SEDORIC À NU

André Chéramy, Yann Legrand
and Ray MacLaughlin

Deuxième Edition (1997)

AVANT-PROPOS

Ce manuel représente, comme vous pouvez l'imaginer, un énorme travail. Nous sommes partis avec très peu d'informations: "L'Oric à nu" de Fabrice Broche, le manuel du Sédoric, un article paru dans "Théoric". Les noms des vecteurs, des variables et des routines ont été fidèlement conservés autant que faire se peut. Aucun des termes utilisés dans ces diverses sources n'a été modifié afin que chacun puisse s'y retrouver plus facilement! Nous avons été obligés de créer quelques appellations nouvelles, parfois en Franglais, en essayant de privilégier, les habitudes, l'intuition et même la clarté!

Nous avons essayé d'utiliser le moins d'abréviations possible:

corresp	correspondant(e, s, es)
CR	Carriage Return (retour en début de ligne)
CTRL	contrôle
LF	Line Feed (passage à la ligne suivante)
LL	octet de poids faible (Low)
LM	Langage Machine ou langage assembleur
HH	octet de poids fort (High)
s/p	sous-programme

Afin de simplifier au maximum, les adresses, en hexadécimal sont écrites sans #, certaines adresses, comprises entre C400 et C7FF sont suivies d'une lettre minuscule (a, b, c, etc..) afin de différencier à quelle banque interchangeable elle correspond. Enfin, les valeurs absolues (ou "immédiates") hexadécimales sont précédées d'un "#" comme en BASIC (la notation usuelle "#$", utilisée en LM nous a semblé inutilement compliquée).

Malgré nos soins, nos relectures et nos vérifications, il est évident qu'un ouvrage de cette importance comporte des erreurs. Si vous en trouvez, n'hésitez pas à nous en faire part. De toute façon, faites nous parvenir vos coordonnées, afin que nous puissions vous faire parvenir gratuitement une éventuelle mise à jour[1]. La présente version (reconnaissable à son texte en "Times New Roman") a en fait subit de légères corrections par rapport à la précédente (reconnaissable à ses caractères "Courrier").

Il est possible de faire beaucoup plus avec Sédoric qu'il n'est indiqué dans le manuel. Notamment, nous indiquons comment utiliser les routines de Sédoric dans un programme en LM. Certains écueils dûs à des bogues sont signalés. Pour chaque commande ou presque une rubrique "Non documenté" vous apportera de précieuses informations.

Ce livre ne s'adresse pas uniquement aux maniaques du LM et aux bidouilleurs, mais aussi très simplement à l'utilisateur "normal" de notre machine favorite, à condition qu'il ne se laisse pas impressionner par les pages de listing de désassemblage.

Ce livre n'est pas à lire, mais à utiliser au cas par cas, en fonction de vos problèmes. Il comporte donc de nombreuses redites, afin que chaque partie soit compréhensible séparément. Nous espérons que vous y trouverez les réponses aux questions que vous vous posez.

[1]Correspondance: André Chéramy, 54 rue de Sours, 28000 CHARTRES

*** PREMIÈRE PARTIE DE LA RAMOVERLAY *** (DE C000 À C3FF)

...et tout d'abord quelques informations générales sur Sédoric..

ANALYSE DES COMMANDES SEDORIC

Lors de l'initialisation, le noyau du code Sédoric est implanté en **RAMOVERLAY** (RAMOV, C000 à FFFF) et quelques modifications sont apportées aux pages 0, 2, et 4 de la RAM afin de permettre l'analyse des commandes Sédoric et d'accéder à cette RAMOV tout en préservant l'accès aux commandes BASIC (mêmes adresses, C000 à FFFF, mais en ROM).

Le coeur de l'analyseur de commande se trouve en ROM. Sans vouloir entrer dans le détail (voir "L'Oric à Nu" de Fabrice Broche), cet analyseur fait appel a une routine de lecture, située en page zéro (routine CHARGET, 00E2 à 00F2 avec entrée secondaire en 00E8). Cette routine actualise le pointeur de texte (**TXTPTR**, 00E9/00EA) sur la ligne de commande (**TIB**, Terminal Input Buffer) ou bien sur la ligne de programme, lit le caractère présent au pointeur, exécute un saut en ROM à l'adresse d'entrée de l'interpréteur (JSR ECB9) suivi d'un retour au point d'appel initial (RTS). Sous Sédoric, le JSR en ROM et le RTS sont remplacés par un saut en page 4 (JMP 0400). C'est là qu'intervient la fameuse page 4 du Sédoric.

La routine **CHARGET** (00E2 ou 00E8) peut être appelée à plusieurs endroits à partir de la ROM, notamment en C90C ou en CA88. Dans ces deux cas, l'adresse de retour-1 est empilée. Si cette adresse est C90E, il s'agit de l'adresse de retour en C90F appartenant au s/p "exécuter une ligne" de l'interpréteur BASIC. Si cette adresse est CA8A, il s'agit de l'adresse de retour en CA8B appartenant au s/p "IF".

La routine en 0400 effectue plusieurs tâches:

Elle analyse si le caractère lu (et situé maintenant dans l'accumulateur A) est un chiffre. Si c'est le cas retour au cours normal des choses (c'est à dire à l'interpréteur en ECB9).

De même si A contient un code égal ou supérieur à #80 (c'est à dire un mot-clé BASIC) on retourne à l'interpréteur en ECB9.

Si ni l'un ni l'autre n'est le cas, les registres A et X sont sauvegardés en 000E et 000F avant de recevoir l'adresse présente sur la pile afin de savoir d'où CHARGET avait été appelé. Si aucune des deux adresses indiquées plus haut n'est trouvée, l'adresse de retour est remise en place sur la pile, les valeurs initiales des registres A et X sont restaurées et le programme retourne à l'interpréteur en ECB9 (tout se passe comme si le détour par la page 4 du Sédoric n'avait pas eu lieu).

Si l'adresse C90E indiquée plus haut est trouvée, le bit n°7 du drapeau 04FC est mis à zéro. Si c'est l'adresse CA8A ce bit est mis à 1.

Puis la routine recherche si un signe "=" est présent sur la ligne de commande ou de programme et ce jusqu'au prochain "0" ou ":" marquant la fin de l'instruction. Si un signe "=" est rencontré, il s'agissait d'affecter une variable, l'adresse de retour est remise en place sur la pile, les valeurs initiales des registres A et X sont restaurées et le programme retourne à l'interpréteur en ECB9.

S'il n'y a pas de signe "=", on est en présence d'une commande Sédoric on continue sans restaurer l'adresse de retour sur la pile. S'agit-il d'un mot-clé utilisateur? (voir manuel Sédoric page 106). Un JSR 04E9 est effectué, qui conduit à un JMP à l'adresse de l'interpréteur utilisateur ou à un simple RTS (cas général, si pas d'interpréteur utilisateur).

Un JSR 0467 est alors effectué (entrée vecteur "!"). Une bascule sur la RAMOV est opérée, puis un saut au sous-programme D3AE (**INTERPRETEUR SEDORIC** d'où l'on reviendra par un RTS), enfin une bascule sur la ROM permet de reprendre le cours normal de la routine en 0447 où le flag 04FC est testé.

Si le bit n°7 de ce flag est nul un JMP C8C1 (s/p exécuter une ligne) est effectué. Sinon (bit n°7 à 1), un "IF" est en cours et on met à 1 le bit n°7 du flag 0252 (drapeau "IF" en cours).

Le RTS final achève ce s/p 0400 et permet de retourner au programme appelant.

NB: Par simplification et en l'absence de spécification, les adresses de la ROM indiquées sont celles de la version 1.1

(Atmos).

Le désassemblage de la page 4 se trouve un peu plus loin, en C700 (page 15 de ce livre).

TABLE DES VARIABLES SYSTEMES

Page #00

00 à 0B	zone de travail (RENUM, fichiers) dont:
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 "Buffer Général"
	adresse du début de la fiche dans le "Buffer Général"
	adresse du début des data dans le "Buffer Général"
08/09	rang du secteur où se trouve la fiche depuis le début du fichier
0A	n° logique en cours (de 0 à 63)
0B	FTYPE, type de fichier: OPEN "**R**" (#00) ou "**S**" (#80) ou "**D**" (#01)
0C	sauvegarde de A
0D	sauvegarde de Y
0E	sauvegarde de A ou de LL
0F	sauvegarde de X ou de HH
16/17	sauvegarde de TXTPTR
18/19	adresse utilisée pour encodage/décodage des mots-clés
24	utilisé dans s/p
27	sauvegarde de P (en plus des utilisations Oric/Atmos)
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)
35/84	TIB, Terminal Input Buffer, c'est à dire tampon clavier (80 octets)
7B	(#00 Atmos et #01 Sédoric)
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 1er 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
F0/F1	Vecteur Interpréteur (ECB9 Atmos et 0400 Sédoric)
F2 à F9	TRAV0 à TRAV7 zone de travail dont:
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 corresp au n° logique dans la "Table NL" (cette paire d'octet est l'offset du début du "Channel Buffer" du fichier ouvert corresp, 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 "Buffer Général"
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 "Record Buffer"
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 OPEN R (b3 à 1) et #10 si OPEN S (b4 à 1)
	(ce sont les types Sédoric: les "pseudo-fichiers" d'accès **D**isques n'en ont pas)

Page #02

023C/3D	Vecteur "Prendre un caractère au clavier" (EB78 Atmos et 045B Sédoric)
0245/46	Vecteur IRQ (EE22 Atmos et 0488 Sédoric)
0248/49	Vecteur NMI (F8B2 Atmos et 04C4 Sédoric)
0271	"Couleur" du curseur (#01 Atmos et #00 Sédoric)
0274/0275	Clignotement curseur (#0004 Atmos et #000B Sédoric)

276/77 Timer 3 (#6B81 Atmos et #F6D7 Sédoric)
2A0 (#FF Atmos et #05 Sédoric)
2BE (#80 Atmos et #FF Sédoric)
2F5/F6 Vecteur ! (D336 Atmos et 0467 Sédoric)
2FC/FD Vecteur &() (D336 Atmos et 0461 Sédoric)

Page #03 (I/O = Entrées/Sorties)

Lorsqu'on POKE ou PEEK une adresse entre #0300 et #3FF, le VIA 6522 (Versatile Interface Adaptator) est automatiquement activé. La ROM utilise les adresses de #0300 à #030F pour le port imprimante et le PSG8912 (Programmable Sound Generator, qui gère aussi le clavier). Sédoric utilise en plus les adresses de #0310 à #031B pour le lecteur de disquette. Pour plus d'information voir "L'Oric à Nu" pages 18 à 23 et "Manuel de l'Oric Atmos" pages 257 à 261 et 304 à 312. Voici un résumé des registres du VIA:

0300 à 030F I/O ROM pour PSG, clavier et imprimante

300- VIA**DRB D**ATA **R**egistre du port **B**: les 8 bits de DATA port B (entrée ou sortie selon VIADDRB). En sortie, ces données sont toujours latchées (figées jusqu'à la prochaine opération). En entrée, elles sont latchées selon VIAACR (voir plus bas). La lecture et l'écriture dans VIADRB modifient VIAPCR et VIAIFR (CB2 et CB1 ainsi que IRQ corresp).

301- VIA**DRA D**ATA **R**egistre du port **A**: les 8 bits de DATA port A (entrée ou sortie selon VIADDRA). En sortie, ces données sont toujours latchées. En entrée, elles sont latchées selon VIAACR (voir plus bas). La lecture et l'écriture dans VIADRA modifient VIAPCR et VIAIFR (CA2 et CA1 ainsi que IRQ corresp). On peut utiliser VIAORA/VIAIRA à la place de VIADRA afin de ne pas modifier les status d'interruption de CA2 et CA1.

302- VIA**DDRB D**irection des **D**onnées **R**egistre port **B**: chacun des 8 bits de ce registre indique si le bit corresp de VIADRB est en entrée (bit à 0) ou en sortie (bit à 1).

303- VIADDRA idem pour le port A

304- VIA**T1L** LL T1 counter

305- VIA**T1H T**imer **1** HH T1 counter

306- VIA**T1LL** LL T1 latch

307- VIA**T1LH** HH T1 latch

308- VIAT2L **T**imer **2**

309- VIAT2H

30A- VIA**SR S**hift **R**egistre (inutilisable avec l'Oric)

30B- VIA**ACR A**utorisation **C**ontrôle **R**egistre: 8 bits comme suit:
b0: **LA L**atch en entrée sur le port **A**
b1: **LB L**atch en entrée sur le port **B**
b2 à b4: non utilisés avec l'Oric (Shift Registre)
b5: **MT2 M**ode **T**imer **2** décrémentation selon O2 si 0, selon PB6 si 1
b6: **MT1 M**ode **T**imer **1** monostable si à 0 ou roue libre si à 1
b7: **MPB7 M**ode **PB7** sortie interdite si 0 ou autorisée si 1 (T1=0)

30C- VIA**PCR P**ériphérique **C**ontrôle **R**egistre: 8 bits comme suit:
b0: **FA** à 1 si détecte **F**ront montant sur broche C**A**1, à 0 si front descendant
b1 à b3: codage entrée/sortie sur CA2 (port A)("Oric à nu" page 21)
b4: **FB** à 1 si détecte **F**ront montant sur broche C**B**1, à 0 si front descendant
b5 à b7: codage entrée/sortie sur CB2 (port B)("Oric à nu" page 21)

30D- VIA**IFR** Indication **I**nterruption **R**egistre: 8 bits comme suit:
b0 et b1: **CA2** & **CA1** 0 si lecture/écriture VIADRA, 1 si transition CA2 ou CA1
b2: non utilisé avec l'Oric (Shift Registre)
b3 et b4: **CB2** & **CB1** 0 si lecture/écriture VIADRB, 1 si transition CB2 ou CB1
b5: **T2** 0 si lecture sur VIAT2L ou écriture sur VIAT2H et 1 si T2 = 0
b6: **T1** 0 si lecture sur VIAT1L ou écriture sur VIAT1H et 1 si T1 = 0
b7: **IRQ** 0 si IRQ traitée et 1 si IRQ activée et autorisée

30E- VIA**IER I**nterruption autorisation (**E**nable) **R**egistre: 8 bits:
b0 et b1: **CA2** et **CA1** mettre à 1 pour spécifier CA2 et/ou CA1
b2: non utilisé avec l'Oric (Shift Registre)
b3 et b4: **CB2** et **CB1** mettre à 1 pour spécifier CB2 et/ou CB1
b5 et b6: **T2** et **T1** mettre à 1 pour spécifier T2 et/ou T1
b7: **EN** mettre à 0 pour interdire ou à 1 pour autoriser les interruptions spécifiées par les bits b0 à b6

30F- VIA**ORA/**VIA**IRA O**utput **R**egistre **A/I**nput **R**egistre **A:** Ce registre peut être utilisé comme VIADRA sans modifier les IRQ de CA1 et de CA2

0310 à 031B I/O RAMOV lecteur de disquette Oric

310- reçoit X, code de commande: #88 (lecture), #A8 (écriture), #C0 (activation drive C00B)
311- piste sous la tête
312-
313- registre DATA pour lecture/écriture disquette
314- reçoit 04FB = code DRIVE et FACE
315-
316-
317-
318- Ready
319-
31A-

031B-

Page #04

04F0/F1		EXEVEC+1	adresse d'exécution
04FB		ROMRAMOV	flag ROM/RAMOV
			code DRIVE et FACE
04FC		FLAGIF	flag "IF" b7=1 si IF en cours
04FD		ERROR	numéro de l'erreur
04FE/04FF		NOLIGN	numéro de la ligne de l'erreur

Page #C0

C000-	00	DRIVE	numéro du lecteur actif
C001-	0B	PISTE	numéro de piste (b7=1 si face B)
C002-	06	SECTEUR	numéro du secteur
C003-	00 C2	RWBUF	adresse de chargement du secteur
C005-	88		type d'erreur (b5 à 0 = WRITE, à 1 = READ FAULT)
C006-	02 08 07		utilisés dans la routine XRWTS
C009-	00	DRVDEF	numéro du lecteur par défaut
C00A-	00	DRVSYS	numéro du lecteur système
C00B-	00 0B		activation drive et piste
C00D-	00 00	EXTER	adresse messages d'erreur externe
C00F-	00 00	EXTMS	adresse messages externes
C011-	00 00		valeur qu'avait TXTPTR avant STRUN
C012-	01 00		valeur du n° de ligne avant STRUN
C015-	00	EXTNB	numéro du bloc externe (banque active)
C016-	00		flag banque changée
C017-	00		n° de l'"I/O ERROR"
C018-	00		flag ERR (b7 à 1 si SET, à 0 si OFF)
C019-	00 00		adresse de gestion de l'erreur (exemple FF37)
C01B-	20 20	ERRGOTO	n° ligne BASIC où il faut reprendre après erreur
C01D-	85 D6	ERRVEC	adresse de traitement des erreurs (D685 par ex)
C01F-	35 00	SVTPTR	sauvegarde TXTPTR (pointeur tampon clavier)
C021-	20 20		sauvegarde du pointeur de tampon clavier
C023-	FB	SAUVES	sauvegarde pointeur de pile (si erreur)
C024-	80	ATMORI	#00 (ROM V 1.0) ou #80 (ROM V 1.1)
C025-	14	POSNMP	piste du nom cherché dans le catalogue
C026-	04	POSNMS	secteur du nom cherché dans le catalogue
C027-	F0	POSNMX	position dans ce secteur de catalogue

BUFNOM

C028-	00	drive	préfixe de BUFNOM: n° du drive
C029-	53 4D 20 20 20 20 20 20 20		nom en 9 caractères (dernière lettre en C031)
C032-	43 4F 4D		extension en 3 caractères (dernière lettre en C034)
C035-	0B 0A	PSDESP	coordonnées du secteur de descripteur principal
C037-	02 40	NSTOTP	nombre de secteurs totaux + PROT/UNPROT
			NB: les 16 octets C029 à C038 = une ligne de catalogue
C039-	AD AD AD 00	TABDRV	table d'activation des lecteurs
C03D-	40	MODCLA	mode clavier (b6=ACCENT, b7=AZERTY)
C03E-	64 00	DEFNUM	origine par défaut (NUM, RENUM)
C040-	0A 00	DEFPAS	pas par défaut (NUM, RENUM)
C042-	64 00	TRAVNUM	n° de ligne (nombre sur 2 octets) (NUM, RENUM)
C044-	0A 00	TRAVPAS	"pas" de numérotation utilisé (NUM, RENUM)
C046-	0D		sauve A = code ASCII corresp à la touche
C047-	09		sauve X = nombre de caractères dans buffer entrée
C048-	00		type de code de fonction:
			b6=0 si commande Sédoric (RAMOV visée)
			b6=1 si commande BASIC (ROM visée)
			b7=0 si commande redéfinissable ou prédéfinie
			b7=1 dans tous les autres cas
C049-	00		b7=0 si code ASCII normal
			b7=1 si code de fonction en cours
C04A-	00		b7 selon point entrée dans s/p D843/D845 RAMOV
C04B-	00		première lettre d'un mot-clé Sédoric
			ou nombre de secteurs par piste
C04C-	20	DEFAFF	code ASCII devant les nombres décimaux
C04D-	00	VSALO0	code pour SAve/LOad b6=1 si ",V" b7=1 si ",N"
C04E-	00	VSALO1	code pour SAve/LOad b6=1 si ",A" b7=1 si ",J"
C04F-	3B 1A	LGSALO	longueur du fichier (FISALO - DESALO)
C051-	41	FTYPE	type du fichier chargé (voir manuel p 100)
C052-	00 50	DESALO	adresse de DEbut du fichier
			nombre de fiches d'un fichier à accès direct **D**

