2 JSR D22C D067/ROM exige une "," place TXTPTR sur l'octet suivant
FE18- 20 FA D2 JSR D2FA E853/ROM évalue un nombre non signé à TXTPTR et revient avec ce nombre dans YA, 33/34 et D3/D4 et mise à jour de TXTPTR sur l'octet suivant ce nombre (C = 0, N et Z positionnés selon l'octet de poids faible)

Teste si 33/34 est nul, RTS si oui, sinon décrémente 33/34

FE1B- 08 PHP sauvegarde les indicateurs 6502
FE1C- 78 SEI interdit les interruptions
FE1D- A5 33 LDA 33
FE1F- 05 34 ORA 34 teste si 33/34 est nul
FE21- F0 13 BEQ FE36 si oui, termine en FE36 (enregistrement trouvé)
FE23- A5 33 LDA 33
FE25- D0 02 BNE FE29 sinon, décrémente 33/34
FE27- C6 34 DEC 34
FE29- C6 33 DEC 33 passe à l'enregistrement suivant

Vérifie si fin de fichier atteinte

FE2B- 20 0E FD JSR FD0E réinitialise 00/01, 02/03, 04/05 et 06/07, sauvegarde les octets de rang #03, #04 et #05 du "Channel Buffer" en C086, C087, C088 et teste si la fin du fichier est atteinte (Z = 1)
FE2E- F0 06 BEQ FE36 si oui, termine en FE36

Copie l'enregistrement suivant dans le "General Buffer"

FE30- 20 D9 FD JSR FDD9 si la fin du fichier n'est pas atteinte, copie un enregistrement complet (type, longueur et valeur de la variable) du "Channel's own Data Buffer" (charge un second secteur dans le buffer si nécessaire), vers le "General Buffer", 06/07 pointe sur le type, la longueur et la valeur du data
FE33- 4C 1D FE JMP FE1D reprise forcée en FE1D

Enregistrement ou fin de fichier trouvé

FE36- 28 PLP récupère les indicateurs
FE37- 60 RTS

Copie l'enregistrement du "General Buffer" dans le "Channel's own Data Buffer", sauve le buffer sur la disquette et charge le secteur de data suivant si nécessaire)

(sous-programme FE38-FE57, appelé de FF25 par la commande BUILD et de FA4D par la commande PUT)

Les data présents sur la disquette sont mis à jour via le "Channel's own Data Buffer": ils sont copiés du "General Buffer" au point indiqué dans le "Channel's own Data Buffer" jusqu'à ce que la fin de celui-ci soit atteinte. Le "Channel's own Data Buffer" est alors sauvé et rechargé avec le secteur suivant contenant la suite de l'enregistrement à mettre à jour.

FE38- A0 00 LDY #00 index de lecture
FE3A- B1 06 LDA (06),Y lit type d'enregistrement dans le "General Buffer"
FE3C- 20 73 FD JSR FD73 écrit un octet dans le "Channel's own Data Buffer" et avance dans ce buffer, sauve le secteur de data sur la disquette et charge le secteur suivant si nécessaire car les divers enregistrements sont écrits les uns à la suite des autres
FE3F- A0 01 LDY #01 vise l'octet suivant pour lire la
FE41- B1 06 LDA (06),Y longueur d'enregistrement dans le "General Buffer"
FE43- C8 INY vise le début effectif de l'information et
FE44- 84 F7 STY F7 sauve dans F7 la valeur de cet index de lecture
FE46- 85 F8 STA F8 sauve dans F8 la longueur d'enregistrement
FE48- E6 F8 INC F8 pour copier la longueur et tous les octets
FE4A- 20 73 FD JSR FD73 écrit un octet dans le "Channel's own Data Buffer" et avance dans ce buffer, sauve le secteur de data sur la disquette et charge le secteur suivant si nécessaire, pour mise à jour de la suite des data
FE4D- A4 F7 LDY F7 récupère la valeur courante de l'index de lecture
FE4F- B1 06 LDA (06),Y lit un octet dans le "General Buffer"
FE51- E6 F7 INC F7 indexe le suivant
FE53- C6 F8 DEC F8 décompte le nombre d'octets à copier
FE55- D0 F3 BNE FE4A et reboucle tant qu'il en reste
FE57- 60 RTS