054-	FF B3	FISALO	adresse de FIn du fichier
			longueur de fiches d'un fichier à accès direct **D**
056-	00 50	EXSALO	adresse d' EXécution du fichier
058-	00 00	NSRSAV	nombre de secteurs restant à sauver
			nombre de secteurs supplémentaires requis
05A-	01 00	NSSAV	nombre de secteurs à sauver
05C-	0B 0A	PSDESC	coordonnées piste/secteur du 1er secteur descripteur
			ou de l'avant-dernier secteur de catalogue
05E-	01	NSDESC	nombre de secteurs descripteurs utilisés
05F-	0E	PTDESC	pointeur dans le secteur descripteur (BUF1)
060-	0F 00 05 01 E2 26		RWBUF lors de l'activation du DRIVE visé
066-	23 DE 80		ces 4 séquences,
069-	23 DE 80		concernent les 4 routines
06C-	23 DE 80		définies
06F-	23 DE 80		par la commande USER
072-	7F		flag LOAD: AUTO si b7=1, STOP si b7=0 ou
			flag DEL si b7=0, DELBAK ou DESTROY si b7=1
			flag BACKUP "monodrive" (à 1 si monodrive)
			flag pour lecture du code foreground/background (LINE et BOX)
073-	01		flag pour affichage de DEFFAFF (affichage d'un nombre décimal)
			flag BACKUP "source in drive" (à 1 si en place)
074-	00		flag BACKUP "format" (à 1 si formatage demandé)
075-	2E		sauvegarde de "caractère" pour LINPUT
076/C07F			"Général Field Buffer" (entrée courante du "Field Buffer") dont:
076/C07A	00 00 00 00 00		nom du champ (5 caractères significatifs)
07B-	00		index de l'élément de pseudo-tableau
07C-	00		n° logique pour ce champ
07D-	00		offset début de la fiche à début de ce champ
			index du début du champ dans l'enregistrement
			longueur totale des champs du "Field Buffer"
07E-	00		longueur du champ
			(1 si octet, 2 si entier, 5 si réel, l si alphanumérique)
07F-	00		type de champ
			(#00 réel, #01 entier, #40 octet, #80 alphanumérique)
080-	00		sauvegarde du n° logique de la dernière commande FIELD
081-	00		compteur de longueur totale des champs du "Field Buffer"
			puis #01, #40 ou #80 (si CLOSE)
082-	00		flag mis à #80 lors de CLOSE
			flag du point d'entrée du s/p 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
083-	00		HH de l'adresse de Buffer
			longueur d'une fiche (OPEN R) ou #00 (OPEN S ou OPEN D)
084-	00		pointeur dans le dernier descripteur
085/08/09			nombre d'octets précédant la fiche dans le fichier (sur 3 octets)
085-	00		rang de l'octet de début de la fiche dans le secteur
086-	00		index dans la liste des coordonnées du descripteur courant
087-	00		n° du descripteur courant
088-	00		index dans le buffer lu de ou à écrire sur la disquette
089-	00 00		DEBBAS (vise le lien de la première ligne) (SEEK)
08B-	00		longueur de la chaîne à chercher (SEEK)
08C-	00		position dans liste des coordonnées pour COPY
08D-	00 00		nombre de secteurs restant à charger par COPY
08F-	00		position de pointeur pour gestion des fichiers

090-	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00	NFA "Source" pour COPY*
09D-	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00	NFA "Cible" pour COPY*
0AA-	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00	Zone de 79 caractères
0B8-	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00	pour stocker
0C6-	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00	la chaîne à chercher
0D4-	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00	par la commande
0E2-	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00	SEEK
0F0-	00 00 00 00 00 00 00 00 00	(de C0AA à C0F8)

0F9-	00	drive source pour BACKUP
0FA-	00	drive cible pour BACKUP
0FB-	00 00	nombre de secteurs par face restant à BACKUPer
0FD-	00	HH de la taille du tampon de BACKUP (#92 ou #AF)
0FE-	00 00	inutilisés

C100/C1FF	BUF1	en général, buffer pour descripteur
C200/C2FF	BUF2	en général, buffer pour bitmap
C300/C3FF	BUF3	en général, buffer pour page de directory

BUFFER 1 (BUF1) #C100 À #C1FF
BUFFER DE LECTURE/ÉCRITURE D'UN SECTEUR

Utilisation universelle, voici par exemple le secteur 1 de la piste 20:

```
      0  1  2  3  4  5  6  7  8  9  A  B  C  D  E  F  0123456789ABCDEF
100- 2A 2A 2A 00 40 64 00 0A 00 94 41 6E 64 72 7B 20  ***.@d....André
110- 43 68 7B 72 61 6D 79 20 7A 7A 7A 7A 20 90 50 52  Chéramy zzzz .PR
120- 49 4E 54 22 42 6F 6E 6A 6F 75 72 2C 20 76 6F 69  INT"Bonjour, voi
130- 63 69 20 6C 65 20 53 7B 64 6F 72 69 63 20 6E 6F  ci le Sédoric no
140- 75 76 65 61 75 21 22 3A 50 49 4E 47 3A 50 52 49  uveau!":PING:PRI
150- 4E 54 22 53 61 6C 75 74 21 22 00 00 00 00 00 00  NT"Salut!"......
160- 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
170- 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
180- 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
190- 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
1A0- 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
1B0- 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
1C0- 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
1D0- 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
1E0- 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
1F0- 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
```

Le Secteur Système (secteur 1 de la piste 20) est structuré ainsi:

100-	octets n°00/03	table des drives (contient le nombre de pistes)
104-	octets n°04	type de clavier (b6=1 si ACCENT SET et b7=1 si AZERTY)
105-	octets n°05/06	départ de RENUM (ici #0064 = 100)
107-	octets n°07/08	"pas" de RENUM (ici #0000A = 10)
109-	octets n°09/1D	nom de la disquette (ici #94 papier bleu, #90 noir)
11E-	octets n°1E/59	INIST, instructions exécutées au démarrage (60 octets)
15A-	octets n°5A/FF	non utilisés (#00) (l'INIST aurait pu être plus long)

Le BUF1 est également utilisé pour lire les secteurs de descripteurs.

Le 1er secteur de descripteur chargé dans BUF1 est structuré ainsi:

100-	octets n°00/01	"lien" (coordonnées du descripteur suivant)
102-	octet n°02	contient #FF (seulement si 1er secteur descripteur) (Le pointeur X est positionné sur ce #FF, et permet de lire la suite)
103-	octet n°03	(C101+X) type de fichier (voir manuel page 100)
104-	octets n°04/05	(C102+X et C103+X) adresse (normale) de début (ou nombre de fiches pour un fichier à accès direct)
106-	octets n°06/07	(C104+X et C105+X) adresse (normale) de fin (ou longueur d'une fiche pour un fichier à accès direct)
108-	octets n°08/09	(C106+X et C107+X) adresse d'exécution si AUTO
10A-	octets n°0A/0B	(C108+X et C109+X) nombre de secteurs à charger
10C-	octets n°0C/FF	(C100+Y et C101+Y) liste coordonnées piste/secteur des secteurs à charger soit 122 paires de 2 octets. Si le nombre de secteurs à charger dépasse 122, lorsque Y atteint #00 (fin BUF1), il faut charger le descripteur suivant dont la structure est simplifiée:

100-	octets n°00/01	"lien" (coordonnées du descripteur suivant)
102-	octets n°02/FF	(C100+Y et C101+Y) liste des coordonnées piste/secteur des secteurs à charger (Y de #02 à #EF maxi), 127 paires de 2 octets. Si le nombre de secteurs à charger dépasse 122 + 127 = 249, il faut charger le descripteur suivant etc...

Pour les fichiers "mergés"

Le "lien" du dernier descripteur de chaque fichier indique les coordonnées du 1er 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 "mergé". 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. Le 1er descripteur de chaque fichier,

dont les coordonnées figurent dans l'"entrée" du catalogue, est le descripteur "principal".

BUFFER 2 (BUF2) #C200 À #C2FF
BUFFER DE LECTURE/ÉCRITURE DU SECTEUR DE BITMAP

Voici un exemple de bitmap, le secteur 2 de la piste 20:

```
     0  1  2  3  4  5  6  7  8  9  A  B  C  D  E  F
200- FF 00 6A 04 0C 00 2A 10 01 AA 00 00 00 00 00 00
210- 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
220- 00 00 00 00 00 00 00 00 00 C0 FF FF FF FF FF FF
230- FF FF FF FF FF FF FF FF B0 6D FF FF FF FF FF FF
240- FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF
250- FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF
260- FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF
270- FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF
280- FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF
290- FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF
2A0- FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF
2B0- FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF
2C0- FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF
2D0- FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF
2E0- FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF
2F0- FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF
```

Le Secteur de Bitmap (secteur 2 de la piste 20) est structuré ainsi:

200-	octets n°00	contient toujours #FF
201-	octets n°0	contient toujours #00
202-	octets n°02/03	nombre de secteurs libres (ici #046A = 1130)
204-	octets n°04/05	nombre de fichiers (ici #000C = 12)
206-	octets n°06	nombre de pistes/face (ici #2A = 42)
207-	octets n°07	nombre de secteurs/piste (ici #10 = 16)
208-	octets n°08	nombre de secteurs de directory (ici #01 = 1)
209-	octets n°09	copie de l'octet n°06 dont on a ajusté le b7 à 0 si simple face ou à 1 si double face (en principe, car hélas c'est "la" bogue)
20A-	octets n°0A	#00 si Master, #01 si Slave ou #47 ("G") si Games
20B-	octets n°0B/0F	contient toujours #00 (non utilisés)
210-	octets n°10/FF	Bitmap: chaque bit représente un secteur. Ce secteur est libre si le bit corresp est à 1 ou occupé s'il est à 0. Les bits de chaque octet sont lus de droite à gauche (sens b0 -> b7), mais les octets sont lus de gauche à droite (sens octet n°#10 -> n°#FF)

Par exemple, on voit ici que la plupart des secteurs sont libres (octets à #FF, c'est à dire à 1111 1111). Mais les octets n°38 et 39 indiquent #B0 et #6D. Il s'agit des 16 secteurs de la piste n°20. Les 8 premiers secteurs sont répertoriés par l'octet n°38 et on a #B0 = 1011 0000 qui indique que les secteurs n°1 à 4 sont occupés (les bits b0 à b3 sont à 0), les secteurs n°5 et 6 sont libres (les bits b4 et b5 sont à 1) le secteur n°7 est occupé (bit b6 à 0) et le secteur n°8 est libre (bit b7 à 1). Les 8 derniers octets sont répertoriés par l'octet n°39 où l'on a #6D = 0110 1101 qui indique que les secteurs n°10, 13 et 16 sont occupés, alors que les octets n°9, 11, 12, 14 et 15 sont libres. Cet exemple était simplifié par le fait que cette disquette étant formatée à 16 secteurs par piste, chaque piste est représentée par une paire d'octets.

Certains secteurs sont déjà occupés au départ, ce sont:

-Les secteurs situés au début de la disquette et dont le nombre varie selon qu'il s'agit d'une disquette Master (94), Slave (8) ou de type "G" (17) (Shortsed initialisé par GAMEINIT)

-Le Secteur Système (secteur 1 de la piste 20) voir ci-dessus
-Le Secteur Bitmap (secteur 2 de la piste 20) voir ci-dessus
-Le secteur 3 de la piste 20 qui ne contient que des #00 (non utilisés)
-Les secteurs de catalogues (secteurs 4, 7, 10, 13 et 16 de la piste 20) voir ci-dessous qui sont marqués "occupés" (réservés) mais contiennent des #00 tant qu'ils ne sont pas réellement utilisés

BUFFER 3 (BUF3) #C300 À #C3FF
BUFFER DE LECTURE/ÉCRITURE D'UN SECTEUR DE DIRECTORY

Voici un exemple de secteur de catalogue, le secteur 4 de la piste 20:

```
        0  1  2  3  4  5  6  7  8  9  A  B  C  D  E  F  0123456789ABCDEF
C300- 00 00 D0 00 00 00 00 00 00 00 00 00 00 00 00 00
C310- 44 45 42 55 54 53 45 44 4F 4E 45 57 05 0F 04 C0  DEBUTSEDONEW...
C320- 54 32 30 53 31 32 34 20 20 4E 45 57 06 03 04 C0  T20S124  NEW...
C330- 52 41 4D 4F 56 20 20 20 20 4E 45 57 06 07 41 C0  RAMOV    NEW..A
C340- 52 41 4D 4D 4F 56 45 52 20 43 4F 4D 0A 08 02 C0  RAMMOVER COM...
C350- 50 34 20 20 20 20 20 20 20 43 4F 4D 0A 0E 02 C0  P4       COM...
C360- 52 41 4D 42 4B 31 20 20 20 43 4F 4D 0A 0A 05 40  RAMBK1   COM...@
C370- 52 41 4D 42 4B 32 20 20 20 43 4F 4D 0B 01 05 40  RAMBK2   COM...@
C380- 52 41 4D 42 4B 33 20 20 20 43 4F 4D 0B 06 05 40  RAMBK3   COM...@
C390- 52 41 4D 42 4B 34 20 20 20 43 4F 4D 0B 0B 05 40  RAMBK4   COM...@
C3A0- 52 41 4D 42 4B 35 20 20 20 43 4F 4D 0B 10 05 40  RAMBK5   COM...@
C3B0- 52 41 4D 42 4B 36 20 20 20 43 4F 4D 0C 05 05 40  RAMBK6   COM...@
C3C0- 54 4F 54 4F 20 20 20 20 20 43 4F 4D 0C 0A 05 40  TOTO     COM...@
C3D0- 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
C3E0- 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
C3F0- 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
```

Un Secteur de Catalogue est structuré ainsi:

C300-	octets n°00/01	coordonnées (piste/secteur) du catalogue suivant (ici un seul secteur de catalogue est en service, il n'y a donc pas de suivant)
C302-	octets n°02	n° de l'octet de la 1ère entrée libre (#00 si plein) (ici la prochaine entrée libre se situe en C3**D0**)
C303-	octets n°03/0F	contient toujours #00 (inutilisés)
C310-	octets n°10/FF	15 "entrées" de catalogue (une ligne de 16 octets par entrée)

Chaque "entrée" de catalogue est structurée ainsi:

octets n°00 à 08	nom complété à droite par des espaces (#20)
octets n°09 à 0B	extension (idem)
octet n°0C	piste du descripteur
octet n°0D	secteur du descripteur (exemple, le descripteur de "TOTO.COM", se trouve au secteur n°#0A de la piste n°#0C)
octet n°0E	nombre de secteurs du fichier (y compris le(s) descripteurs) (le fichier "TOTO.COM" comprend 5 secteurs dont 1 descripteur et 4 secteurs de fichier proprement dit)
octet n°0F	attribut de protection (b6=1, PROT si b7=1, UNPROT si b7=0) (#40 = 0100 0000 pour UNPROT et #C0 = 1100 0000 pour PROT)

*** BANQUE n°0 (#C400 à #C7FF) ***

INITIALISATION SÉDORIC
(CETTE BANQUE SERA ÉCRASÉE PAR LA SUITE)

Entrée réelle?

400-	AD 07 C0	LDA C007	flag non identifié (C007 contient #08 = 0000 1000, mais on peut imaginer que dans certains cas il puisse valoir #01 = 0000 0001)
403-	4A	LSR	b0 -> C qui passe à zéro (mais peut-être à 1?)
404-	A9 00	LDA #00	A = 0000 0000
406-	6A	ROR	C passe dans le b7
407-	8D 24 C0	STA C024	mise à jour b7 de ATMORI selon b0 de C007
			NB: ATMORI vaut #00 si ROM V1.0 ou #80 si ROM V1.1
40A-	10 0F	BPL C41B	continue en C41B si ROM V1.0

ROM V1.1

40C-	A9 50	LDA #50	A = 80 caractères
40E-	8D 56 02	STA 0256	longueur ligne imprimante V 1.1
411-	4A	LSR	A = 40 caractères et Z = 0
412-	85 31	STA 31	longueur ligne imprimante V 1.0
414-	85 32	STA 32	position maxi pour tabulation par ","
416-	8D 57 02	STA 0257	longueur d'une ligne écran V 1.1
419-	D0 06	BNE C421	suite forcée en C421 car Z = 0 dans tous les cas

ROM V1.0

41B-	A9 5D	LDA #5D	A = 93 (bogue de la ROM V 1.0)
41D-	85 31	STA 31	longueur ligne imprimante V 1.0
41F-	85 32	STA 32	position maxi pour tabulation par ","

Mise en place de la page 4

421-	EE C1 02	INC 02C1	LL de HIMEM soit #00 -> #01
424-	EE C2 02	INC 02C2	HH de HIMEM soit #98 -> #99, HIMEM = #9901
427-	A2 00	LDX #00	mise en place de la page 4: RAZ index
429-	BD 00 C6	LDA C600,X	pointe par défaut sur version Oric-1
42C-	2C 24 C0	BIT C024	teste ATMORI: = #80 si V 1.1 N = 1 (négatif)
42F-	10 03	BPL C434	si N = 0 (version Oric-1) Ok on copie
431-	BD 00 C7	LDA C700,X	sinon on remplace par version Atmos
434-	9D 00 04	STA 0400,X	recopie en page 4
437-	E8	INX	compteur passera de #00 à #FF, soit 256 octets
438-	D0 EF	BNE C429	reboucle jusqu'à #FF inclus

Modification de la routine CHARGET en page 0

43A-	A9 4C	LDA #4C	
43C-	A0 00	LDY #00	
43E-	A2 04	LDX #04	
440-	85 EF	STA EF	remplace JSR ECB9 par JMP 0400
442-	84 F0	STY F0	
444-	86 F1	STX F1	

Modification vecteurs IRQ et NMI

446-	A9 88	LDA #88	
448-	A0 C4	LDY #C4	NB: X a encore la valeur #04
44A-	2C 24 C0	BIT C024	teste ATMORI: si V 1.1 N = 1 (négatif)
44D-	10 26	BPL C475	si Oric-1, continue en C475
44F-	8D 45 02	STA 0245	remplace JMP #EE22 (IRQ)
452-	8E 46 02	STX 0246	par JMP #0488 (new IRQ)
455-	8C 48 02	STY 0248	remplace JMP #F8B2 (NMI)
458-	8E 49 02	STX 0249	par JMP #04C4 (new NMI)

Modification s/p "Prendre un caractère au clavier"

45B-	A9 5B	LDA #5B	
45D-	8D 3C 02	STA 023C	remplace JMP #EB78
460-	8E 3D 02	STX 023D	par JMP #045B

Modification délai et vitesse d'autorépétition clavier

463-	A9 09	LDA #09	
465-	A0 01	LDY #01	
467-	8D 4E 02	STA 024E	délai: remplace #10 par #09
46A-	8C 4F 02	STY 024F	vitesse: remplace #04 par #01

```
        Modification paramètres pour mode console
C46D-   A9 0F       LDA #0F
C46F-   A2 70       LDX #70         adresse V 1.1 pour
C471-   A0 D0       LDY #D0         "SYNTAX ERROR"
C473-   D0 12       BNE C487        suite forcée en #C487

        Même chose pour ORIC-1
C475-   8D 29 02    STA 0229        remplace JMP #EC03 (IRQ)
C478-   8E 2A 02    STX 022A        par JMP #0488 (new IRQ)
C47B-   8C 2C 02    STY 022C        remplace JMP #F430 (NMI)
C47E-   8E 2D 02    STX 022D        par JMP #04C4 (new NMI)
C481-   A9 07       LDA #07         new paramètres pour mode console
C483-   A2 E4       LDX #E4         adresse V 1.0 pour
C485-   A0 CF       LDY #CF         "SYNTAX ERROR"

        Suite commune ORIC-1 et ATMOS
C487-   8D 6A 02    STA 026A        mode console Oric-1/Atmos
C48A-   8E F9 02    STX 02F9        nouveau vecteur "SYNTAX ERROR" Sédoric
C48D-   8C FA 02    STY 02FA        (#D070 pour Atmos et #CFE4 pour Oric-1)
C490-   A2 04       LDX #04
C492-   A9 A5       LDA #A5
C494-   A0 D0       LDY #D0
C496-   8D FE FF    STA FFFE
C499-   8C FF FF    STY FFFF        vecteur #D0A5 placé en #FFFE (sur RAMOV)
C49C-   A9 67       LDA #67
C49E-   A0 61       LDY #61
C4A0-   8D F5 02    STA 02F5        vecteur "!" redirigé sur #0467
C4A3-   8E F6 02    STX 02F6
C4A6-   8C FC 02    STY 02FC        vecteur "&()" redirigé sur #0461
C4A9-   8E FD 02    STX 02FD
C4AC-   A9 00       LDA #00         mise à zéro des variables suivantes:
C4AE-   8D 09 C0    STA C009        DRVDEF lecteur actif A par défaut
C4B1-   8D 0A C0    STA C00A        DRVSYS lecteur système A par défaut
C4B4-   8D 0B C0    STA C00B        activation drive
C4B7-   8D 0C C0    STA C00C        activation piste
C4BA-   8D 15 C0    STA C015        EXTNB numéro du bloc externe (banque 0)
C4BD-   8D 18 C0    STA C018        flag ERR OFF (b7 à 1 si ERR ON)
C4C0-   8D DF 02    STA 02DF        KEYBUF pas de touche pressée
C4C3-   8D 48 C0    STA C048        type de code de fonction
C4C6-   85 87       STA 87          pointeur de la pile des descripteurs
C4C8-   A9 85       LDA #85
C4CA-   A0 D6       LDY #D6
C4CC-   8D 1D C0    STA C01D        ERRVEC adresse de traitement des erreurs: #D685
C4CF-   8C 1E C0    STY C01E
C4D2-   AD 11 03    LDA 0311        (#310 à #31B I/O lecteur de disquette Oric)
C4D5-   8D 0C C0    STA C00C        activation piste

        Place une série de vecteurs "SYNTAX ERROR"
C4D8-   A9 23       LDA #23
C4DA-   A0 DE       LDY #DE         AY = adresse DE23 du vecteur "SYNTAX ERROR"
C4DC-   A2 80       LDX #80
C4DE-   8D 66 C0    STA C066
C4E1-   8C 67 C0    STY C067        C066/67/68 = DE23 vecteur "SYNTAX ERROR" et #80
C4E4-   8E 68 C0    STX C068
C4E7-   8D 69 C0    STA C069
C4EA-   8C 6A C0    STY C06A        C069/6A/6B = DE23 vecteur "SYNTAX ERROR" et #80
C4ED-   8E 6B C0    STX C06B
C4F0-   8D 6C C0    STA C06C
C4F3-   8C 6D C0    STY C06D        C06C/6D/6E = DE23 vecteur "SYNTAX ERROR" et #80
C4F6-   8E 6E C0    STX C06E
C4F9-   8D 6F C0    STA C06F
C4FC-   8C 70 C0    STY C070        C06F/70/71 = DE23 vecteur "SYNTAX ERROR" et #80
C4FF-   8E 71 C0    STX C071

        Caractère pour LINPUT
C502-   A9 2E       LDA #2E         caractère "."
C504-   8D 75 C0    STA C075        placé en C075

        Teste si Sédoric est bien en mémoire
C507-   A9 1A       LDA #1A
```

```
C509- A0 00       LDY #00       redirige le vecteur d'exécution
C50B- 8D F0 04    STA 04F0      sur #001A (imprimer chaîne AY)
C50E- 8C F1 04    STY 04F1
C511- A5 00       LDA 00        teste la mémoire à l'adresse 00
C513- F0 12       BEQ C527      si nulle, OK continue en #C527
C515- A2 FF       LDX #FF       sinon copie le message
C517- E8          INX           "** WARNING ** DOS is altered"
C518- BD 74 C5    LDA C574,X    (terminé par un 0)
C51B- 9D 00 B9    STA B900,X    au début de la zone
C51E- D0 F7       BNE C517      du jeu semi-graphique
C520- A9 00       LDA #00       puis l'affiche en utilisant le
C522- A0 B9       LDY #B9       vecteur #001A, c'est à dire la
C524- 20 EC 04    JSR 04EC      routine ROM #CCB0 (ou #CBED si Oric-1)

      Initialise TABDRV, MODCLA, DEFNUM et DEFPAS
C527- A9 14       LDA #14       piste #14 secteur #01
C529- A0 01       LDY #01       secteur système de la disquette Sédoric
C52B- 20 5D DA    JSR DA5D      XPBUF1 charge dans BUF1 le secteur Y de la piste A
C52E- A2 08       LDX #08       X = 8, compteur qui sera décrémenté
C530- BD 00 C1    LDA C100,X    copie les 4 premiers octets (n° 0 à 3) dans TABDRV:
C533- 9D 39 C0    STA C039,X    table d'activation des lecteurs (C039/3C)
C536- E0 05       CPX #05       le 5ème (n°4) dans MODCLA (C03D)
C538- 90 03       BCC C53D      les 4 derniers (n° 5 à 8) dans DEFNUM (C03E/3F)
C53A- 9D 3D C0    STA C03D,X    et DEFPAS (C040/41) ainsi que de C042 à C045
C53D- CA          DEX           (bravo les algorithmes clairs!)
C53E- 10 F0       BPL C530      reboucle tant que X est positif ou nul
C540- 20 A3 EB    JSR EBA3      redéfini les caractères normaux selon MODCLA
C543- 20 D8 D5    JSR D5D8      XROM exécute à partir de la RAM un s/p ROM
C546- E0 F7       F7E0          adresse ROM 1.0 puis adresse ROM 1.1:
C548- 16 F8       F816          Générer les caractères alternés

      Copie les instructions de démarrage dans le tampon clavier
C54A- A2 41       LDX #41       le secteur #01 de piste #14 est toujours dans BUF1
C54C- BD 1E C1    LDA C11E,X    lit les 66 octets n° #1E à #5F (de C11E à C15F)
C54F- 95 36       STA 36,X      et les copie dans KEYBUF de 0036 à #0077 inclus
C551- CA          DEX           octet suivant et reboucle tant que X pas négatif
C552- 10 F8       BPL C54C      (il s'agit du contenu de INIST + 6 octets à #00)
C554- A9 3A       LDA #3A       enfin ajoute #3A par-devant en 35 (début du TIB)
C556- 85 35       STA 35        c'est à dire ":"

      Exécute les instructions de démarrage
C558- 20 06 D2    JSR D206      CBF0/ROM va à la ligne
C55B- A9 BD       LDA #BD
C55D- A0 C4       LDY #C4       AY = adresse C4BD, interpréteur Atmos par défaut
C55F- 2C 24 C0    BIT C024      teste ATMORI (b7 à 1 si ATMOS)
C562- 30 02       BMI C566      les Atmos continuent en C566, merci pour eux!
C564- A9 CD       LDA #CD       AY = adresse C4CD, interpréteur Oric-1
C566- 8D F0 04    STA 04F0      place cette adresse AY (s/p ROM entrée interpréteur)
C569- 8C F1 04    STY 04F1      dans EXEVEC (vecteur exécution)
C56C- A2 34       LDX #34
C56E- A0 00       LDY #00       pour ajuster TXTPTR à 0034
C570- 58          CLI           autorise les interruptions
C571- 4C 71 04    JMP 0471      et continue selon EXEVEC (Interpréteur BASIC)
                                C'est parti mon kiki, maintenant ça tourne sous Sédoric!

      MESSAGE: DOS IS ALTERED!
C574- 0A 8C 81                                        LF TEXTE CLIGNOTANT ENCRE ROUGE
C577- 2A 2A 20 57 41 52 4E 49 4E 47 20 2A 2A          ** WARNING **
C584- 88 87                                           TEXTE NORMAL ENCRE BLANCHE
C586- 44 4F 53 20 69 73 20 61 6C 74 65 72 65 64 20 21 DOS is altered !
C596- 0D 0A 00                                        RETURN LF FIN

      Ceci est un résidu de fausse couche!
C599- 4C 64 D3    JMP D364      XAFSC affiche X+1ème mess externe terminé par "caractère + 128"
C59C- 60          RTS           et retourne (mais s/p appelé de nulle part!)
C59D- AD AE C5    LDA C5AE      A = #27 (idem: s/p appelé de nulle part!)
C5A0- AE AF C5    LDX C5AF      X = #09
C5A3- 8D 01 C0    STA C001      PISTE (n° de piste pour I/O)
C5A6- 8E 02 C0    STX C002      SECTEUR (n° de secteur pour I/O)
C5A9- AD B0 C5    LDA C5B0      A = #1A
```

```
C5AC-  D0 DB     BNE C589     branchement forcé vers un non sens
C5AE-  27 09 1A  DATA
```

DIVERS MESSAGES

(Le numéro d'ordre X est indiqué)

```
5B1-  49 4E 20 44 52 49 56 45 A0
      IN DRIVE■
5BA-  4C 4F 41 44 20 44 49 53 43 53 20 46 4F 52 20 42 41 43 4B 55 50 20 46 52 4F 4D A0
      LOAD DISCS FOR BACKUP FROM■
5D5-  20 54 4F A0
      ■TO■
5D9-  0D 0A 4C 4F 41 44 20 53 4F 55 52 43 45 20 44 49 53 43 A0
      CR LF LOAD SOURCE DISC■
5EC-  0D 0A 4C 4F 41 44 20 54 41 52 47 45 54 20 44 49 53 53 2C 28
      CR LF LOAD TARGET DISS,(
```

Ces 103 octets (C599 à C5FF), malheureusement situés à un endroit précaire, sont disponibles. Il s'agit d'un reliquat de la banque n°2 (commande BACKUP) qui n'a pas été effacé au cours de la mise au point et inutilement sauvé avec la banque n°0. En fait, les 3 derniers octets de cette page sont peut-être utilisés. en effet, ils sont en surimpression des messages de la banque n°2 où ils valaient #43, #A0 et #0D ("C", LF et CR). Ici, ils valent #53, #2C et #28, mais peuvent prendre les valeurs #41, #F1 et #2B ou, sur la disquette du Sédoric V2.0GB, les valeurs #41, #2D et #2B ou #53, #52 et #28 pour Stratoric V1.0. Il peut s'agir d'une zone DATA utilisée pendant le BOOT et écrasée par la suite.

Sédoric V2.0 GB

L'octet en C5FE a été changé et prend la valeur #2D, mais la signification de ce changement n'est pas encore connue.

BANQUE n°0, 3ème secteur: source de la page 4 version Oric-1

```
600- C9 30 90 04 C9 3A 90 35 86 0F AA 30 2E 85 0E 68   bogue "CSAVE"
610- AA 68 48 E0 F7 D0 04 C9 C8 F0 09 E0 58 D0 18 C9
620- CA D0 14 24 18 6E FC 04 A0 FF C8 B1 E9 F0 11 C9
630- 3A F0 0D C9 D4 D0 F3 8A 48 A5 0E A6 0F 4C 41 EA   bogue "CSAVE"
640- 68 20 E9 04 20 67 04 0E FC 04 B0 03 4C AD C8 EA
650- EA EA 60 20 77 04 B1 16 4C 77 04 EA EA EA EA EA
660- EA A9 8E A0 F8 D0 04 A9 AE A0 D3 8D F0 04 8C F1
670- 04 20 77 04 20 EF 04 08 48 78 AD FB 04 49 02 8D
680- FB 04 8D 14 03 68 28 60 2C 0D 03 50 0F 48 A9 04
690- 2D 6A 02 F0 03 EE 74 02 68 4C 03 EC 68 68 85 F2
6A0- 68 AA A9 36 A0 D1 D0 C3 20 F2 04 68 40 8D 14 03
6B0- 6C FC FF 18 20 77 04 48 A9 04 48 A9 A8 48 08 B0
6C0- 03 4C 28 02 20 88 F8 A9 17 A0 EC 20 6B 04 4C 75
6D0- C4 A9 04 48 A9 F1 48 8A 48 98 48 20 F2 04 4C 70
6E0- D2 EA EA EA EA EA EA EA EA 4C 87 04 4C 71 04 4C
6F0- 00 00 4C 77 04 4C B3 04 4C B4 04 84 00 00 00 00
```

BANQUE n°0, 4ème et dernier secteur: source de la page 4 version Atmos

```
700- C9 30 90 04 C9 3A 90 35 86 0F AA 30 2E 85 0E 68   bogue "CSAVE"
710- AA 68 48 E0 0E D0 04 C9 C9 F0 09 E0 8A D0 18 C9
720- CA D0 14 24 18 6E FC 04 A0 FF C8 B1 E9 F0 11 C9
730- 3A F0 0D C9 D4 D0 F3 8A 48 A5 0E A6 0F 4C B9 EC   bogue "CSAVE"
740- 68 20 E9 04 20 67 04 0E FC 04 B0 03 4C C1 C8 6E
750- 52 02 60 20 77 04 B1 16 4C 77 04 A9 45 A0 D8 D0
760- 0A A9 8E A0 F8 D0 04 A9 AE A0 D3 8D F0 04 8C F1
770- 04 20 77 04 20 EF 04 08 48 78 AD FB 04 49 02 8D
780- FB 04 8D 14 03 68 28 60 2C 0D 03 50 0F 48 A9 04
790- 2D 6A 02 F0 03 EE 74 02 68 4C 22 EE 68 68 85 F2
7A0- 68 AA A9 36 A0 D1 D0 C3 20 F2 04 68 40 8D 14 03
7B0- 6C FC FF 18 20 77 04 48 A9 04 48 A9 A8 48 08 B0
7C0- 03 4C 44 02 20 B8 F8 A9 17 A0 EC 20 6B 04 4C 71
7D0- C4 A9 04 48 A9 F1 48 8A 48 98 48 20 F2 04 4C 06
7E0- D3 EA EA EA EA EA EA EA EA 4C 87 04 4C 71 04 4C
7F0- 00 00 4C 77 04 4C B3 04 4C B4 04 84 00 00 00 00
```

DÉSASSEMBLAGE DE LA PAGE 4 SÉDORIC

Complément de l'interpréteur basic (en fait s'intercale entre la fin du s/p 00E2 et le début du s/p ECB9)

0400-	C9 30	CMP "0"	si A est un chiffre (de 0 à 9)
0402-	90 04	BCC 0408	on retourne à ECB9 en ROM
0404-	C9 3A	CMP ":"	sinon on continue en 0408
0406-	90 35	BCC 043D	
0408-	86 0F	STX 0F	les registres A et X sont
040A-	AA	TAX	sauvegardés en 0E et 0F
040B-	30 2E	BMI 043B	si A est négatif (token BASIC)
040D-	85 0E	STA 0E	récupère A et X et continue en ECB9 (bogue "CSAVE")
040F-	68	PLA	prend adresse sur pile
0410-	AA	TAX	A (HH) et X (LL)
0411-	68	PLA	
0412-	48	PHA	remet HH sur la pile
0413-	E0 0E	CPX #0E	
0415-	D0 04	BNE 041B	si adresse était C90E met à 0
0417-	C9 C9	CMP #C9	le b7 du drapeau 04FC
0419-	F0 09	BEQ 0424	et continue en 0428
041B-	E0 8A	CPX #8A	
041D-	D0 18	BNE 0437	si adresse était CA8A met à 1
041F-	C9 CA	CMP #CA	le b7 du drapeau 04FC
0421-	D0 14	BNE 0437	et continue en 0428
0423-	24 18	BIT 18	continue en 0425
0424-	18	CLC	
0425-	6E FC 04	ROR 04FC	si ni C90E ni CA8A continue en 0437
0428-	A0 FF	LDY #FF	
042A-	C8	INY	s'il y a un "=" d"ici la fin
042B-	B1 E9	LDA (E9),Y	de la commande (marquée par
042D-	F0 11	BEQ 0440	"0" ou par ":") il s'agit
042F-	C9 3A	CMP "0"	d'une affectation basic
0431-	F0 0D	BEQ 0440	on remet LL sur la pile
0433-	C9 D4	CMP ":"	(HH y est déjà)
0435-	D0 F3	BNE 042A	on récupère les valeurs
0437-	8A	TXA	d'origine de A et X
0438-	48	PHA	et on retourne en ECB9
0439-	A5 0E	LDA 0E	bogue "CSAVE"
043B-	A6 0F	LDX 0F	sinon on continue en 0440
043D-	4C B9 EC	JMP ECB9	
0440-	68	PLA	on retire HH de la pile
0441-	20 E9 04	JSR 04E9	vers vecteur utilisateur
0444-	20 67 04	JSR 0467	vers vecteur "!" Sédoric
0447-	0E FC 04	ASL 04FC	teste b7 du flag 04FC
044A-	B0 03	BCS 044F	s'il est à 0 continue en C8C1
044C-	4C C1 C8	JMP C8C1	(sous-prog ROM exécuter ligne)
044F-	6E 52 02	ROR 0252	sinon met à 1 le b7 du flag
0452-	60	RTS	"IF" (0252) et fin s/p 0400

NB: Lorsque le s/p 00E2 est appelé au point C90C, l'adresse de retour-1 C90E est empilée, s'il est appelé en C4C1, C4C3 est empilé. Enfin, lorsque 00E8 est appelé en CA88, à partir du s/p "IF", l'adresse CA8A est empilée.