Copie le nom du fichier source dans BUFNOM et teste jokers

(ce sous-programme FE58-FE94, est appelé par la commande COPY)

FE58- 46 F2 LSR F2 force à zéro le b7 de F2 (flag "?" présent dans le nom de fichier cible sans homologue dans le nom de fichier source)
FE5A- 46 F4 LSR F4 force à zéro le b7 de F4 (flag "?" présent dans le nom de fichier source). Rappel: avec COPY et COPYO, les "?" sont autorisés dans le nom de fichier source, mais doivent être répétés aux mêmes endroits dans le nom de fichier cible ou alors il faut ?????????.??? dans le nom de fichier cible. Avec COPYM, les "?" sont autorisés dans le nom de fichier source, mais pas dans le nom de fichier cible.
FE5C- A2 0C LDX #0C pour copier 12 octets (nom et extension)
FE5E- CA DEX indexés de 11 à 0
FE5F- 30 22 BMI FE83 si fini, continue en FE83
FE61- BD 91 C0 LDA C091,X lit un octet du nom de fichier source
FE64- 9D 29 C0 STA C029,X écrit cet octet dans BUFNOM
FE67- BC 9E C0 LDY C09E,X lit l'octet correspondant du fichier cible
FE6A- C9 3F CMP #3F l'octet source est-il un "?"?
FE6C- F0 08 BEQ FE76 si oui, continue en FE76 avec C = 1
FE6E- C0 3F CPY #3F l'octet cible est-il un "?"?
FE70- D0 EC BNE FE5E sinon, reprend en FE5E (ni dans source, ni dans cible)
FE72- 66 F2 ROR F2 si oui, force le b7 de F2 à 1 (donc dans le cas ou un "?" a été

trouvé dans la cible, mais pas dans la source)

FE74-	D0 E8	BNE FE5E	reboucle en FE5E, (forcé car F2 non nul)
FE76-	66 F4	ROR F4	force le b7 de F4 à 1 (donc dans le cas ou un "?" a été trouvé dans la source)
FE78-	24 16	BIT 16	teste si b6 de 16 est à 1 (COPYM)
FE7A-	70 E2	BVS FE5E	si oui, reboucle en FE5E ("?" autorisés dans source pour COPYM)
FE7C-	C0 3F	CPY #3F	sinon, l'octet cible est-il aussi un "?"?
FE7E-	F0 DE	BEQ FE5E	si oui, reboucle en FE5E ("?" homologue requis est présent)
FE80-	4C AC D5	JMP D5AC	sinon, "INVALID_FILE_NAME_ERROR"
FE83-	24 F2	BIT F2	teste si le b7 de F2 est à 0 (pas de "?" présent dans la cible sans homologue dans la source)
FE85-	10 0C	BPL FE93	si oui, termine en FE93, sinon...
FE87-	A2 0C	LDX #0C	pour tester 12 octets qui doivent tous être des "?"
FE89-	BD 9D C0	LDA C09D,X	lit un octet du nom de fichier cible
FE8C-	C9 3F	CMP #3F	est-ce un "?"?
FE8E-	D0 F0	BNE FE80	sinon, "INVALID_FILE_NAME_ERROR"
FE90-	CA	DEX	octet précédent
FE91-	D0 F6	BNE FE89	reboucle en FE89 s'il en reste
FE93-	58	CLI	autorise les interruptions
FE94-	60	RTS	

EXÉCUTION DE LA COMMANDE SEDORIC LTYPE

(Fichiers "**S**")
(sous-programme FE95-FE97 et suite à TYPE)

Rappel de la syntaxe

LTYPE NL

Permet d'imprimer le contenu intégral d'un fichier séquentiel (voir TYPE).