Gestion du vecteur d'exécution

0453-	20 77 04	JSR 0477	entrée appelée de la RAMOV pour lecture dans
0456-	B1 16	LDA (16),Y	la ROM à l'adresse pointée en 16/17 + Y puis
0458-	4C 77 04	JMP 0477	retour sur la RAMOV
045B-	A9 45	LDA #45	s/p EB78 modifié
045D-	A0 D8	LDY #D8	initialise pour exécution
045F-	D0 0A	BNE 046B	en D845 (Prendre un caractère au clavier)
0461-	A9 8E	LDA #8E	entrée vecteur "&()"
0463-	A0 F8	LDY #F8	initialise pour exécution
0465-	D0 04	BNE 046B	en F88E
0467-	A9 AE	LDA #AE	entrée vecteur "!" initialise
0469-	A0 D3	LDY #D3	pour exécution en D3AE
046B-	8D F0 04	STA 04F0	mise à jour du l'adresse du
046E-	8C F1 04	STY 04F1	s/p à exécuter (EXEVEC+1)
0471-	20 77 04	JSR 0477	entrée s/p EXERAM: bascule
0474-	20 EF 04	JSR 04EF	ROM/RAMOV, vecteur EXEVEC
0477-	08	PHP	entrée s/p bascule ROM/RAMOV
0478-	48	PHA	(n'affecte aucun registre)
0479-	78	SEI	
047A-	AD FB 04	LDA 04FB	**il existe une autre version (celle de Stratoric)**:
047D-	49 02	EOR #02	LDA 0321 EOR #06

```
-7F-  8D FB 04   STA 04FB     STA 0321
-82-  8D 14 03   STA 0314     PLA PLP RTS
-85-  68         PLA          qui est plus courte et se
-86-  28         PLP          termine en 0484, la zone
-87-  60         RTS          0485 à 0487 est donc libre
                              NB: ce s/p permet d'exécuter les routines F590, D3AE, D136, EC17, F88E, D845
      Nouvel IRQ (remplace EE22 de la ROM)
488-  2C 0D 03   BIT 030D     (s/p utilisé à partir de la ROM)
-8B-  50 0F      BVC 049C
-8D-  48         PHA
-8E-  A9 04      LDA #04
-90-  2D 6A 02   AND 026A     mode console Oric-1/Atmos
-93-  F0 03      BEQ 0498
-95-  EE 74 02   INC 0274     clignotement curseur
498-  68         PLA
-99-  4C 22 EE   JMP EE22     on est bien sur la ROM
49C-  68         PLA
-9D-  68         PLA
-9E-  85 F2      STA F2
-A0-  68         PLA
-A1-  AA         TAX
-A2-  A9 36      LDA #36      initialisation pour exécution
-A4-  A0 D1      LDY #D1      du s/p D136 en RAMOV
-A6-  D0 C3      BNE 046B     avec retour sur la ROM
      Nouveau COLDSTART (remplace FFFC et donc F88F)
4A8-  20 F2 04   JSR 04F2     bascule ROM/RAMOV
-AB-  68         PLA          dépile
-AC-  40         RTI          retour d'interruption
-AD-  8D 14 03   STA 0314     flag ROM/RAMOV
-B0-  6C FC FF   JMP (FFFC)   vecteur coldstart ROM
      s/p IRQRAM et NMIRAM
4B3-  18         CLC          entrée IRQRAM (04B4 = entrée NMIRAM)
4B4-  20 77 04   JSR 0477     bascule ROM/RAMOV
-B7-  48         PHA          empile A
-B8-  A9 04      LDA #04
-BA-  48         PHA
-BB-  A9 A8      LDA #A8
-BD-  48         PHA          empile adresse 04A8 (coldstart)
-BE-  08         PHP          empile indicateurs d'état 6502
-BF-  B0 03      BCS 04C4     si NMI saute en 04C4
-C1-  4C 44 02   JMP 0244     si IRQ continue en 0244 soit en 0488
4C4-  20 B8 F8   JSR F8B8     nouvel NMI: en ROM F8B8 au lieu
-C7-  A9 17      LDA #17      de F8B2 (saute warmstart BASIC)
-C9-  A0 EC      LDY #EC
-CB-  20 6B 04   JSR 046B     exécute s/p EC17 sur RAMOV
-CE-  4C 71 C4   JMP C471     exécute enfin warmstart BASIC
      Chercher un tableau (lorsqu'on est en RAMOV)
-D1-  A9 04      LDA #04
-D3-  48         PHA          empile 04F1 adresse de retour -1
-D4-  A9 F1      LDA #F1      (le retour se fera en 04F2,
-D6-  48         PHA          c'est à dire bascule ROM/RAMOV)
-D7-  8A         TXA
-D8-  48         PHA          sauve X sur la pile
-D9-  98         TYA
-DA-  48         PHA          sauve Y sur la pile
-DB-  20 F2 04   JSR 04F2     bascule ROM/RAMOV (ici, accède à la ROM)
-DE-  4C 06 D3   JMP D306     trouver le tableau et retour en RAMOV
      Vecteurs et Drapeaux
-E1-  EA EA EA EA EA EA EA EA NOP NOP NOP NOP NOP NOP NOP NOP
4E9-  4C 87 04   JMP 0487     DETE2C vecteur utilisateur
4EC-  4C 71 04   JMP 0471     EXERAM exécution de s/p indiqué à EXEVEC+1  sur RAMOV (si ROM active) ou sur
      ROM (si RAMOV active)
4EF-  4C 00 00   JMP 0000     EXEVEC vecteur exécution, dont..
4F0-  00 00                   adresse exécution
4F2-  4C 77 04   JMP 0477     RAMROM bascule RAMOV/ROM
4F5-  4C B3 04   JMP 04B3     IRQRAM exécution IRQ sur RAM
4F8-  4C B4 04   JMP 04B4     NMIRAM exécution NMI sur RAM
4FB-  94                      flag ROM/RAMOV
4FC-  00         BRK          flag C90E/CA8A
4FD-  00         BRK          numéro de l'erreur
4FE-  00 00      BRK          numéro de la ligne de l'erreur
```

0500- 00 BRK début BASIC

*** BANQUES INTERCHANGEABLES ***

(localisées de C400 à C7FF)

Banque n°1: RENUM, DELETE et MOVE
Banque n°2: BACKUP
Banque n°3: SEEK, CHANGE et MERGE
Banque n°4: COPY
Banque n°5: SYS, DNAME, DTRACK, TRACK,INIST, DNUM, DSYS, DKEY et VUSER
Banque n°6: INIT

Banque n°1 (adresse Cxxxa): RENUM, DELETE et MOVE

Cette banque se trouve à partir du #42 (66ème) secteur de la disquette MASTER

400a	00 00	EXTER	Adresse des messages d'erreur externe (néant)
402a	00 00	EXTMS	Adresse des messages externes (néant)
404a	4C 32 C4	JMP C432	Entrée commande RENUM
407a	4C EC C6	JMP C6EC	Entrée commande DELETE
40Aa	4C 56 C7	JMP C756	Entrée commande MOVE

EXECUTION COMMANDE SEDORIC RENUM

Rappel de la syntaxe
RENUM (NEWNUM)(,NEWPAS)(,PRELGN)(,DERLGN)

Renumérote les lignes du bloc commençant à PRELGN et se terminant à DERLGN inclue en utilisant NEWNUM comme premier nouveau numéro de ligne et NEWPAS comme nouveau pas de numérotation. Cette commande utilise des valeurs par défaut pour remplacer les paramètres éventuellement omis: DEFNUM (100), DEFPAS (10), DEFPRE (0 ou à défaut, première ligne du programme) et DEFDER (dernière ligne du programme jusqu'à 65534). DEFNUM et DEFPAS peuvent être modifiés par la commande DNUM.

Les instructions GOTO, GOSUB, ON GOTO, ON GUSUB, THEN, ELSE, RUN, RESTORE sont mises à jour en fonction des nouveaux numéros de lignes. Lorsqu'une ligne n'est pas trouvée, RENUM prend la suivante. Attention, RENUM ne met pas à jour les n° de lignes exprimés sous forme de variables ou d'expressions numériques.

Variables utilisées

9/01	copie de DEBBAS (9A/9B), début programme BASIC, pointe sur le 1er lien
2/03	copie de FINBAS (9C/9D), fin du programme BASIC
4/05	copie de HIMEM (A6/A7), limite de la mémoire utilisable
8/09	pointeur source pour relocation finale, initialisé à DEBBAS-1
A/0B	pointeur cible pour relocation finale, initialisé à DEBBAS-1
7/C8	copie de HIMEM (A6/A7), adresse haute de la cible lors de la 1ère relocation
9/CA	copie de FINBAS (9C/9D), adresse haute de la source lors de 1ère relocation
E/CF	DEBBAS-1 (vise #00 début programme) adresse basse source 1ère relocation
2	Index pour la table "RENUM" contenant les paramètres à utiliser
4/F5	pointeur dans le bloc bas (cible)
8/F9	initialisé avec #00F3, utilisé pour pointage de lien
9/EA	TXTPTR, pointeur dans le bloc haut (source)
6E4/C6E5	NEWNUM, égale DEFNUM par défaut
6E6/C6E7	NEWPAS, égale DEFPAS par défaut
6E8/C6E9	PRELGN, 0 par défaut
6EA/C6EB	DERLGN, #FFFF par défaut

Informations non documentées

En mode direct, il est possible d'entrer des numéros de ligne jusqu'à 63999 (#F9FF)! L'utilisation de RENUM permet de se retrouver avec des numéros de ligne jusqu'à 65534 (#FFFE). Mais attention, certaines commandes peuvent déclencher un "ILLEGAL DIRECT ERROR" si elles se trouvent dans une ligne dont le numéro est supérieur à 65279 (#FEFF). Les numéros #FF00 et suivants déclenchent ce type d'erreur car le #FF sert d'indicateur de mode direct.

Il n'y a pas d'analyse de validité sur la valeur des paramètres de RENUM. Par exemple un NEWPAS de 0 marche, mais toutes les lignes portent le même numéro (NEWNUM ou DEFNUM)! Il est donc fortement conseillé d'effectuer une sauvegarde du programme avant chaque RENUM, car il n'est pas toujours simple de tout prévoir.

RENUM ne permet pas d'intervertir des lignes; les lignes restent dans le même ordre. Si vous devez déplacer des lignes dans votre programme, il faut découper celui-ci en morceaux, faire un RENUM des morceaux, les sauver et les recoller

dans un ordre différent à l'aide de la commande LOAD,J ou MERGE.
Le token RESTORE (#9A) est pris en compte, mais pas le "restore" (en minuscules) du Sédoric: C'est un comble! La documentation n'est pas claire la-dessus: en effet "!restore" fait bel et bien appel au restore de Sédoric, mais n'est pas mis à jour par RENUM, alors que "RESTORE" fait appel au RESTORE de la ROM, est pris en considération par RENUM, mais n'est pas mis à jour pour cause de manque d'argument! Utilisez donc toujours "!RESTORE".

Toute commande placée après un GOTO n'est jamais exécutée. De manière inattendue et surprenante, lorsqu'il n'y a pas de ":" entre le GOTO et cette commande aucune "SYNTAX ERROR" n'est déclenchée! Autre curiosité du BASIC: contrairement à ce qui se passe avec REM, la chaîne qui suit "" est toujours codée avec les token BASIC. Or dans tous les cas, il est possible de placer un "" non précédé d'un ":" pour introduire un commentaire. Dans ces conditions, les pères du Sédoric ont dû faire preuve d'imagination pour analyser les lignes BASIC afin de renuméroter correctement les GOTO, GOSUB etc.. voir la routine C548 qui vaut son pesant d'or!

Utilisation en LM

Bien que cela ne présente aucun intérêt, il doit être possible de renuméroter un programme BASIC à partir d'un programme en langage machine en faisant un JSR F14E après avoir basculé sur la RAMOV. Avant ce JSR, il est possible d'initialiser directement DEFNUM et DEFPAS en C03E/C03F et C040/C041. On peut aussi écrire les paramètres NEWNUM, NEWPAS, PRELGN et DERLGN dans le tampon clavier, initialiser TXTPTR puis faire le JSR F14E (voir les détails en annexe).

Initialise NEWNUM, NEWPAS, PRELGN et DERLGN
selon DEFNUM, DEFPAS, DEFPRE et DEFDER
(Mise en place des valeurs par défaut)

C40Da	A2 03	LDX #03	index pour copier 4 octets de C03E/C041 vers
C40Fa	BD 3E C0	LDA C03E,X	C6E4/C6E7 (C6E4/5 = C03E/F = DEFNUM, C6E6/7 =
C412a	9D E4 C6	STA C6E4,X	C040/1 = DEFPAS) (**n°** début et **pas** par **déf**aut)
C415a	CA	DEX	vise l'octet précédent
C416a	10 F7	BPL C40F	reboucle tant qu'il en reste à copier
C418a	8E EA C6	STX C6EA	X = #FF pour avoir DEFDER = C6EA/B = #FFFF
C41Ba	8E EB C6	STX C6EB	(**der**nier n° de ligne par **déf**aut)
C41Ea	E8	INX	X = #00 pour avoir DEFPRE = C6E8/9 = #0000
C41Fa	8E E8 C6	STX C6E8	(**pre**mier n° de ligne par **déf**aut) et retourne
C422a	8E E9 C6	STX C6E9	avec X = #00 (index pour écrire dans la table
C425a	60	RTS	"RENUM" en C6E4/C6EB)

Ajustements pour paramètres omis

C426a	20 98 D3	JSR D398	XCRGET lit le caract. suiv. à TXTPTR et le convertit en MAJ.
C429a	E8	INX	indexe le paramètre suivant pour écrire dans
C42Aa	E8	INX	la table "RENUM" (2 octets par paramètre)
C42Ba	E0 08	CPX #08	valeurs valides: 0 < X < 7
C42Da	D0 06	BNE C435	si oui, continue en C435
C42Fa	4C 23 DE	JMP DE23	sinon, "SYNTAX ERROR"

Entrée commande RENUM

Rappel: une ligne BASIC est précédée de 5 octets: un #00 qui marque le début de ligne, 2 octets LLHH de lien (adresse du lien de la ligne suivante) et 2 octets LLHH de n° de ligne. Cet en-tête est suivi des instructions BASIC proprement dites. La fin du programme est marquée par un HH de lien nul. En pratique, par trois #00 (nouvelle ligne et lien nul)

Analyse de syntaxe et mise à jour des paramètres

C432a	20 0D C4	JSR C40D	initialise DEFNUM, DEFPAS, DEFPRE et DEFDER
C435a	20 9E D3	JSR D39E	XCRGOT relit le caractère à TXTPTR et le convertit en MAJUSCULE
C438a	F0 14	BEQ C44E	continue en C44E s'il n'y a plus de paramètres
C43Aa	C9 2C	CMP #2C	le caractère lu est-il une ","? (un paramètre sauté)
C43Ca	F0 E8	BEQ C426	si oui, continue en C426
C43Ea	86 F2	STX F2	sinon, sauve X dans F2 (index table "RENUM")
C440a	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)
C443a	A6 F2	LDX F2	récupère X
C445a	9D E5 C6	STA C6E5,X	
C448a	98	TYA	sauve la valeur à la position X de table "RENUM"
C449a	9D E4 C6	STA C6E4,X	
C44Ca	90 E7	BCC C435	reboucle en C435 si un nombre à été évalué (C = 0)
C44Ea	A5 A6	LDA A6	
C450a	A4 A7	LDY A7	
C452a	85 04	STA 04	copie HIMEM (A6/A7) en 04/05
C454a	84 05	STY 05	
C456a	A5 9A	LDA 9A	
C458a	A4 9B	LDY 9B	
C45Aa	85 00	STA 00	copie DEBBAS (9A/9B) en 00/01

```
45Ca  84 01      STY 01
45Ea  A5 9C      LDA 9C
460a  A4 9D      LDY 9D
462a  85 02      STA 02        copie FINBAS (9C/9D) en 02/03
464a  84 03      STY 03
466a  20 A8 C4   JSR C4A8      entrée réelle de la routine RENUM proprement dite
469a  4C B4 E0   JMP E0B4      restaure liens de lignes et pointeurs puis termine

      Recherche l'adresse d'une ligne BASIC
46Ca  86 33      STX 33        sauve XA en 33/34
46Ea  85 34      STA 34
470a  A5 00      LDA 00        XA = adresse du lien en 00/01
472a  A6 01      LDX 01
474a  4C DC C6   JMP C6DC      recherche adresse CE/CF de la ligne BASIC XA

      Remet octet en place et incrémente pointeurs source et cible
477a  91 F4      STA (F4),Y    sauve octet lu à TXTPTR selon adresse en F4/F5
479a  E6 E9      INC E9
47Ba  D0 02      BNE C47F      incrémente TXTPTR (source = bloc haut)
47Da  E6 EA      INC EA
47Fa  E6 F4      INC F4
481a  D0 0D      BNE C490      incrémente F4/F5 (cible = bloc bas)
483a  E6 F5      INC F5
485a  60         RTS           retourne avec Z = 0 car adresse cible jamais page zéro

      Mise à jour du contenu de 08/09
486a  98         TYA
487a  18         CLC
488a  65 08      ADC 08
48Aa  85 08      STA 08        08/09 = 08/09 + Y
48Ca  90 02      BCC C490
48Ea  E6 09      INC 09
490a  60         RTS

      Mise à jour du contenu de 0A/0B
491a  98         TYA
492a  18         CLC
493a  65 0A      ADC 0A
495a  85 0A      STA 0A        0A/0B = 0A/0B + Y
497a  90 F7      BCC C490
499a  E6 0B      INC 0B
49Ba  60         RTS

      Mise à jour du pointeur cible F4/F5
49Ca  18         CLC
49Da  A9 05      LDA #05
49Fa  65 F4      ADC F4        F4/F5 = F4/F5 + 5
4A1a  85 F4      STA F4        (en-tête de ligne = 5 octets)
4A3a  90 EB      BCC C490
4A5a  E6 F5      INC F5
4A7a  60         RTS
```

Entrée réelle de la routine RENUM proprement dite

Voici l'organigramme utilisé:

- Déplacement de l'ensemble du programme BASIC vers le haut sous HIMEM
- Analyse octet par octet du programme avant de le remettre en place
- Recherche des **T**okens à **R**enuméroter "TR" qui sont suivis d'un **N**uméro de ligne à **R**enuméroter "NR"
- Remplacement de ce n° "NR" qui est écrit en clair (avec les caractères de 0 à 9) par les 5 octets suivants: #FF, puis 2 octets de "NR" convertit en hexadécimal et enfin adresse sur 2 octets du prochain "TR" à mettre à jour, (il s'agit d'une sorte de lien des "TR", à distinguer des liens de lignes normaux, mais fonctionnant sur le même principe)
- Restauration des liens de lignes du programme BASIC redescendu
- Pour chaque "TR", remplace chaque "NR" en hexadécimal par l'adresse de la ligne corresp
- Dernière ligne à renuméroter = fin provisoire du programme BASIC
- Mise à jour des n° de ligne de toutes les lignes de PRELGN à DERLGN
- Pour chaque "TR" remplace l'adresse de la ligne par le n° corresp

```
      Déplace le programme BASIC vers le haut sous HIMEM
4A8a  78         SEI           interdit les interruptions
4A9a  A4 01      LDY 01
4ABa  A6 00      LDX 00
4ADa  D0 01      BNE C4B0
```

```
C4AFa  88        DEY       calcule DEBBAS - 1
C4B0a  CA        DEX       (qui vise le #00 de début de la 1ère ligne)
C4B1a  86 CE     STX CE
C4B3a  84 CF     STY CF    et copie cette valeur en CE/CF
C4B5a  86 08     STX 08    (adresse 1er octet du bloc à déplacer)
C4B7a  84 09     STY 09
C4B9a  86 0A     STX 0A    ainsi qu'en 08/09 et en 0A/0B
C4BBa  84 0B     STY 0B
C4BDa  A5 02     LDA 02
C4BFa  A4 03     LDY 03    copie FINBAS (02/03) en C9/CA
C4C1a  85 C9     STA C9    (adresse dernier octet du bloc à déplacer)
C4C3a  84 CA     STY CA
C4C5a  A5 04     LDA 04
C4C7a  A4 05     LDY 05    copie HIMEM (04/05) en C7/C8
C4C9a  85 C7     STA C7    (adresse dernier octet cible du bloc)
C4CBa  84 C8     STY C8
C4CDa  20 D4 C6  JSR C6D4  MOVE vers le haut (sous HIMEM) du programme BASIC
C4D0a  E6 C8     INC C8
C4D2a  A5 C7     LDA C7    calcule la nouvelle adresse de début BASIC
C4D4a  A4 C8     LDY C8    (bloc haut) et la copie en E9/EA (TXTPTR)
C4D6a  85 E9     STA E9    (pointe sur le #00 de début de la 1ère ligne)
C4D8a  84 EA     STY EA
C4DAa  A5 CE     LDA CE
C4DCa  A4 CF     LDY CF    copie ancien DEBBAS - 1 (CE/CF) en F4/F5
C4DEa  85 F4     STA F4    (pointe sur le #00 de début de la 1ère ligne)
C4E0a  84 F5     STY F5
C4E2a  A9 F3     LDA #F3
C4E4a  A0 00     LDY #00   F8/F9 = #00F3
C4E6a  85 F8     STA F8
C4E8a  84 F9     STY F9
```