ATTENTION: si vous aviez déjà visualisé le fichier avec TYPE, n'oubliez pas de remettre le pointeur d'enregistrements à la position voulue avec REWIND et éventuellement JUMP. Le fichier doit évidemment être ouvert, sinon "FILE_NOT_OPEN_ERROR" et être du bon type, sinon "FILE_TYPE_MISMATCH_ERROR".

Variables utilisées

En voir la liste et les fonctions dans les généralités sur la gestion de fichiers de data situées devant le sous-programme F3CF.

Entrée de la commande LTYPE

FE95-	20 C5 E7	JSR E7C5	PR SET et enchaîne avec la commande TYPE

EXÉCUTION DE LA COMMANDE SEDORIC TYPE

(Fichiers "**S**")
(sous-programme FE98-FEDF)

Rappel de la syntaxe

TYPE NL

Permet d'afficher, intégralement ou non, le contenu d'un fichier séquentiel. L'affichage commence à l'enregistrement courant du fichier (que l'on peut positionner par exemple avec la commande JUMP) et se termine à la fin du fichier (ou si l'on tape CTRL/C). Entre deux enregistrements, il est possible de faire une pause en tapant sur une touche et de reprendre en pressant la barre d'espace. Les expressions alphanumériques sont affichées comme des chaînes et les expressions numériques sous leur forme décimale. Le fichier doit évidemment être ouvert, sinon "FILE_NOT_OPEN_ERROR" et être du bon type, sinon "FILE_TYPE_MISMATCH_ERROR".

Variables utilisées

En voir la liste et les fonctions dans les généralités sur la gestion de fichiers de data situées devant le sous-programme F3CF.

Analyse de syntaxe et saisie du paramètre

FE98- 20 C0 FA JSR FAC0 vérifie l'existence de **FI**, la validité du NL et si le fichier est déjà ouvert, réinitialise les adresses 00/01, 02/03, 04/05, 06/07, 0B et C083 et retourne avec Y = 0, Z = 0 et N = 1 ("FILE_TYPE_MISMATCH_ERROR" si NL ne correspond pas à un fichier **S**équentiel)
FE9B- 20 02 D3 JSR D302 JSR EB78/ROM si touche frappée alors N = 1 et A = code ASCII correspondant, sinon N = 0
FE9E- 10 0C BPL FEAC continue en FEAC si pas de touche pressée pour afficher l'enregistrement suivant dans son entier
FEA0- 20 3D FF JSR FF3D XGETCAR attend un caractère au clavier, revient avec ce caractère dans A
FEA3- C9 20 CMP #20 la touche pressée est-elle un espace?
FEA5- F0 05 BEQ FEAC si oui, on reprend l'affichage
FEA7- C9 03 CMP #03 la touche pressée est-elle un CTRL/C?
FEA9- D0 F5 BNE FEA0 sinon, reprend l'attente d'une touche
FEAB- 60 RTS si oui, simple RTS, abandonne l'affichage

Reprend ou continue l'affichage

FEAC- 20 0E FD JSR FD0E réinitialise 00/01, 02/03, 04/05 et 06/07, sauvegarde les octets de rang #03, #04 et #05 du "Channel Buffer" en C086, C087, C088 et teste si la fin du fichier est atteinte (Z = 1)
FEAF- F0 16 BEQ FEC7 fin du fichier, termine en FEC7
FEB1- 20 D9 FD JSR FDD9 si la fin du fichier n'est pas atteinte, recopie un enregistrement complet (type, longueur et valeur de la variable) du "Channel's own Data Buffer" (charge un second secteur dans le buffer si nécessaire), dans le "General Buffer", 06/07 pointe sur le type, la longueur et la valeur du data

```
FEB4-    F0 E5       BEQ FE9B  la longueur de l'enregistrement est nulle, reprend en FE9B
FEB6-    85 F2       STA F2    sauve la longueur de l'enregistrement en F2
FEB8-    8A          TXA  type d'enregistrement
FEB9-    10 0F       BPL FECA  suite en FECA si ce n'est pas une chaîne alphanumérique
FEBB-    C8          INY  progresse dans la chaîne
FEBC-    B1 06       LDA (06),Y  lit un octet de la chaîne
FEBE-    20 2A D6    JSR D62A  XAFCAR et l'affiche (écran ou imprimante)
FEC1-    C6 F2       DEC F2    nombre de caractères restant à afficher
FEC3-    D0 F6       BNE FEBB  reboucle en FEBB, s'il en reste
FEC5-    F0 D4       BEQ FE9B  reboucle en FE9B (passe à l'enregistrement suivant)
```