Analyse octet par octet avant de remettre en place

Les 5 octets d'en-tête de ligne seront recopiés tels quels sans modification.

```
C4EAa  A0 00     LDY #00   index pour lecture à TXTPTR (bloc BASIC en haut)
C4ECa  B1 E9     LDA (E9),Y lit octet à TXTPTR + Y dans le bloc haut
C4EEa  D0 25     BNE C515  continue en C515 si pas #00 de début de ligne
C4F0a  A0 02     LDY #02   si début de ligne, indexe octet fort du lien
C4F2a  B1 E9     LDA (E9),Y et le lit (à 0 si fin du programme BASIC)
C4F4a  F0 0E     BEQ C504  continue en C504 s'il est nul (fin atteinte)
C4F6a  A0 00     LDY #00   si pas fini, indexe pour lire ligne depuis début
C4F8a  A2 04     LDX #04   indexe pour lire les 5 octets d'en-tête de ligne
C4FAa  B1 E9     LDA (E9),Y lit un octet à TXTPTR dans le bloc haut
C4FCa  20 77 C4  JSR C477  remet cet octet en place dans le bloc bas et incrémente les pointeurs source (bloc haut) et
       cible (bloc bas)
C4FFa  CA        DEX       décrémente le compteur d'octets à déplacer
C500a  10 F8     BPL C4FA  et reboucle en C4FA tant qu'il en reste
C502a  30 E6     BMI C4EA  fini, reprend en C4EA (analyse l'octet suivant, c'est à dire le début de la ligne proprement
       dite)
```

Fin du programme BASIC atteinte

```
C504a  91 F4     STA (F4),Y remet en place l'octet lu (#00) (Y = 2 en entrée)
C506a  88        DEY       vise l'octet précédent et reboucle en C504 pour
C507a  10 FB     BPL C504  mettre à 0 les 3 derniers octets du programme
C509a  A0 04     LDY #04
C50Ba  91 F8     STA (F8),Y force à zéro l'octet visé par F8/F9 + 4 c'est à dire le lien du dernier TR qu'il faudra remettre
       à jour
C50Da  4C 99 C5  JMP C599  mise à jour proprement dite
```

C'était un code ordinaire

```
C510a  20 77 C4  JSR C477  remet cet octet en place dans le bloc bas et incrémente les pointeurs source (bloc haut) et
       cible (bloc bas)
C513a  D0 D5     BNE C4EA  rebouclage forcé car revient de C477 avec Z = 0
```

Est-ce l'un des tokens succeptibles d'être mis à jour?

```
C515a  C9 97     CMP #97   est-ce le token "GOTO"?
C517a  F0 14     BEQ C52D  si oui, continue en C52D
C519a  C9 9B     CMP #9B   est-ce le token "GOSUB"?
C51Ba  F0 10     BEQ C52D  si oui, continue en C52D
C51Da  C9 C8     CMP #C8   est-ce le token "ELSE"?
C51Fa  F0 0C     BEQ C52D  si oui, continue en C52D
```

```
521a  C9 C9     CMP #C9     est-ce le token "THEN"?
523a  F0 08     BEQ C52D    si oui, continue en C52D
525a  C9 9A     CMP #9A     est-ce le token "RESTORE"?
527a  F0 04     BEQ C52D    si oui, continue en C52D
529a  C9 98     CMP #98     est-ce le token "RUN"?
52Ba  D0 E3     BNE C510    sinon, reprend en C510 (c'était un code ordinaire)
                            A ce stade, lorsqu'un "TR" est trouvé, TXTPTR pointe sur lui avec Y = #00.
```

Est-ce un "TR" (Token à Renuméroter)?

Le 1er code suivant ce token est-il un chiffre de 0 à 9?

Rappel de la syntaxe de GOTO, ON GOTO, GOSUB, ON GOSUB, ELSE, THEN, RESTORE et RUN: selon les cas, ils peuvent être suivis d'un espace, d'un n° de ligne, d'une variable numérique (donc commençant par un caractère alphabétique), d'une expression numérique (pouvant commencer par une parenthèse), d'un autre token (par exemple ELSE GOSUB) ou de rien du tout, c'est à dire du code de fin d'instruction ":" ou de celui de fin de ligne #00. Les variables et expressions numériques ne sont pas mises à jour par RENUM.

```
52Da  20 77 C4  JSR C477    remet cet octet en place dans le bloc bas et incrémente les pointeurs source (bloc haut) et
                            cible (bloc bas) sans toucher Y
530a  B1 E9     LDA (E9),Y  lit octet à TXTPTR + Y (octet suivant le "TR")
532a  F0 B6     BEQ C4EA    reprend en C4EA si nul (fin ligne atteinte)
534a  C9 20     CMP #20     est-ce un espace? (à negliger: sans signification)
536a  F0 F5     BEQ C52D    si oui, OK, reboucle en C52D
538a  C9 30     CMP #30     est-ce un code < #30? (#30 = début des chiffres)
53Aa  90 D4     BCC C510    si oui, pas bon, reprend en C510 (ex: parenthèse)
53Ca  C9 97     CMP #97     est-ce le token "GOTO"?
53Ea  F0 ED     BEQ C52D    si oui, OK, reboucle en C52D
540a  C9 9B     CMP #9B     est-ce le token "GOSUB"?
542a  F0 E9     BEQ C52D    si oui, OK, reboucle en C52D
544a  C9 3A     CMP #3A     est-ce un code >= #3A? (#39 = fin des chiffres)
546a  B0 C8     BCS C510    si oui, pas bon, reprend en C510 (exemple ":")
```

Est-ce un n° valable, qu'il faudra mettre à jour?

Les n° de lignes sont codés en ASCII (ex: 35 = #30 suivi de #35) et prennent donc de 1 à 5 caractères. Il faut trouver l'octet qui suit le dernier chiffre. Cet octet peut être #00 (fin ligne), #20 (espace), #27 ("" pour REM, ça c'est bizarre! voir "non documenté"), #2C (","), un autre code <#30 ("+", "-", "*", "/" etc), #3A (":"), #C8 (token ELSE) ou un autre code >#3A (pas de mise à jour).

```
548a  C8        INY         un chiffre a été trouvé, incrémente index Y. TXTPTR pointe sur le 1er chiffre du n° de ligne
                            à metttre à jour
549a  B1 E9     LDA (E9),Y  lit octet suivant à TXTPTR + Y
54Ba  F0 1A     BEQ C567    si nul, OK continue en C567 (fin de ligne atteinte)
54Da  C9 27     CMP #27     est-ce un """? (REM, voir "non documenté" au début)
54Fa  F0 16     BEQ C567    si oui, OK continue en C567 (c'est valide!)
551a  C9 30     CMP #30     est-ce un code < #30? (#30 = début des chiffres)
553a  90 06     BCC C55B    si oui, continue en C55B (BCC C55F aurait été mieux!)
555a  C9 3A     CMP #3A     est-ce un code < #3A? (#39 = fin des chiffres)
557a  90 EF     BCC C548    si oui, reboucle en C548 (c'est encore un chiffre)
559a  F0 0C     BEQ C567    si c'est un ":", OK continue en C567
55Ba  C9 C8     CMP #C8     est-ce le token "ELSE"?
55Da  F0 08     BEQ C567    si oui, OK continue en C567
55Fa  C9 2C     CMP #2C     est-ce une ","?
561a  F0 04     BEQ C567    si oui, OK continue en C567
563a  C9 20     CMP #20     est-ce un espace? si oui, OK continue en C567
565a  D0 83     BNE C4EA    sinon, reboucle en C4EA (autres cas invalides)
```

Le code suivant le "TR" a été trouvé

```
567a  48        PHA         si oui, empile l'octet qui suit le dernier chiffre
568a  A9 00     LDA #00     et le remplace par un #00 dans le bloc du haut
56Aa  91 E9     STA (E9),Y  à ce moment, Y = nombre de chiffres du n° de ligne
56Ca  A6 F4     LDX F4
56Ea  8A        TXA         le pointeur cible F4/F5 est copié selon l'adresse
56Fa  A0 03     LDY #03     en F8/F9 + 3 (et suivante car 2 octets)
571a  91 F8     STA (F8),Y
573a  C8        INY
574a  A5 F5     LDA F5      puis en F8/F9 qui garde donc en memmoire le point
576a  91 F8     STA (F8),Y  où l'on en est resté dans le bloc du bas
578a  86 F8     STX F8
57Aa  85 F9     STA F9
```

Nous sommes en présence d'une astuce extra: chaque "NR" (n° de ligne appelé par un token et à mettre à jour) est remplacé par un groupe de 5 octets: #FF, le n° codé en hexadécimal sur deux octets et 2 octets de lien indiquant l'adresse du prochain

"NR" qu'il faudra mettre à jour. L'adresse du 1er "NR" sera gardé en (F3) + 3 c'est à dire en F6/F7 (voir initialisation en C4E2). L'adresse du 2ème "NR" sera gardée dans les 2 derniers octets du 1er etc... Les 2 octets de lien de token du dernier "NR" doivent indiquer qu'il s'agit du dernier et qu'il n'y a pas d'adresse suivante. Pour cela, le HH (le 2ème des deux) sera mis à zéro en C50B (Il ne peut pas y avoir de programme BASIC en page zéro).

C57Ca	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)
C57Fa	A0 02	LDY #02	
C581a	91 F4	STA (F4),Y	
C583a	88	DEY	place le résultat dans le bloc du bas selon
C584a	A5 33	LDA 33	l'adresse en F4/F5 + 1 (et suivante car 2 octets)
C586a	91 F4	STA (F4),Y	et met #FF selon adresse en F4/F5
C588a	88	DEY	
C589a	A9 FF	LDA #FF	
C58Ba	91 F4	STA (F4),Y	
C58Da	20 9C C4	JSR C49C	mise à jour du pointeur cible F4/F5 = F4/F5 + #05. On laisse donc de la place pour 5 octets. A la suite des 2 DEY, on a Y = #00.
C590a	68	PLA	récupère l'octet précédemment empilé
C591a	91 E9	STA (E9),Y	et l'écrit dans bloc haut à TXTPTR car Y = #00. C'est à dire à la suite du 1er octet qui suit le n° qui a été évalué. En fait l'octet empilé reprend sa place initiale. Ce tour de passe-passe permet d'éviter que certains codes ne viennent troubler l'évaluation du nombre.
C593a	C9 2C	CMP #2C	est-ce une ","? (ON .. GOTO .. , autre n° de ligne)
C595a	F0 96	BEQ C52D	si oui, reboucle en C52D ("," équivaut à un "TR")
C597a	D0 CC	BNE C565	sinon, reprend en C565, c'est à dire en C4EA

Mise à jour proprement dite

Restauration des liens de lignes du bloc BASIC redescendu

C599a	A6 00	LDX 00	XA = DEBBAS = 1er lien de ligne du programme
C59Ba	A5 01	LDA 01	
C59Da	18	CLC	pour addition ultérieure
C59Ea	A0 01	LDY #01	index pour lecture
C5A0a	86 F4	STX F4	mise à jour du pointeur F4/F5
C5A2a	85 F5	STA F5	pour viser le 1er lien du bloc bas
C5A4a	B1 F4	LDA (F4),Y	lit deuxième octet HH du lien
C5A6a	F0 22	BEQ C5CA	continue en C5CA si fin de programme BASIC
C5A8a	A0 03	LDY #03	sinon, prépare Y pour début de boucle
C5AAa	C8	INY	index pour lecture de ligne proprement dite
C5ABa	B1 F4	LDA (F4),Y	lit octet de ligne BASIC proprement dite
C5ADa	F0 0A	BEQ C5B9	continue en C5B9 si atteint ligne suivante
C5AFa	C9 FF	CMP #FF	est-ce un #FF? (flag de "TR")
C5B1a	D0 F7	BNE C5AA	sinon, reboucle en C5AA
C5B3a	98	TYA	si oui, Y = Y + 4 (pour accélérer la lecture)
C5B4a	69 04	ADC #04	
C5B6a	A8	TAY	
C5B7a	90 F2	BCC C5AB	rebouclage forcé en C5AB (cherche ligne suivante). Une ligne ne pouvant avoir plus de 256 caractères, il n'y a jamais de retenue

Ligne suivante trouvée: mise à jour du lien de ligne précédent

C5B9a	98	TYA	(F4/F5 pointe sur le #00 de début de ligne)
C5BAa	38	SEC	
C5BBa	65 F4	ADC F4	calcule XA = F4/F5 + Y + 1 = adresse du lien actuel
C5BDa	AA	TAX	
C5BEa	A0 00	LDY #00	et met le résultat en place dans le lien précédent
C5C0a	91 F4	STA (F4),Y	
C5C2a	98	TYA	on reprend donc avec XA pointant sur le lien actuel
C5C3a	65 F5	ADC F5	
C5C5a	C8	INY	et Y = #01 pour explorer la ligne suivante
C5C6a	91 F4	STA (F4),Y	
C5C8a	90 D6	BCC C5A0	rebouclage forcé en C5A0 (cherche ligne suivante)

Fin du programme BASIC atteinte

C5CAa	A6 F6	LDX F6	XA = F6/F7 = lien indiquant adresse du 1er "TR"
C5CCa	A5 F7	LDA F7	(Z = 1 si F7 est nul, c'est à dire si pas de "TR")