L'affichage sur le périphérique courant est terminé

FEC7- 4C D6 E7 JMP E7D6 restaure l'affichage sur l'écran en exécutant un JSR C82F/ROM (mettre l'imprimante hors service) et retourne

L'enregistrement n'est pas une chaîne alphanumérique

```
FECA-    18          CLC  prépare l'addition 06/07 = 06/07 + 2
FECB-    A5 06       LDA 06    06/07 pointera non plus sur le début du "General
FECD-    A4 07       LDY 07    Buffer", mais sur le début des data
FECF-    69 02       ADC #02   (saute le type et la longueur de l'enregistrement)
FED1-    90 01       BCC FED4
FED3-    C8          INY  report de retenue
```

FED4- 20 BA D2 JSR D2BA JSR DE7B/ROM place dans ACC1 (floating point accumulator) la valeur pointée par AY

FED7- 20 D2 D2 JSR D2D2 E0D5/ROM convertit ACC1 en chaîne décimale AY située à partir de #0100 (qui contient le signe "-" ou un espace) et terminée par #00

```
FEDA-    20 37 D6    JSR D637  XAFSTR affiche chaîne terminée par 0 d'adresse AY
FEDD-    4C 9B FE    JMP FE9B  reprise en FE9B pour l'enregistrement suivant
```

EXÉCUTION DE LA COMMANDE SEDORIC BUILD

(Fichiers "**S**")
(sous-programme FEE0-FF42)

Rappel de la syntaxe

BUILD NL

Permet de saisir des caractères au clavier et de les sauver dans le fichier séquentiel de n° logique NL, préalablement ouvert à l'aide de la commande OPEN S. La fin de la saisie se fait en tapant CTRL/C

Variables utilisées

En voir la liste et les fonctions dans les généralités sur la gestion de fichiers de data situées devant le sous-programme F3CF.

Principe et informations non documentées

Ajoute les caractères tapés à la fin d'un fichier non vide, même si on a fait un REWIND avant, ce qui est de toute façon inutile, puisque la première chose que BUILD effectue est un APPEND. La conséquence en est que la gestion d'un fichier "mixte" dont tous les enregistrements n'ont pas forcément la taille "standard" de 216 caractères peut être complexe.

Les caractères tapés sont collectés dans un tampon de 218 octets situé dans le "General Buffer" et sont sauvés sur la disquette par ajout d'enregistrements successifs de type alphanumérique de 216 octets à la fin du fichier séquentiel ouvert. Il est difficile d'exploiter ces chaînes alphanumériques de 216 octets sans en connaître l'organisation (sauf avec LTYPE et TYPE qui le font de manière transparente pour l'utilisateur). Lors du CTRL/C final, le dernier tampon incomplet est sauvegardé avec ses #00 inutiles sans mise à jour du nombre exact de caractères que comporte le dernier enregistrement.