Remplace chaque "NR" par l'adresse de la ligne corresp

C5CEa	F0 23	BEQ C5F3	si Z = 1, continue en C5F3 (il n'y a plus de "NR")
C5D0a	86 F4	STX F4	si Z = 0, mise à jour de F4/F5
C5D2a	85 F5	STA F5	qui pointe sur le prochain "NR"
C5D4a	A0 01	LDY #01	
C5D6a	B1 F4	LDA (F4),Y	lecture dans XA de ce "NR"
C5D8a	AA	TAX	selon adresse en F4/F5 + 1 et 2

```
5D9a  C8        INY
5DAa  B1 F4     LDA (F4),Y
5DCa  20 6C C4  JSR C46C      recherche l'adresse CE/CF de la ligne BASIC n° XA
5DFa  A0 01     LDY #01
5E1a  A5 CE     LDA CE
5E3a  91 F4     STA (F4),Y    place adresse selon F4/F5 + 1 (et suivant car 2 octets)
5E5a  C8        INY
5E6a  A5 CF     LDA CF
5E8a  91 F4     STA (F4),Y
5EAa  C8        INY
5EBa  B1 F4     LDA (F4),Y     lit les deux octets suivants
5EDa  AA        TAX            (lien = adresse du "TR" suivant)
5EEa  C8        INY
5EFa  B1 F4     LDA (F4),Y
5F1a  D0 DB     BNE C5CE      reboucle en C5CE s'il y en a encore à mettre à jour

      Dernière ligne à renuméroter = fin provisoire du programme BASIC
5F3a  AE EA C6  LDX C6EA      XA = DERLGN (n° de la dernière ligne à renuméroter)
5F6a  AD EB C6  LDA C6EB
5F9a  20 6C C4  JSR C46C      cherche adresse en CE/CF du lien de ligne BASIC n° XA
5FCa  A0 01     LDY #01
5FEa  B1 CE     LDA (CE),Y    empile le HH du lien de cette ligne et le remplace
600a  48        PHA           par #00, marquant ainsi une fausse fin de programme
601a  A9 00     LDA #00
603a  91 CE     STA (CE),Y
605a  AE E8 C6  LDX C6E8      XA = PRELGN (n° de la première ligne à renuméroter)
608a  AD E9 C6  LDA C6E9
60Ba  20 6C C4  JSR C46C      cherche adresse en CE/CF du lien de ligne BASIC n° XA

      Mise à jour des n° de ligne de toutes les lignes de PRELGN à DERLGN
60Ea  A0 03     LDY #03
610a  AD E5 C6  LDA C6E5
613a  91 CE     STA (CE),Y    copie NEWNUM à la place de l'ancien n°
615a  88        DEY
616a  AD E4 C6  LDA C6E4
619a  91 CE     STA (CE),Y
61Ba  18        CLC
61Ca  6D E6 C6  ADC C6E6
61Fa  8D E4 C6  STA C6E4
622a  AD E5 C6  LDA C6E5      calcule NEWNUM = NEWNUM + NEWPAS
625a  6D E7 C6  ADC C6E7
628a  8D E5 C6  STA C6E5
62Ba  B0 10     BCS C63D      continue en C63D si dépassement de capacité
62Da  A0 00     LDY #00
62Fa  B1 CE     LDA (CE),Y
631a  AA        TAX           XA = adresse de la ligne BASIC suivante
632a  C8        INY
633a  B1 CE     LDA (CE),Y
635a  F0 18     BEQ C64F      continue en C64F si HH nul
637a  86 CE     STX CE        (dernière ligne à renuméroter atteinte)
639a  85 CF     STA CF        mise à jour de CE/CF avec adresse ligne suivante
63Ba  D0 D1     BNE C60E      si HH du lien non nul, rebouclage forcé en C60E

      Dépassement: renumérotation impossible
63Da  68        PLA           élimine 1 octet de la pile (celui du HH inutile)
63Ea  20 0D C4  JSR C40D      re-initialise NEWNUM, NEWPAS, PRELGN et DERLGN
641a  A0 03     LDY #03       index pour ecrire 4 octets
643a  8A        TXA           force A à zéro (sorti du s/p C40D avec X = #00)
644a  99 E4 C6  STA C6E4,Y    force NEWNUM et NEWPAS à zéro
647a  88        DEY           vise le précédant
648a  10 FA     BPL C644      reboucle tant qu'il en reste à mettre à zéro
64Aa  EE E6 C6  INC C6E6      NEWPAS passe à 1
64Da  D0 A4     BNE C5F3      rebouclage forcé vers C5F3 (renum de sauvetage)

      Dernière ligne à renuméroter atteinte
64Fa  68        PLA           récupère un octet sur la pile et l'écrit selon
650a  91 CE     STA (CE),Y    adresse en CE/CF + Y (c'est à dire le remet à sa place)
652a  A5 F7     LDA F7        teste F7
654a  F0 42     BEQ C698      si nul (il n'y a pas de "TR"), continue en C698,
656a  A0 01     LDY #01       sinon ...
```

```
C658a  B1 F6      LDA (F6),Y
C65Aa  85 F8      STA F8        lit 2 octets selon F6/F7 + 1 et 2
C65Ca  C8         INY           (adresse de ligne dont il faut trouver le nouveau n°)
C65Da  B1 F6      LDA (F6),Y    et les copie en F8/F9
C65Fa  85 F9      STA F9
C661a  C8         INY
C662a  B1 F6      LDA (F6),Y    lit 2 octets selon F6/F7 + 3 et 4
C664a  48         PHA           (lien pour adresse suivante)
C665a  C8         INY           et les empile
C666a  B1 F6      LDA (F6),Y
C668a  48         PHA
C669a  A0 03      LDY #03
C66Ba  B1 F8      LDA (F8),Y
C66Da  48         PHA           YA = 2 octets lus à F8/F9 + 2 et 3
C66Ea  88         DEY           (nouveau n° de la ligne)
C66Fa  B1 F8      LDA (F8),Y
C671a  A8         TAY
C672a  68         PLA
C673a  20 CA D2   JSR D2CA      JSR DF40/ROM transfère un nombre non signé YA dans ACC1 (floating point
       accumulator)
C676a  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
C679a  A0 00      LDY #00
C67Ba  B9 01 01   LDA 0101,Y    recherche le 0 qui marque la fin de la chaîne
C67Ea  F0 05      BEQ C685      si trouvé, continue en C685
C680a  91 F6      STA (F6),Y    non trouvé, écrit octet lu selon F6/F7 + Y. C'est à dire à l'emplacement du nouveau n° de
       ligne situé après le token.
C682a  C8         INY           index octet suivant
C683a  D0 F6      BNE C67B      reboucle en C67B tant que pas trouvé ce 0
C685a  A9 FF      LDA #FF       A = #FF (bouche trou pour justifier le n° à droite)
C687a  C0 05      CPY #05       a t-on Y = 5? (le n° comporte t-il 5 caractères?)
C689a  F0 05      BEQ C690      si oui, continue en C690 (fini)
C68Ba  91 F6      STA (F6),Y    sinon, place ce #FF selon F6/F7 + Y
C68Da  C8         INY           index octet suivant
C68Ea  D0 F7      BNE C687      reboucle en C687 jusqu'à ce que le n° ait 5 catact
C690a  68         PLA
C691a  85 F7      STA F7        dépile adresse de lien et l'écrit en F6/F7
C693a  68         PLA
C694a  85 F6      STA F6
C696a  B0 BA      BCS C652      reprise forcée en C652 car C à 1 en C689
C698a  A0 00      LDY #00       index pour rechercher #FF ou début nouvelle ligne
C69Aa  B1 08      LDA (08),Y    lit octet selon adresse en 08/09 (DEBBAS - 1)
C69Ca  F0 13      BEQ C6B1      continue en C6B1 si nul
C69Ea  C9 FF      CMP #FF
C6A0a  F0 05      BEQ C6A7      continue en C6A7 si #FF
C6A2a  91 0A      STA (0A),Y    écrit octet lu selon adresse en 0A/0B (DEBBAS - 1)
C6A4a  C8         INY           octet suivant
C6A5a  D0 F3      BNE C69A      branchement forcé en C69A
C6A7a  20 91 C4   JSR C491      mise à jour 0A/0B = 0A/0B + Y pointeur reloc cible
C6AAa  C8         INY           ce qui revient à sauter le #FF qui vient d'être trouvé et donc à redescendre la suite du
       programme d'une place
C6ABa  20 86 C4   JSR C486      mise à jour 08/09 = 08/09 + Y pointeur reloc source
C6AEa  4C 98 C6   JMP C698      reprise forcée en C698

       Octet lu est nul (nouvelle ligne)
C6B1a  C8         INY           vise 2 octets plus loin
C6B2a  C8         INY           c'est à dire sur le HH du lien
C6B3a  B1 08      LDA (08),Y    empile l'octet lu selon adresse en 08/09 + 2
C6B5a  08         PHP           sauvegarde les indicateurs 6502 dont Z
C6B6a  88         DEY
C6B7a  88         DEY
C6B8a  A2 04      LDX #04
C6BAa  B1 08      LDA (08),Y    lit 5 octets situés à partir de l'adresse en 08/09
C6BCa  91 0A      STA (0A),Y    et les copie à partir de l'adresse en 0A/0B
C6BEa  C8         INY           c'est à dire recopie les 5 octets d'en-tête
C6BFa  CA         DEX           directement dans le bloc bas
C6C0a  10 F8      BPL C6BA      reboucle tant qu'il en reste à recopier
C6C2a  28         PLP           récupère les indicateurs 6502 dont Z
C6C3a  F0 09      BEQ C6CE      continue en C6CE si Z = 1 (fin du programme)
C6C5a  20 86 C4   JSR C486      mise à jour 08/09 = 08/09 + Y
```

```
6C8a  20 91 C4    JSR C491      mise à jour 0A/0B = 0A/0B + Y
6CBa  4C 98 C6    JMP C698      reprise forcée en C698
```

Voilà, c'est fini

```
6CEa  88          DEY           dépile 2 octets de la pile
6CFa  88          DEY           (lien suivant qui n'existe pas)
6D0a  58          CLI           autorise les interruptions
6D1a  4C 91 C4    JMP C491      mise à jour 0A/0B = 0A/0B + Y et retourne au sous-programme appelant, c'est à dire en
C469 où une restauration finale des liens de lignes est effectuée
```

MOVE vers le haut (sous HIMEM) du programme BASIC

```
E/CF                            1er octet du bloc à déplacer,
9/CA                            dernier octet du bloc à déplacer,
7/C8 et AY                      adresse cible du bloc (attention: haut du nouveau bloc).
                                Retourne avec C7/C8 pointant sur le nouveau début du bloc - #100.

6D4a  20 D8 D5    JSR D5D8      XROM Exécute à partir de la RAM une routine ROM
6D7a  FB C3 F7 C3               adresse ROM 1.0 adresse ROM 1.1
6DBa  60          RTS
```

Recherche une ligne BASIC de n° 33/34 à partir de l'adresse XA
Si trouve, retourne avec C = 1 et adresse en CE/CF (visant le 1er octet de lien)

```
6DCa  20 D8 D5    JSR D5D8      XROM exécute à partir de la RAM une routine ROM
6DFa  E8 C6 BD C6 adresse ROM 1.0 adresse ROM 1.1
6E3a  60          RTS
```

Table "RENUM"

```
6E4a  00 00       NEWNUM        n° de ligne pour commencer la renumérotation
6E6a  00 00       NEWPAS        "pas" pour incrémenter les n° de lignes
6E8a  00 00       PRELGN        n° de la première ligne à renuméroter
6EAa  00 00       DERLGN        n° de la dernière ligne à renuméroter
```

EXECUTION COMMANDE SEDORIC DELETE

Rappel de la syntaxe

DELETE (DELPRE)-(DELDER)

DELPRE est le numéro de la première ligne à supprimer (1ère ligne du programme si ce paramètre est omis). DELDER est le numèro de la dernière ligne à supprimer (dernière ligne du programme si ce paramètre est omis). Utilisable en mode programme, mais DELETE doit se trouver avant le bloc supprimé. Les variables sont conservées. Les fonctions définies par DEF FN se trouvant dans le bloc supprimé sont perdues.

Liste des variables utilisées

```
2/F3              DELPRE        première ligne à supprimer
3/34              DELDER        dernière ligne à supprimer
4/F5                            longueur de la zone à supprimer = nombre d'octets à supprimer
E/CF                            pointeur source dans le bloc haut
2/F3                            pointeur cible dans le bloc bas
```

Informations non documentées

Contrairement à ce qu'affirme le manuel, lorsque le "-" est omis, DELETE n'equivaut pas à un NEW, mais déclenche une SYNTAX ERROR. Par contre DELETE- équivaut à un NEW.

Si le ou les n° indiqués sont supérieurs à 63999 (ce qui peut être le cas après un RENUM, voir à cette commande) on obtient une ILLEGAL QUANTITY ERROR. Il devient donc impossible d'effacer certaines lignes.

Utilisation en LM

Bien que cela ne présente aucun intérêt, il doit être possible de supprimer un bloc de lignes d'un programme BASIC à partir d'un programme en langage machine en faisant un JSR F142 après avoir basculé sur la RAMOV. Avant cela, il faut initialiser DELPRE et DELDER en plaçant ces valeurs dans le tampon clavier puis en initialisant TXTPTR (voir les détails en annexe).

Analyse de syntaxe

```
6ECa  20 F3 D1    JSR D1F3      CAE2/ROM évalue en 33/34 le n° ligne à TXTPTR, retour avec #00 si le paramètre à
TXTPTR est omis. Génère une "ILLEGAL QUANTITY ERROR" si ce n° est supérieur à 63999 et une "SYNTAX
ERROR" si le caractère trouvé est non numérique
6EFa  20 9C D1    JSR D19C      C6B3/ROM cherche l'adresse de la ligne BASIC à partir du début du programme BASIC. Si
la ligne n'est pas trouvée, retour avec l'adresse de la ligne suivante ou celle de la fin du programme
6F2a  A5 CE       LDA CE        place l'adresse de cette ligne dans F2/F3 (DELPRE)
6F4a  85 F2       STA F2        (astuce douteuse: pour économiser un LDA CF,
6F6a  86 F3       STX F3        récupère HH dans X (voir routine dans "l'Oric à nu")
```

```
C6F8a  A9 FF      LDA #FF
C6FAa  85 33      STA 33        force 33/34 à #FF (#FFFF pour DELDER par défaut)
C6FCa  85 34      STA 34
C6FEa  A9 CD      LDA #CD       token "-"
C700a  20 2E D2   JSR D22E      JSR D067/ROM et D3A1/RAMOV demande "-" à TXTPTR, lit le caractère suivant à
                                TXTPTR et le convertit en MAJUSCULE
C703a  F0 09      BEQ C70E      continue en C70E si fin des paramètres
C705a  20 F3 D1   JSR D1F3      CAE2/ROM évalue en 33/34 le n° ligne à TXTPTR, retour avec #00 si le paramètre à
                                TXTPTR est omis. Génère une "ILLEGAL QUANTITY ERROR" si ce n° est supérieur à 63999 et une "SYNTAX
                                ERROR" si le caractère trouvé est non numérique
C708a  E6 33      INC 33
C70Aa  D0 02      BNE C70E      incrémente ce n° de ligne (DELDER)
C70Ca  E6 34      INC 34
C70Ea  20 9C D1   JSR D19C      C6B3/ROM cherche l'adresse de la ligne BASIC à partir du début du programme BASIC.
                                C'est à dire, l'adresse de la ligne qui suit la dernière ligne à supprimer
C711a  A5 CE      LDA CE
C713a  38         SEC
C714a  E5 F2      SBC F2
C716a  85 F4      STA F4        teste si adresse fin < adresse début
C718a  8A         TXA           (F4/F5 = CE/CF - F2/F3)
C719a  E5 F3      SBC F3
C71Ba  85 F5      STA F5
C71Da  90 C4      BCC C6E3      si oui, simple RTS

       Début de la routine de délétion
C71Fa  08         PHP           sauvegarde les indicateurs 6502
C720a  78         SEI           interdit les interruptions
C721a  A0 00      LDY #00
C723a  A5 CE      LDA CE
C725a  C5 A0      CMP A0        teste si pointeur CE/CF < fin tableaux A0/A1
C727a  A5 CF      LDA CF        (DELETE redescend les zones des
C729a  E5 A1      SBC A1        variables et des tableaux)
C72Ba  90 17      BCC C744      si oui, continue en C744

       Reajuste les pointeurs BASIC et restaure les liens
C72Da  A2 04      LDX #04       sinon, on a fini le transfert car le pointeur
C72Fa  B5 9C      LDA 9C,X      source a atteint la fin des tableaux
C731a  E5 F4      SBC F4
C733a  95 9C      STA 9C,X      calcule la nouvelle valeur de A0/A1 (fin des
C735a  B5 9D      LDA 9D,X      tableaux) et la met en place
C737a  E5 F5      SBC F5
C739a  95 9D      STA 9D,X
C73Ba  CA         DEX           calcule la nouvelle valeur de 9E/9F (fin des variables)
C73Ca  CA         DEX           puis la nouvelle valeur de 9C/9D (fin du programme BASIC)
C73Da  10 F0      BPL C72F      reboucle en C72F pour X = 2 puis X = 0
C73Fa  20 88 D1   JSR D188      C563/ROM restaure les liens des lignes
C742a  28         PLP           récupère les indicateurs 6502
C743a  60         RTS           sortie de la commande DELETE

       Tranfère un octet du bloc haut vers le bloc bas
C744a  B1 CE      LDA (CE),Y    lit octet selon CE/CF (début bloc à déplacer) et
C746a  91 F2      STA (F2),Y    l'écrit selon F2/F3 (zone après fin bloc à dépl)
C748a  E6 CE      INC CE
C74Aa  D0 02      BNE C74E      incrémente les pointeurs et reboucle pour
C74Ca  E6 CF      INC CF        tester si le pointeur source atteint le
C74Ea  E6 F2      INC F2        pointeur de fin des tableaux
C750a  D0 D1      BNE C723
C752a  E6 F3      INC F3
C754a  D0 CD      BNE C723      rebouclage forcé en C723
```

EXECUTION COMMANDE SEDORIC MOVE

Rappel de la syntaxe

MOVE DEBBLOC,FINBLOC,DEBCIBL

Assure le transfert d'un bloc mémoire compris entre DEBBLOC (inclus) et FINBLOC (exclu) vers DEBCIBL, nouvelle adresse de début du bloc. Aucun de ces trois paramètres ne peut être omis. L'utilisation de cette commande peut rendre les plus grands services, mais aussi à l'origine des plus grandes catastropphes. Là encore, prévoir une sauvegarde si besoin avant de se lancer!

Liste des variables utilisées

```
F2/F3          DEBBLOC    adresse du premier octet du bloc mémoire à déplacer
```

4/F5		FINBLOC	adresse de l'octet qui suit le dernier octet du bloc à déplacer
6/F7		DEBCIBL	nouvelle adresse de début du bloc mémoire
8			complément à 2 de Y, le nombre d'octets de la page incomplète
9			nombre de pages entières à transférer

Non documenté dans le manuel

Même si le manuel indique qu'un MOVE malencontreux peut écraser la RAMOV, il cache l'essentiel, à savoir que l'on peut utiliser MOVE pour travailler sur la RAMOV. Il est possible par exemple de la descendre en RAM, de la corriger et de la remettre en place!

Remarques sur la routine MOVE

La sécurité et la vitesse de transfert ont été privilégiés au détriment de la concision: cette routine comporte de belles longueurs!

Utilisation en LM

Voilà une routine bien précieuse dans les programmes en langage machine. Cependant, il ne semble pas très simple d'utiliser directement la commande MOVE du Sédoric, parce qu'il faut d'abord charger la banque n°1, initialiser F2/F3, F4/F5 et F6/F7 et enfin faire un JSR C771. L'autre solution, plus indirecte consiste à placer les paramètres DEBBLOC, FINBLOC et DEBCIBL dans le tampon clavier, à initialiser TXTPTR puis faire le JSR F136 (voir les détails en annexe).

Analyse de syntaxe

756a	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)
759a	84 F2	STY F2	place le résultat YA en F2/F3 (DEBBLOC)
75Ba	85 F3	STA F3	
75Da	20 2C D2	JSR D22C	D067/ROM exige une "," et place TXTPTR sur l'octet suivant
760a	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)
763a	84 F4	STY F4	place le résultat YA en F4/F5 (FINBLOC)
765a	85 F5	STA F5	
767a	20 2C D2	JSR D22C	D067/ROM exige une "," place TXTPTR sur l'octet suivant
76Aa	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)
76Da	84 F6	STY F6	place le résultat YA en F6/F7 (DEBCIBLE)
76Fa	85 F7	STA F7	

Calcule le nombre de pages entières à transférer

771a	08	PHP	sauvegarde les indicateurs 6502
772a	78	SEI	interdit les interruptions
773a	38	SEC	
774a	A5 F4	LDA F4	calcule:
776a	E5 F2	SBC F2	YX = F4/F5 - F2/F3 = LNGBLOC longueur du bloc
778a	A8	TAY	ou X = HH est le nombre de pages entières
779a	A5 F5	LDA F5	et Y le nombre d'octets de la page incomplète
77Ba	E5 F3	SBC F3	
77Da	AA	TAX	
77Ea	90 48	BCC C7C8	erreur si FINBLOC < DEBBLOC (LNGBLOC négatif)
780a	86 F9	STX F9	F9 = nombre de pages entières à transférer

Evalue la direction du MOVE à utiliser

782a	A5 F6	LDA F6	calcule F6/F7 - F2/F3 pour
784a	C5 F2	CMP F2	évaluer si DEBCIBLE < DEBBLOC
786a	A5 F7	LDA F7	(cas d'un MOVE descendant sans problème)
788a	E5 F3	SBC F3	
78Aa	B0 3F	BCS C7CB	si ce n'est pas le cas (c'est à dire si F6/F7 >= F2/F3 soit DEBCIBL >= DEBBLOC), continue en C7CB (cas d'un MOVE ascendant)

MOVE descendant (par le début)

F6/F7 < F2/F3 (cas où C = 0) Il faut commencer le transfert par le début.

```
        DEBCIBL          <-      DEBBLOC             FINCIBL             FINCBLOC
        |                        |                   |                   |
        V                        V                   V                   V
=============================================================================
=
        F6/F7                    F2/F3=F4/F5-YX                          F4/F5
        DEBCIBL+Y-Y              FINBLOC-X+(-Y)
```

(DEBCIBL+Y),(-Y) (FINBLOC-X),(-Y)

Explication: Si ce MOVE descendant est sans problème, les auteurs de Sédoric ont choisi une méthode bien compliquée. Il s'agit de lire les octets un à un à partir de DEBBLOC et de les copier à partir de DEBCIBL. Sachant que la longueur du bloc à transférer est de YX octets, c'est à dire X fois 256 octets plus Y octets, il faut copier X "pages" complètes et une page incomplète de Y octets.

On veut utiliser une boucle du type LDA (DEBBLOC),Y STA(DEBCIBL),Y. Pour parvenir à leurs fins, les auteurs ont utilisé non pas Y, mais le complément à 2 de Y. Ce qui donne quelques barbarismes mathématiques:
DEBBLOB = FINBLOC - LNGBLOC soit F2/F3 = F4/F5 - YX qui s'écrit aussi F4/F5 - X - Y ou F4/F5 - X + complèment à 2 de Y. Pour lire les octets, on aura donc un LDA(F4/F5-X),(complément à 2 de Y)!

De même, DEBCIBL = DEBCIBL + Y - Y ou DEBCIBL + Y + complément à 2 de Y. Soit F6/F7 + Y + complément à 2 de Y. Et ce n'est pas tout, pour ajouter Y, on va retrancher le complément à 2!!! Pour écrire les octets, on aura donc STA(F6/F7-complément à 2 de Y), (complément à 2 de Y).