Entrée de la commande BUILD

FEE0- 20 07 FE JSR FE07 effectue un APPEND, c'est à dire se place à la fin du fichier ("FILE_TYPE_MISMATCH_ERROR" si NL ne correspond pas à fichier "**S**")
FEE3- 20 00 FF JSR FF00 initialise un premier tampon de 218 octets forcés à #00 dans le "General Buffer"
FEE6- 20 3D FF JSR FF3D XGETCAR attend un caractère au clavier, revient avec ce caractère dans A
FEE9- A4 F2 LDY F2 récupère Y, pointeur dans le tampon
FEEB- C9 03 CMP #03 le caractère saisi est-il un CTRL/C?
FEED- F0 48 BEQ FF37 si oui, continue en FF37, fin de saisie, ajoute le #FF qui marque la fin de fichier et sauve le tampon
FEEF- C9 0D CMP #0D le caractère saisi est-il un RETURN?
FEF1- D0 05 BNE FEF8 sinon, saute les deux instructions suivantes
FEF3- 20 1B FF JSR FF1B si oui, met CR dans tampon, sauve celui-ci si plein
FEF6- A9 0A LDA #0A prend un LF dans A pour l'ajouter à la suite du CR
FEF8- 20 1B FF JSR FF1B met le caractère dans le tampon, sauve celui-ci si plein
FEFB- 84 F2 STY F2 sauvegarde le pointeur Y dans F2
FEFD- 4C E6 FE JMP FEE6 et reboucle en FEE6

Initialise un premier enregistrement de 216 octets forcés à #00 dans le "General Buffer"

FF00- 20 A8 F4 JSR F4A8 place l'adresse du début du "Channel Buffer" correspondant au NL en 00/01, celle du début du "Channel's own Data Buffer" en 02/03, celle du "Descriptor Buffer" en 04/05, celle du début du "General Buffer" en 06/07 et enfin, met à jour C083 (#00) et 0B (flag "**S**") puis retourne avec Y = #00
FF03- A9 80 LDA #80 type d'enregistrement "chaîne alphanumérique"
FF05- 91 06 STA (06),Y place #80 dans le premier octet du tampon
FF07- C8 INY
FF08- A9 D8 LDA #D8 longueur de l'enregistrement (216 octets)
FF0A- 91 06 STA (06),Y place cette valeur en deuxième position du tampon
FF0C- A9 00 LDA #00 pour remise à zéro de la chaîne alphanumérique
FF0E- C8 INY vise le premier octet de la chaîne
FF0F- 84 F2 STY F2 garde Y dans F2 index dans la chaîne

FF11- 91 06 STA (06),Y force à 0 les 216 octets suivants
FF13- C8 INY
FF14- C0 DA CPY #DA teste si Y atteint #DA (218 = 216 + les 2 premiers)
FF16- D0 F9 BNE FF11 reboucle tant qu'il en reste
FF18- A0 02 LDY #02 vise le début de la chaîne
FF1A- 60 RTS et retourne

Affiche le caractère à l'écran, le place dans le tampon, sauve celui-ci s'il est plein et en initialise un nouveau

FF1B- 91 06 STA (06),Y place dans le tampon le caractère saisi
FF1D- 20 2A D6 JSR D62A XAFCAR affiche le caractère ASCII contenu dans A
FF20- C8 INY vise la place suivante dans le tampon
FF21- C0 DA CPY #DA la fin du tampon est-elle atteinte?
FF23- D0 F5 BNE FF1A sinon, retourne à la routine de saisie

La fin du tampon est atteinte: sauve les caractères présents dans le tampon situé dans le "General Buffer", recopie ce tampon dans le "Channel's own Data Buffer", y ajoute un #FF de fin de fichier, écrit le "Channel's own Data Buffer" sur la disquette et enfin initialise un nouveau tampon dont les 218 octets sont forcés à zéro

FF25- 20 38 FE JSR FE38 recopie l'enregistrement du "General Buffer" dans le "Channel's own Data Buffer" en utilisant le secteur suivant si nécessaire. C'est la procédure normale pour un fichier "**S**" où les enregistrements sont mis bout à bout dans le "Channel's own Data Buffer", lequel est sauvegardé dès qu'il est plein. La seule différence est qu'ici les enregistrements sont de longueur fixe pré-definie (218 octets).
FF28- A9 FF LDA #FF marqueur de fin de fichier
FF2A- 20 CC FD JSR FDCC lit l'index Y du "Channel's own Data Buffer"
FF2D- 91 02 STA (02),Y place #FF à l'adresse 02/03 + Y
FF2F- 20 46 FD JSR FD46 sauve sur la disquette le secteur du fichier qui est présent dans le "Channel's own Data Buffer", ce qui constitue une sauvegarde provisoire des derniers caractères saisis
FF32- A0 02 LDY #02 reprend Y = 2 (vise le début du tampon)
FF34- 4C 00 FF JMP FF00 suite en FF00 et retourne à la routine de saisie