Calcul du complément à 2 de Y

C78Ca	98	TYA	
C78Da	49 FF	EOR #FF	on inverse les bits
C78Fa	69 01	ADC #01	et on ajoute 1
C791a	A8	TAY	
C792a	85 F8	STA F8	résultat dans Y et dans F8
C794a	90 03	BCC C799	saute les 2 instructions suivantes si C = 0
C796a	CA	DEX	décréme X = nombre de pages entières à transférer
C797a	E6 F5	INC F5	incrémente HH de FINBLOC pour compenser

Calcule adresse cible

C799a	38	SEC	
C79Aa	A5 F6	LDA F6	calcule F6/F7 = F6/F7 - F8
C79Ca	E5 F8	SBC F8	c'est à dire F6/F7 + Y
C79Ea	85 F6	STA F6	
C7A0a	B0 02	BCS C7A4	
C7A2a	C6 F7	DEC F7	

Calcule adresse source

C7A4a	18	CLC	
C7A5a	A5 F5	LDA F5	calcule F5 = F5 - F9
C7A7a	E5 F9	SBC F9	c'est à dire F5 - X
C7A9a	85 F5	STA F5	

Transfère la page incomplète

C7ABa	E8	INX	
C7ACa	B1 F4	LDA (F4),Y	lit octet selon source + Y
C7AEa	91 F6	STA (F6),Y	écrit octet selon cible + Y
C7B0a	C8	INY	indexe octet suivant
C7B1a	D0 F9	BNE C7AC	reboucle en C7AC tant que la page n'est pas finie

Tranfère les pages complètes

Afin d'accélérer le processus, deux octets sur 256 sont copiés à chaque tour.

C7B3a	E6 F5	INC F5	indexe page suivante en lecture
C7B5a	E6 F7	INC F7	indexe page suivante en écriture
C7B7a	CA	DEX	décrémente le nombre de pages à copier
C7B8a	F0 3C	BEQ C7F6	toutes les pages ont été copiées, termine en C7F6
C7BAa	B1 F4	LDA (F4),Y	lit octet selon source + Y
C7BCa	91 F6	STA (F6),Y	écrit octet selon cible + Y
C7BEa	C8	INY	indexe octet suivant
C7BFa	B1 F4	LDA (F4),Y	lit octet selon source + Y
C7C1a	91 F6	STA (F6),Y	écrit octet selon cible + Y
C7C3a	C8	INY	indexe octet suivant
C7C4a	D0 F4	BNE C7BA	reboucle en C7BA tant que la page n'est pas finie
C7C6a	F0 EB	BEQ C7B3	reboucle en C7B3 pour indexer page suivante

C7C8a	4C 20 DE	JMP DE20	"ILLEGAL QUANTITY ERROR"

MOVE ascendant
F6/F7 >= F2/F3 (cas où C = 1) Il faut commencer le transfert par la fin.

```
DEBBLOC      DEBCIBL      FINBLOC          ->          FINCIBLE
  |            |            |                         |
  V            V            V                         V
==========================================================================
```

==

F2/F3	F6/F7	F4/F5		
		DEBBLOC+XY		DEBCIBL+XY
		(DEBBLOC+X),Y	->	(DEBCIBL+X),Y

Explication: On ne peut opérer comme précedemment sous peine d'écraser le bloc source pendant la copie. Il faut lire les octets un à un à partir de FINBLOC et les copier à partir de FINCIBL. Sachant que LGNBLOC, la longueur du bloc à transférer est de YX octets, il faut copier X "pages" complètes et une page incomplète de Y octets.

Au cours de ce MOVE ascendant, dans la boucle LDA (FINBLOC),Y STA(DEBCIBL),Y l'index Y sera décrémenté. On calcule les adresses source et cible de la manière suivante:
FINBLOC = DEBBLOC + LNGBLOC = F2/F3 + YX qui s'écrit aussi F2/F3 + X + Y. Pour lire les octets, on aura donc un LDA(F2/F3+X),Y

De même, FINCIBL = DEBCIBL + LGNBLOC = F6/F7 + YX = F6/F7 +X + Y. Pour écrire les octets, on aura donc STA(F6/F7+X),Y.

Calcule les adresses source et cible

7CBa	8A	TXA	X = nombre de pages entières à transférer
7CCa	18	CLC	calcule F3 = F3 + X
7CDa	65 F3	ADC F3	(les HH suffisent)
7CFa	85 F3	STA F3	
7D1a	8A	TXA	
7D2a	18	CLC	
7D3a	65 F7	ADC F7	De même, calcule F7 = F7 + X
7D5a	85 F7	STA F7	

Transfère la page incomplète

7D7a	E8	INX	nombre total de pages (partielle + entières)
			La première page copiée sera partielle puisque Y < 256.
7D8a	88	DEY	ce qui explique pourquoi l'octet de FINBLOC est exclu du transfert.
			Il manque un INY avant le début de cette boucle.
7D9a	B1 F2	LDA (F2),Y	lit octet selon le pointeur source + Y
7DBa	91 F6	STA (F6),Y	écrit octet selon le pointeur cible + Y
7DDa	98	TYA	teste Y
7DEa	D0 F8	BNE C7D8	reboucle en C7D8 si page pas finie

Transfère les pages complètes

Afin d'accélérer le processus, deux octets sur 256 sont copiés à chaque tour.

7E0a	C6 F3	DEC F3	indexe page précédente en lecture
7E2a	C6 F7	DEC F7	indexe page précédente en écriture
7E4a	CA	DEX	décrémente le nomble de pages à déplacer
7E5a	F0 0F	BEQ C7F6	termine en C7F6 s'il n'en reste plus
7E7a	88	DEY	
7E8a	B1 F2	LDA (F2),Y	lit octet selon le pointeur source + Y
7EAa	91 F6	STA (F6),Y	écrit octet selon le pointeur cible + Y
7ECa	88	DEY	
7EDa	B1 F2	LDA (F2),Y	lit octet selon le pointeur source + Y
7EFa	91 F6	STA (F6),Y	écrit octet selon le pointeur cible + Y
7F1a	98	TYA	teste Y
7F2a	D0 F3	BNE C7E7	reboucle en C7E7 tant que la page n'est pas finie
7F4a	F0 EA	BEQ C7E0	reboucle en C7E0 pour indexer la page suivante

Fini

7F6a	28	PLP	récupère les indicateurs 6502
7F7a	60	RTS	et retourne (fin de MOVE)
7F8a	4C B4 04	JMP 04B4	NMIRAM
7FBa	84 00 00 00 00		5 octets semblent inutilisés = libres

Banque n°2 (adresse Cxxxb): BACKUP

Cette banque se trouve à partir du #47 (71ème) secteur de la disquette MASTER

C400b	00 00	EXTER	adresse des messages d'erreur externe (néant)
C402b	B0 C5	EXTMS	adresse des messages externes D'autres messages ont été placés dans une zone supplémentaire de C6A7 à C6EF!

EXECUTION COMMANDE SEDORIC BACKUP

Rappel de la syntaxe
BACKUP (lecteur source) (TO lecteur cible)
Effectue une copie conforme de la disquette présente dans le lecteur source sur la disquette présente dans le lecteur cible.

Variables utilisées

0A/0B	pointeur dans le tampon de formatage
0C	nombre d'octets à copier
F2	longueur de la chaîne à saisir
F3	options pour XLINPU
F4	mode de sortie de XLINPU
F5	nombre de pistes/face
F6	nombre de secteurs restant à écrire dans une piste
F7	n° de secteur à écrire dans le tampon de formatage
F8	n° de face (#00 si première, 01 si deuxième face) longueur de la chaîne à saisir
F9	nombre de secteurs à copier sur la disquette position de la chaîne saisie dans BUF1
C000	DRIVE
C001	PISTE
C002	SECTEUR
C003/C004	RWBUF
C04C	DEFAFF, code ASCII devant les nombres décimaux
C072	b7 à 1 si "monodrive"
C073	b7 à 1 si "source in drive"
C074	b7 à 1 si "format" (formatage demandé)
C0F9	n° du drive source
C0FA	n° du drive cible
C0FB/C0FC	nombre de secteurs/face, nombre de secteurs restant à BACKUPer
C0FD	HH de la taille du tampon de BACKUP
C200/C2FF	BUF2
C5AE	n° de la piste où l'on en est
C5AF	n° du secteur où l'on en est
C5B0	HH de la taille du tampon de BACKUP
C671/C69A	table de formatage
C698	index de base pour gaps
C69B/C6A6	table des variables internes
C69Bb 00 98	adresse pour préparer en RAM une piste complète avec ses "gaps"
C69Db 00 B1	adresse de fin de ce tampon de préparation de piste
C69Fb 70	#10 si 18 ou 19 secteurs/piste ou #70 si 16 ou 17 secteurs/piste
C6A0b 98	#98 est le HH de l'adresse du tampon de formatage
C6A1b 64	index élargi pour "gaps" = index de base + #3C
C6A2b 01	HH de cet index élargi (#100 octets pour les data)
C6A3b 11	nombre de secteurs par piste (repris dans F6)
C6A4b 12	nombre de secteurs par piste + #01
C6A5b 00	nombre de pistes/face (repris dans F5) [et nombre de faces]
C6A6b 00	nombre de faces: #00 si une face et #01 si deux faces
C6A7/C6EF	autres messages

Informations non documentées
Bogue monumentale, due à un défaut de la commande INIT: le flag "D" n'étant pas correctement mis à jour, BACKUP se contente de copier la première face des disquettes pour lesquelles DIR affiche "S". Correction possible: voir plus loin.

Autre bogue (plus commune celle-là): bien que les commandes Sédoric soient sensées être acceptée indifféremment en minuscules ou en majuscules, le TO doit obligatoirement être en majuscules. En effet, backup A TO b marche, mais BACKUP A to B déclenche une SYNTAX ERROR! Curieusement, BACKUP TO est accepté et donne la même chose que BACKUP tout court.

Après un BACKUP, il est possible de récupérer un programme BASIC (pourvu qu'il ne dépasse pas l'adresse #5FF), à l'aide de la commande OLD.

Enfin le mode HIRES n'est pas autorisé pendant un BACKUP.

Utilisation en LM

Il doit être possible d'effectuer un BACKUP à partir d'un programme en langage machine en faisant un JSR F151 après avoir basculé sur la RAMOV. Si ce BACKUP ne concerne pas le drive courant, il sera nécessaire, avant ce JSR, d'écrire les paramètres source et cible dans le tampon clavier, d'initialiser TXTPTR (voir les détails en annexe).

Analyse de syntaxe

404b	20 DE DF	JSR DFDE	vérifie si bien en mode TEXT
407b	20 0D E6	JSR E60D	valide drive indiqué à TXTPTR, sinon valide DRVDEF
40Ab	8C F9 C0	STY C0F9	n° du drive source
40Db	F0 05	BEQ C414	saute les 2 instructions suivantes si fin des paramètres
40Fb	A9 C3	LDA #C3	token "TO" sera demandé à TXTPTR (bogue usuelle: le "to" en minuscules ne sera pas pris en compte et déclenchera une belle "SYNTAX ERROR")
411b	20 2E D2	JSR D22E	JSR D067/ROM et D3A1/RAMOV demande "TO" à TXTPTR, lit le caractère suivant à TXTPTR et le convertit en MAJUSCULE
414b	20 0D E6	JSR E60D	valide drive indiqué à TXTPTR, sinon valide DRVDEF
417b	8C FA C0	STY C0FA	n° du drive cible

Formatage?

41Ab	A2 06	LDX #06	indexe le message "CRLFFormat TARGET disc (Y/N)**:**"
41Cb	20 64 D3	JSR D364	XAFSC affiche X+1ème message externe terminé par "caractère + 128"
41Fb	20 02 D3	JSR D302	JSR EB78/ROM si touche frappée alors N = 1 et A = code ASCII corresp, sinon N = 0
422b	10 FB	BPL C41F	reboucle tant qu'une touche n'a pas été pressée
424b	20 A1 D3	JSR D3A1	minMAJ convertit en MAJUSCULE le caractère dans A
427b	C9 59	CMP #59	est-ce un "Y"?
429b	F0 08	BEQ C433	si oui, continue en C433 avec C = 1 (flag "format")
42Bb	C9 1B	CMP #1B	est-ce "ESC"?
42Db	D0 01	BNE C430	sinon, saute l'instruction suivante
42Fb	60	RTS	si oui, simple RTS, c'est fini
430b	18	CLC	flag "format" OFF
431b	A9 4E	LDA #4E	lettre "N"
433b	6E 74 C0	ROR C074	le b7 de C074 porte le flag "format"
436b	20 2A D6	JSR D62A	XAFCAR affiche le caractère ASCII contenu dans A
439b	20 06 D2	JSR D206	CBF0/ROM va à la ligne
43Cb	20 06 D2	JSR D206	CBF0/ROM va à la ligne

Monodrive?

43Fb	AD F9 C0	LDA C0F9	compare n° drive source
442b	CD FA C0	CMP C0FA	et n° drive cible
445b	18	CLC	flag "monodrive" OFF par défaut (deux drives)
446b	D0 01	BNE C449	saute l'instruction suivante s'il y a deux drives
448b	38	SEC	flag "monodrive" ON (un seul drive)
449b	6E 72 C0	ROR C072	le b7 de C072 porte le flag "monodrive"
44Cb	30 18	BMI C466	si monodrive, continue en C466

Demande les deux disquettes si "monodrive" OFF

44Eb	A2 01	LDX #01	indexe le message "LOAD DISCS FOR BACKUP FROM "
450b	20 64 D3	JSR D364	XAFSC affiche X+1ème message externe terminé par "caractère + 128"
453b	AD F9 C0	LDA C0F9	n° du drive source
456b	20 0E D6	JSR D60E	convertit n° lecteur en lettre et l'affiche
459b	A2 02	LDX #02	indexe le message " TO "
45Bb	20 64 D3	JSR D364	XAFSC affiche X+1ème message externe terminé par "caractère + 128"
45Eb	AD FA C0	LDA C0FA	n° du drive cible
461b	20 0E D6	JSR D60E	convertit n° lecteur en lettre et l'affiche
464b	D0 14	BNE C47A	suite forcée en C47A avec C = 0

Demande la disquette source si "monodrive" ON

466b	A2 03	LDX #03	indexe le message "CRLFLOAD SOURCE DISC "
468b	20 64 D3	JSR D364	XAFSC affiche X+1ème message externe terminé par "caractère + 128"
46Bb	A2 00	LDX #00	indexe le message "IN DRIVE "
46Db	20 64 D3	JSR D364	XAFSC affiche X+1ème message externe terminé par "caractère + 128"
470b	AD F9 C0	LDA C0F9	n° du drive source
473b	20 0E D6	JSR D60E	convertit n° lecteur en lettre et l'affiche
476b	38	SEC	force C = 1 ("la disquette source est en place")
477b	6E 73 C0	ROR C073	force à 1 le b7 de C073 (flag "source in drive")

Lit le format de la disquette source

47Ab	A2 05	LDX #05	indexe le message "CRLFAND PRESS RETURN "
47Cb	20 64 D3	JSR D364	XAFSC affiche X+1ème message externe terminé par "caractère + 128"

C47Fb 20 69 D6 JSR D669 demande un 'ESC' (C = 1) ou un 'RETURN' (C = 0)
C482b 90 01 BCC C485 si "RETURN", saute l'instruction suivante
C484b 60 RTS si "ESC", simple RTS, c'est fini
C485b AD F9 C0 LDA C0F9 n° du drive source
C488b 8D 00 C0 STA C000 devient DRIVE actif
C48Bb 20 4C DA JSR DA4C XPMAP prend le secteur de bitmap dans BUF2

Effectue un NEW
C48Eb A9 00 LDA #00
C490b A0 01 LDY #01 force à 0 l'octet de poids fort du 1er lien (NEW)
C492b 91 9A STA (9A),Y
C494b 20 B4 E0 JSR E0B4 restaure les pointeurs BASIC

Une ou deux faces?
Attention, la bogue de INIT (qui ne met pas à jour le flag "Double face" lorsqu'il y a formatage) se répercute ici. En effet la commande BACKUP ne marche correctement que si ce flag est correct. Si vous avez à faire un BACKUP d'une disquette double face précieuse et que le directory affiche "S" au lieu de "D", il faut changer le flag "Double face" de cette disquette. Pour corriger ce flag sans avoir à repasser par INIT, utilisez un éditeur de disquette (BDDISK par exemple) pour mettre à 1 le b7 du 10ème octet (n°#09) du secteur de bitmap (2ème secteur, piste 20). Par exemple, pour une disquette formatée en 42 pistes, cet octet indique #2A et il faudra le mettre à #AA.
C497b A9 00 LDA #00 n° de la première piste de la première face
C499b 2C 09 C2 BIT C209 teste b7 du 10ème octet de BUF2 (à 1 si double face)
C49Cb 10 05 BPL C4A3 saute les 2 instructions suivantes si nul (simple face)
C49Eb 20 A3 C4 JSR C4A3 BACKUP de la 1ère face lorsqu'il y en a deux
C4A1b A9 80 LDA #80 n° de la première piste de la deuxième face

Préparation des variables pour le BACKUP
C4A3b 8D 01 C0 STA C001 copie le n° de la 1ère piste dans n° de la PISTE
C4A6b 8D AE C5 STA C5AE active et dans le n° de la piste où l'on en est
C4A9b AD 09 C2 LDA C209 le b7 de A indique le nombre de face
C4ACb 29 80 AND #80 ne garde que le b7 et force les autres bits à zéro
C4AEb 0D 06 C2 ORA C206 y ajoute le nombre de pistes par face et sauve
C4B1b 8D A5 C6 STA C6A5 en C6A5 (nombre de pistes et nombre de faces)
C4B4b A9 51 LDA #51 A = "complément" pour écriture avec formatage
C4B6b 2C 74 C0 BIT C074 teste si le flag "formatage" est nul
C4B9b 10 02 BPL C4BD si oui (pas de formatage), saute l'instruction suivante
C4BBb A9 6E LDA #6E A = "complément" pour écriture sans formatage
C4BDb A2 FF LDX #FF HH du totalisateur. En fait, cela revient à retirer #100 du total, car à la 1ère incrémentation Y passera à #00 au lieu de #01
C4BFb AC 07 C2 LDY C207 8ème octet de bitmap: nombre de secteurs par piste
C4C2b 8C A3 C6 STY C6A3 sauve ce nombre de secteurs par piste en C6A3, puis calcule le nombre AX de secteurs par face + un "complément" (#51 ou #6E) - #100 c'est à dire le nombre AX de secteurs par face - #AF ou - #92 (qui représentent le HH de la taille du tampon de BACKUP avec et sans formatage).
C4C5b 18 CLC pour addition de Y fois le nombre de pistes/faces
C4C6b 6D 06 C2 ADC C206 "complément" + nombre de pistes par face
C4C9b 90 01 BCC C4CC saute l'instruction suivante si C = 0
C4CBb E8 INX incrémente le HH du totalisateur si A > #FF
C4CCb 88 DEY décrémente le nombre de secteurs par piste restant
C4CDb D0 F6 BNE C4C5 reboucle tant qu'il en reste à ajouter
C4CFb 8D FB C0 STA C0FB C0FB/C0FC représente le nombre de secteurs/face
C4D2b 8E FC C0 STX C0FC restant à BACKUPer. Comme il ne sera pris en compte qu'après un premier chargement du buffer de BACKUP, on a déjà retiré le HH de la taille de celui-ci. Par exemple, pour une face de 42 pistes de 17 secteurs, AX vaudra soit #21B (avec formatage) soit #238 (sans formatage).
C4D5b C8 INY Y = #01
C4D6b 8C 02 C0 STY C002 n° de SECTEUR actif de départ
C4D9b 8C AF C5 STY C5AF n° du secteur où l'on en est (ici n° de départ)
C4DCb 88 DEY Y = #00 (flag "écriture", est à zéro si lecture)
C4DDb A9 92 LDA #92 HH de la taille du tampon de BACKUP si formatage
C4DFb 2C 74 C0 BIT C074 teste si le flag "formatage" est à 1
C4E2b 30 02 BMI C4E6 si oui (formatage), saute l'instruction suivante

Point de rebouclage n°1
C4E4b A9 AF LDA #AF HH taille du tampon de BACKUP si pas de formatage. Revient ici à la fin de chaque cycle de remplissage/écriture du tampon de BACKUP, afin de réinitialiser la taille du tampon de BACKUP (utile notamment si le premier cycle comportait un formatage et donc un tampon plus petit), sauf pour le dernier cycle pour lequel la taille du tampom est ajustée exactement au nombre de secteurs restant a BACKUPer.