Le caractère saisi était un CTRL/C, termine en recopiant le tampon dans le "Channel's own Data Buffer", en sauvant sur le secteur suivant de la disquette si nécessaire, ajoute un #FF de fin de fichier dans le "Channel's own Data Buffer" et enfin écrit le "Channel's own Data Buffer" sur la disquette

FF37- 20 25 FF JSR FF25 recopie le tampon dans le "Channel's own Data Buffer", y ajoute un #FF de fin de fichier, écrit le "Channel's own Data Buffer" sur la disquette et enfin initialise un nouveau tampon dont les 218 octets sont forcés à zéro (est-ce bien nécessaire?)
FF3A- 4C 46 FD JMP FD46 sauve sur la disquette le secteur du fichier qui est présent dans le "Channel's own Data Buffer" (sauvegarde des derniers caractères saisis) et retourne

XGETCAR attend un caractère au clavier et revient avec ce caractère dans A

FF3D- 20 45 D8 JSR D845 XKEY prend un caractère au clavier (entrée générale)
FF40- 10 FB BPL FF3D reboucle tant qu'aucune touche n'a été pressée

FF42- 60 RTS

TABLE DES VECTEURS SYSTÈME

Cette table se trouve de FF43 à FFC6

FF43- 4C 36 ED JMP **ED36** **XLINPU** routine principale de LINPUT (routine de saisie de chaîne), au retour F4 contient le mode de sortie et D0, D1, D2 donne la longueur et l'adresse de la chaîne dans la zone de stockage des chaînes sous HIMEM.

FF46- 4C 98 D3 JMP **D398** **XCRGET** incrémente TXTPTR, lit un caractère (CHRGET), les espaces sont sautés, le met dans A, le convertit en MAJUSCULE, Z = 1 si fin d'instruction (0 ou :), C = 0 si chiffre 0 à 9 (soit #30 à #39), sinon C =1. Y et X inchangés (identique au CHRGET du BASIC, mais en plus convertit les minuscules en MAJUSCULES)

FF49- 4C 9E D3 JMP **D39E** **XCRGOT** relit le caractère à TXTPTR (sans incrémenter TXTPTR = CHRGOT), puis le convertit en MAJUSCULE, les espaces sont sautés, Z = 1 si fin d'instruction (0 ou :), C = 0 si caractère chiffre 0 à 9 (soit #30 à #39), sinon C = 1, Y et X inchangés (identique au CHRGOT du BASIC, mais en plus convertit les minuscules en MAJUSCULES)

FF4C- 4C 4F D4 JMP **D44F** **XNF** lit un nom de fichier non-ambigu à TXTPTR et l'écrit dans BUFNOM

FF4F- 4C 51 D4 JMP **D451** **XNFA** lit un nom de fichier ambigu à TXTPTR et l'écrit dans BUFNOM

FF52- 4C 64 D3 JMP **D364** **XAFSC** affiche le (X+1) ème message externe terminé par "caractère + 128" EXTMS doit contenir l'adresse - 1 du premier message

FF55- 4C F3 F3 JMP **F3F3** vérifie l'existence du "pseudo-tableau" **FI** au début des tableaux et le crée s'il n'existe pas encore

FF58- 4C A8 F4 JMP **F4A8** place l'adresse du début du "Channel Buffer" correspondant au NL en 00/01, celle du début du "Channel's own Data Buffer" en 02/03, celle du "Descriptor Buffer" en 04/05, celle du début du "General Buffer" en 06/07 et enfin, met à jour C083 (longueur d'une fiche ou #00) et 0B (flag "**S**/**R**/**D**") puis retourne avec Y = #00

FF5B- 4C D9 FD JMP **FDD9** si la fin du fichier n'est pas atteinte, copie un enregistrement complet (type, longueur et valeur de la variable) du "Channel's own Data Buffer" (charge un second secteur dans le buffer si nécessaire), vers le "General Buffer", 06/07 pointe sur le type, la longueur et la valeur du data

FF5E- 4C 38 FE JMP **FE38** écrit l'enregistrement du "General Buffer" dans le "Channel's own Data Buffer" en utilisant le secteur suivant si nécessaire

FF61- 4C 46 FD JMP **FD46** sauve sur la disquette le secteur du fichier qui est présent dans le "General Buffer"

FF64- 4C 2A D6 JMP **D62A** **XAFCAR** affiche le caractère ASCII contenu dans A

FF67- 4C 13 D6 JMP **D613** **XAFHEX** affiche en hexadécimal le contenu de A

FF6A- 4C 37 D6 JMP **D637** **XAFSTR** affiche la chaîne terminée par un 0 et dont l'adresse est donnée par AY

FF6D- 4C D8 D5 JMP **D5D8** **XROM** permet d'exécuter à partir de la RAM une routine ROM. Le JSR XROM doit être suivi dans l'ordre de l'adresse de la routine pour la V1.0, puis de l'adresse pour la V1.1

FF70- 4C EA E0 JMP **E0EA** **charge un fichier** selon X = POSNMX, POSNMP et POSNMS, VSALO0, VSALO1, DESALO

FF73- 4C E5 E0 JMP **E0E5** **XLOADA** charge le fichier dont le nom est dans BUFNOM, selon VSALO0, VSALO1, DESALO

FF76- 4C 28 DE JMP **DE28** **XDEFSA** positionne les valeurs par défaut pour XSAVEB (en fait, positionne pour sauver le programme BASIC)

FF79- 4C E6 DF JMP **DFE6** **XDEFLO** positionne les valeurs par défaut pour XLOADA

FF7C- 4C 9C DE JMP **DE9C** **XSAVEB** sauve le fichier de nom contenu dans BUFNOM, selon VSALO0, VSALO1, DESALO, FISALO, EXSALO

FF7F- 4C 66 E2 JMP **E266** **XNOMDE** détruit le fichier indexé par POSNMX, dont le secteur de catalogue est dans BUF3 (en fait, tout est positionné comme après un XTVCAT)

FF82- 4C 2D DD JMP **DD2D** **XCREAY** crée une table piste secteur de AY secteurs, en fait marque dans la bitmap en BUF2 que le secteur AY est occupé

FF85- 4C 15 DD JMP **DD15** **XDETSE** libère le secteur Y de la piste A sur la bitmap courante dans BUF2 et incrémente le nombre de secteurs libres. Retourne avec C = 1 si ce secteur était déjà libre. Ne pas oublier de sauver le plus tôt possible cette nouvelle bitmap avec SMAP

FF88- 4C 6C DC JMP **DC6C** **XLIBSE** cherche un secteur libre dans la bitmap dans BUF2, retourne avec A = n° de piste et Y = n° de secteur (sinon "DISK_FULL_ERROR")

FF8B- 4C C0 DB JMP **DBC0** **XWDESC écrit** le ou les **descripteurs** du fichier à sauver. Revient avec le nombre de secteurs à sauver dans NSSAV (C05A/5B), les coordonnées du premier secteur descripteur dans PSDESC (C05C/5D), le nombre de descripteurs utilisés dans NSDESC (C05E) et premier descripteur en place

FF8E- 4C 59 DB JMP **DB59** **XTRVCA** cherche une place libre dans le catalogue. A la sortie, POSNMX, POSNMP et POSNMS indiquent la position de la place réservée

FF91- 4C A5 DB JMP **DBA5** **cherche** le POSNMX de la **première place libre** dans le directory

FF94- 4C 41 DB JMP **DB41** **ajuste POSNMX sur entrée suivante** du catalogue et reprend la recherche dans le catalogue du fichier indiqué dans BUFNOM (Z = 1 si fini)

FF97- 4C 30 DB JMP **DB30** **XTVNM** cherche sur le lecteur courant le nom contenu dans BUFNOM. A la sortie, POSNMX, POSNMP, et POSNMS contiennent la position du nom dans le catalogue, et Z = 1 si le fichier n'est pas trouvé

FF9A- 4C 2D DB JMP **DB2D** vérifie que la disquette en place est bien une disquette SEDORIC, cherche le fichier dans le catalogue, revient avec le bon secteur de catalogue en place (de coordonnées POSNMP et POSNMS) et avec X = POSNMX, pointant sur "l'entrée" cherchée ou avec Z = 1 si le fichier n'a pas été trouvé

FF9D- 4C 07 DB JMP **DB07** **XCABU** transfère dans BUFNOM le nom de fichier contenu dans le secteur de catalogue placé dans BUF3, à la position POSNMX

FFA0- 4C FE DA JMP **DAFE** Charge dans BUF3 le secteur pointé par POSNMP et POSNMS puis exécute XCABU (DB07).

FFA3- 4C EE DA JMP **DAEE** **XBUCA** transfère le nom de fichier contenu dans BUFNOM dans le secteur de catalogue contenu dans BUF3, à la position POSNMX

FFA6- 4C CE DA JMP **DACE** **XVBUF1** remplit BUF1 de zéros.

FFA9- 4C A4 DA JMP **DAA4** **XSVSEC** écrit un secteur selon DRIVE, PISTE, SECTEUR et RWBUF

FFAC- 4C 9E DA JMP **DA9E** **XSAY** sauve le secteur visé par RWBUF au secteur Y de la piste A

FFAF- 4C 91 DA JMP **DA91** **XSBUF1** sauve BUF1 au secteur Y de la piste A

FFB2- 4C 82 DA JMP **DA82** **XSCAT** sauve le secteur de catalogue contenu dans BUF3, selon POSNMP et POSNMS

FFB5- 4C 8A DA JMP **DA8A** ancienne routine **XSMAP** (sauve le secteur de bitmap sur la disquette), a été déportée en DC80

FFB8- 4C 73 DA JMP **DA73** **XPRSEC** lit un secteur selon DRIVE, PISTE, SECTEUR et RWBUF

FFBB- 4C 6D DA JMP **DA6D** **XPAY** charge dans RWBUF le secteur Y de la piste A

FFBE- 4C 5D DA JMP **DA5D** **XPBUF1** charge dans BUF1 le secteur Y de la piste A

FFC1- 4C 4C DA JMP **DA4C** **XPMAP** prend le premier secteur de bitmap dans BUF2, vérifie le format, met à zéro le b7 de 2F (flag”première bitmap chargée”).

FFC4- 4C CD CF JMP **CFCD** **XRWTS** accès à la routine de gestion des lecteurs. X contient

la commande. En sortie, Z = 1 si pas d'erreur, Z = 0 sinon. V = 1 si la disquette est protégée en écriture. DRIVE, PISTE, SECTEUR, et RWBUF doivent être à jour

COPYRIGHT

FFC7-	53 45 44 4F 52 49 43 20 31 2E 30 20 70 61 72 20	SEDORIC 1.0 par	
FFD7-	46 2E 42 52 4F 43 48 45 20 65 74 20	F.BROCHE et	
FFE3-	44 2E 53 45 42 42 41 47	D.SEBBAG	
FFEB-	28 63 29 20 31 39 38 35 20 45 55 52 45 4B 41	(c) 1985 EUREKA	
FFFA-	**21 D1**	Vecteur NMI	en D121
FFFC-	**10 23**	Vecteur RESET	en 2310 (non valide)
FFFE-	**A5 D0**	Vecteur IRQ	en D0A5

NOTES PERSONNELLES