Point de rebouclage n°2
a) Revient ici après chaque remplissage du tampon de BACKUP pour passer à l'écriture de ce tampon avec A ayant la même valeur que lors du remplissage.

b) Revient ici au dernier tour de chaque face avec A = nombre de secteurs restants pour fixer exactement la taille du dernier tampon de BACKUP.

```
4E6b 8D FD C0   STA C0FD    sauve HH de la taille du tampon de BACKUP en C0FD
4E9b 8D B0 C5   STA C5B0    sauve HH de la taille du tampon de BACKUP en C5B0
4ECb 2C 72 C0   BIT C072    teste si le flag "monodrive" est à zéro
4EFb 10 1F      BPL C510    si oui (2 drives), continue en C510
4F1b 2C 73 C0   BIT C073    sinon (monodrive), teste le flag "source in drive"
4F4b 30 1A      BMI C510    continue en C510 si disquette source est en place. C'est le cas au 1er tour, car la disquette
```
source a déjà été demandée pour lire le format. Avant l'éventuel formatage, un premier chargement du tampon de BACKUP sera effectué dans la foulée de la lecture de la bitmap source.

Demande la disquette source si lecture ou cible si écriture

```
4F6b 20 06 D2   JSR D206    CBF0/ROM va à la ligne
4F9b A2 03      LDX #03     pour 4ème message externe "CRLFLOAD SOURCE DISC "
4FBb 98         TYA         teste Y (flag "écriture", à zéro si lecture)
4FCb 48         PHA         empile Y temporairement
4FDb F0 01      BEQ C500    saute l'instruction suivante si Y = 0 (lecture)
4FFb E8         INX         indexe le message "CRLFLOAD TARGET DISC "
500b 20 64 D3   JSR D364    XAFSC affiche X+1ème message externe terminé par "caractère + 128"
503b A2 05      LDX #05     indexe le message "CRLFAND PRESS RETURN "
505b 20 64 D3   JSR D364    XAFSC affiche X+1ème message externe terminé par "caractère + 128"
508b 68         PLA
509b A8         TAY         récupère Y
50Ab 20 69 D6   JSR D669    demande un 'ESC' (C = 1) ou un 'RETURN' (C = 0)
50Db 90 01      BCC C510    si "RETURN", saute l'instruction suivante
50Fb 60         RTS         si "ESC", simple RTS, c'est fini
```

Formatage?

```
510b 78         SEI         interdit les interruptions
511b B9 F9 C0   LDA C0F9,Y  n° du drive source si Y = #00 ou cible si Y = #01
514b 8D 00 C0   STA C000    devient le n° du DRIVE actif
517b 2C 74 C0   BIT C074    teste si le flag "formatage" est à zéro
51Ab 10 12      BPL C52E    si oui (pas de formatage), continue en C52E
51Cb 98         TYA         sinon, teste si Y est à zéro, c'est le cas tout au début, un chargement du tampon de BACKUP
```
sera effectué avant le formatage de la cible afin d'éviter un changement de disquette supplémentaire.

```
51Db F0 0F      BEQ C52E    si oui (lecture), continue en C52E
51Fb 4E 74 C0   LSR C074    sinon (écriture), force à zéro par avance le flag
522b 20 41 C6   JSR C641    "formatage" et effectue le formatage, puis force
525b 4E 73 C0   LSR C073    à 0 le flag "source in drive" (car cible en place)
528b A0 00      LDY #00     pour n° de piste de départ
52Ab 8C 01 C0   STY C001    piste n° #00 devient PISTE active
52Db C8         INY         Y = #01 (flag "écriture", à 1 si écriture)
```

BACKUP proprement dit, reset RWBUF et "source in drive"

```
52Eb A9 00      LDA #00
530b 8D 03 C0   STA C003    RWBUF = 0600 adresse du tampon de BACKUP
533b A9 06      LDA #06
535b 8D 04 C0   STA C004
538b 4E 73 C0   LSR C073    force à zéro le flag "source in drive"
```

Lit ou écrit un secteur selon DRIVE, PISTE, SECTEUR et RWBUF

```
53Bb 98         TYA         teste si Y est nul
53Cb F0 05      BEQ C543    si oui (lecture) continue en C543, sinon...
53Eb 20 A4 DA   JSR DAA4    XSVSEC écrit un secteur selon DRIVE, PISTE, SECTEUR et RWBUF
541b F0 03      BEQ C546    et saute l'instruction suivante
543b 20 73 DA   JSR DA73    XPRSEC lit un secteur selon DRIVE, PISTE, SECTEUR et RWBUF
```

Indexe le secteur suivant et reboucle si nécessaire

```
546b EE 04 C0   INC C004    page suivante (incrémente le HH de RWBUF)
549b AE 02 C0   LDX C002    n° du dernier SECTEUR actif
54Cb EC 07 C2   CPX C207    teste si la valeur maximale est atteinte
54Fb D0 05      BNE C556    sinon, saute les deux instructions suivantes
551b EE 01 C0   INC C001    indexe la PISTE suivante
554b A2 00      LDX #00     reset le n° de secteur (à 1 en début de boucle)
556b E8         INX         indexe le secteur suivant
557b 8E 02 C0   STX C002    qui devient le SECTEUR actif
55Ab CE FD C0   DEC C0FD    décrémente le HH de la taille du tampon de BACKUP
55Db D0 DC      BNE C53B    reboucle en C53B si pas plein, sinon...
```

Bascule lecture/écriture

C55Fb	58	CLI	autorise les interruptions
C560b	98	TYA	
C561b	49 01	EOR #01	bascule le flag lecture/écriture
C563b	A8	TAY	
C564b	D0 37	BNE C59D	si écriture, continue en C59D
C566b	AD FC C0	LDA C0FC	si lecture, teste si le b7 de C0FC est à 1, c'est à dire s'il n'y a plus de secteurs à BACKUPer
C569b	30 24	BMI C58F	si c'est le cas, continue en C58F

Ajuste et sauve les variables pour le tour suivant

C56Bb	AD 01 C0	LDA C001	sinon, sauve le n° de PISTE active
C56Eb	AE 02 C0	LDX C002	et le n° de SECTEUR actif
C571b	8D AE C5	STA C5AE	dans n° de PISTE où l'on en est
C574b	8E AF C5	STX C5AF	dans n° de SECTEUR où l'on en est
C577b	38	SEC	
C578b	AD FB C0	LDA C0FB	
C57Bb	E9 AF	SBC #AF	C0FB/C0FC = C0FB/C0FC - #AF
C57Db	8D FB C0	STA C0FB	décrémente le nombre de secteurs restant à
C580b	B0 0A	BCS C58C	BACKUPer pour cette face
C582b	CE FC C0	DEC C0FC	
C585b	10 05	BPL C58C	continue en C58C (en fait en C4E4), si le résultat est positif, c 'est à dire s'il en reste encore après ce prochain tour
C587b	69 AF	ADC #AF	si le résultat est négatif, calcule A = A + #AF = nombre de secteurs restant à BACKUPer qui servira à fixer le HH de la taille du tampon de BACKUP pour le dernier. NB: C0FC reste avec son b7 à 1, ce qui servira de marqueur de fin de BACKUP pour la face en cours.
C589b	4C E6 C4	JMP C4E6	et reprend en C4E6 (avec la valeur de A actuelle)
C58Cb	4C E4 C4	JMP C4E4	reprend en C4E4 (où A sera mis à #AF)

Teste si fini ou s'il y a encore une face à copier

C58Fb	AD 01 C0	LDA C001	teste si le b7 du n° de PISTE active
C592b	4D 09 C2	EOR C209	est différent du b7 du n° de piste maximum
C595b	30 05	BMI C59C	si oui (il y a encore une face à copier), simple RTS
C597b	A2 07	LDX #07	sinon, indexe le message "CRLFLFBackup completeCR**LF**"
C599b	4C 64 D3	JMP D364	XAFSC affiche X+1ème message externe terminé par "caractère + 128"
C59Cb	60	RTS	

Ecriture

C59Db	AD AE C5	LDA C5AE	le n° de piste où l'on en est
C5A0b	AE AF C5	LDX C5AF	et le n° de secteur où l'on en est
C5A3b	8D 01 C0	STA C001	deviennent le n° de PISTE active
C5A6b	8E 02 C0	STX C002	et le n° de SECTEUR actif
C5A9b	AD B0 C5	LDA C5B0	HH de la taille du tampon de BACKUP
C5ACb	D0 DB	BNE C589	reprise forcée en C589 (en fait en C4E6)
C5AEb	00	BRK	n° de la piste où l'on en est
C5AFb	00	BRK	n° du secteur où l'on en est
C5B0b	00	BRK	HH de la taille du tampon de BACKUP

Messages externes de la banque n°2 (C5B1 à C640)

C5B1b	49 4E 20 44 52 49 56 45 **A0**	
01	IN DRIVE■	
C5BAb	4C 4F 41 44 20 44 49 53 43 53 20 46 4F 52 20 42 41 43 4B 55 50 20 46 52 4F 4D **A0**	
02	LOAD DISCS FOR BACKUP FROM■	
C5D5b	20 54 4F **A0**	
03	■TO■	
C5D9b	0D 0A 4C 4F 41 44 20 53 4F 55 52 43 45 20 44 49 53 43 **A0**	
04	CRLFLOAD SOURCE DISC■	
C5ECb	0D 0A 4C 4F 41 44 20 54 41 52 47 45 54 20 44 49 53 43 **A0**	
05	CRLFLOAD TARGET DISC■	
C5FFb	0D 0A 41 4E 44 20 50 52 45 53 53 20 52 45 54 55 52 4E **A0**	
06	CRLFAND PRESS RETURN■	
C612b	0D 0A 46 6F 72 6D 61 74 20 54 41 52 47 45 54 20 64 69 73 63 20 28 59 2F 4E 29 **BA**	
07	CRLFFormat TARGET disc (Y/N)**:**	
C62Db	0D 0A 0A 42 61 63 6B 75 70 20 63 6F 6D 70 6C 65 74 65 0D **8A**	
08	CRLFLFBackup completeCR**LF**	

Formate la ou les faces

C641b	0E A5 C6	ASL C6A5	b7 -> C (à zéro si "Simple", à 1 si "Double" face)
C644b	A9 00	LDA #00	force A à zéro
C646b	2A	ROL	C -> b0 de A et b7 de A (nul) -> C (important)

```
647b  8D A6 C6   STA C6A6   C6A6 = #00 si une face et #01 si deux faces
64Ab  A9 A7      LDA #A7    AY = adresse C6A7 de la chaîne:
64Cb  A0 C6      LDY #C6    "CRLFLFFormating Side 0 Track 00"
64Eb  20 37 D6   JSR D637   AFSTR affiche chaîne terminée par 0 d'adresse AY
651b  20 40 D7   JSR D740   "CURSEUR OFF" (curseur caché = vidéo normale)
654b  4E A5 C6   LSR C6A5   rétablit nombre de pistes/face (sans nombre face)
657b  20 3C C7   JSR C73C   formate la première face (car C vaut toujours 0)
65Ab  AD A6 C6   LDA C6A6   teste si une seule face
65Db  F0 0B      BEQ C66A   si oui, continue en C66A
65Fb  A9 C4      LDA #C4    sinon, AY = C6C4 pour chaîne
661b  A0 C6      LDY #C6    "<- <- <- <- <- <- <- <- <- <- 1 Track 00"
663b  20 37 D6   JSR D637   AFSTR affiche chaîne terminée par 0 d'adresse AY
666b  38         SEC        C = 1 pour deuxième face
667b  20 3C C7   JSR C73C   formate la deuxième face
66Ab  A9 D9      LDA #D9    AY = adresse C6D9 pour chaîne
66Cb  A0 C6      LDY #C6    "LFLFCRFormating complete"
66Eb  4C 37 D6   JMP D637   AFSTR affiche chaîne d'adresse AY et retourne
```

Table de formatage (C671 à C69A)

```
671b  28 4E 0C 00 03 F6 01 FC 28 4E FF          (soit #60 = 96 octets au total)
67Cb  0C 00 03 F5                               (soit 15 octets)
680b  01 FE 01 00 01 00 01 00 01 01 01 F7       (FE pp ff ss 01 CRC, soit 6 octets)
68Cb  16 4E 0C 00 03 F5                         (soit 37 octets)
692b  01 FB 00 00 01 F7                         (soit 258 octets)
698b  28 4E FF                                  (soit 40, 30 ou 12 octets selon le nombre de secteurs/piste)
```

NB: C698 qui est l'index de base pour "gaps" est variable et vaut #28 si 16 ou 17 secteurs par piste, #1E si 18 secteurs par piste ou #0C si 19 secteurs par piste. Lorsque l'on entre dans la table en C67C (X = #0B), le nombre total d'octets écrits dans le tampon est donc lui aussi variable et vaut #164 = 356 si 16 ou 17 secteurs par piste, #15A = 346 si 18 secteurs par piste ou #148 = 328 si 19 secteurs par piste.

Variables internes à la banque N°2

```
69Bb  00 98      adresse pour préparer en RAM une piste complète avec ses "gaps"
69Db  00 B1      adresse de fin de ce tampon de préparation de piste
69Fb  70         #10 si 18 ou 19 secteurs/piste ou #70 si 16 ou 17 secteurs/piste
6A0b  98         #98 est le HH de l'adresse du tampon de formatage
6A1b  64         index élargi pour "gaps" = index de base + #3C
6A2b  01         HH de cet index élargi (#100 octets pour les data)
6A3b  11         nombre de secteurs par piste (repris dans F6)
6A4b  12         nombre de secteurs par piste + #01
6A5b  00         nombre de pistes/face (repris dans F5) [et nombre de faces]
6A6b  00         nombre de faces: #00 si une face et #01 si deux faces
```

Autres messages (C6A7 à C6EF)

```
6A7b  0D 0A 0A 46 6F 72 6D 61 74 69 6E 67 20 53 69 64 65 20 30 20 54 72 61 63 6B 20 30 30 00
      CRLFLFFormating Side 0 Track 00BRK
6C4b  08 08 08 08 08 08 08 08 08 08 31 20 54 72 61 63 6B 20 30 30 00
      <- <- <- <- <- <- <- <- <- <- 1 Track 00BRK
6D9b  0A 0A 0D 11 46 6F 72 6D 61 74 69 6E 67 20 63 6F 6D 70 6C 65 74 65 00
      LFLFCR.Formating completeBRK
```

Met à jour les n° de piste n° de face et n° de secteur

```
6F0b  AD 9F C6   LDA C69F   #10 si 18 ou 19, #70 si 16 ou 17 secteurs/piste
6F3b  AC A0 C6   LDY C6A0   #98 HH de l'adresse du tampon de formatage
6F6b  85 0A      STA 0A     0A/0B = #9810 ou #9870 = pointeur dans ce tampon
6F8b  84 0B      STY 0B     (c'est l'adresse du 1er n° de piste)
6FAb  A2 00      LDX #00    compteur du nombre de secteurs déjà préparés
6FCb  A0 00      LDY #00    index écriture dans chaque zone de préparation
6FEb  AD 01 C0   LDA C001   n° de PISTE active
701b  29 7F      AND #7F    dont on force à zéro le b7 (flag nombre de faces)
703b  91 0A      STA (0A),Y écrit le n° de piste #pp au pointeur + Y
705b  C8         INY        position suivante
706b  A5 F8      LDA F8     A = n° de face
708b  91 0A      STA (0A),Y écrit le n° de face #ff au pointeur + Y
70Ab  C8         INY        position suivante
70Bb  A5 F7      LDA F7     A = n° de secteur
70Db  18         CLC        prépare une addition
70Eb  69 01      ADC #01    A = n° de secteur + #01
710b  20 31 C7   JSR C731   mise à jour éventuelle de F7. Le premier secteur d'une piste n'est pratiquement jamais le n°1
```

(voir plus bas). Supposons qu'une piste à formater en 17 secteurs par piste commence au n°2, par incrémentations

successives, le dernier secteur aurait le n°18. Comme cela n'est pas possible, ce n° est ramené à 1 par le sous-programme C731.

C713b	91 0A	STA (0A),Y	écrit le n° de secteur #ss au pointeur + Y
C715b	18	CLC	prépare une addition
C716b	AD A1 C6	LDA C6A1	LL de l'index élargi pour "gaps"
C719b	65 0A	ADC 0A	mise à jour du pointeur dans le tampon
C71Bb	85 0A	STA 0A	adresse = adresse + index élargi pour "gaps"
C71Db	AD A2 C6	LDA C6A2	HH de l'index élargi, augmente le pointeur d'une page
C720b	65 0B	ADC 0B	corresp aux 256 octets de data du secteur
C722b	85 0B	STA 0B	(adresse en 0A/0B vise le #pp du secteur suivant)
C724b	E8	INX	compteur du nombre de secteurs déjà préparés
C725b	EC A3 C6	CPX C6A3	teste si X atteint le nombre de secteurs par piste
C728b	D0 D2	BNE C6FC	sinon, reboucle en C6FC

Calcul du 1er n° de secteur de la piste suivante

C72Ab	A5 F7	LDA F7	n° du secteur qui vient d'être préparé auquel on
C72Cb	6D A3 C6	ADC C6A3	ajoute le nombre de secteurs par piste et on
C72Fb	E9 04	SBC #04	retranche #04 (les pistes ne commencent pas toutes au secteur n°1, mais selon un ordre plus complexe probablement afin d'avoir une plus grande rapidité d'accès. Les pistes commencent successivement aux secteurs 1, 14, 10, 6, 2, 15, 11, 7, 3, 16, 12, 8, 4, 17, 13, 9, 5, 1 etc... Ainsi, la piste n° 20 d'une disquette commence avec le secteur n°6!)

Mise à jour éventuelle de F7

Pour que #01 =< n° du secteur à écrire =< nombre de secteurs par piste

C731b	CD A4 C6	CMP C6A4	teste si A < nombre de secteurs par piste + #01
C734b	90 03	BCC C739	si oui (C = 0), saute l'instruction suivante
C736b	ED A3 C6	SBC C6A3	sinon (C = 1), retranche de A le nombre maximum de secteurs par piste. Exemple: lors du formatage en 17 secteurs par piste, lorsque A atteint 18, on retranche 17 et le n° est ramené à 1.
C739b	85 F7	STA F7	remet le résultat en place dans F7
C73Bb	60	RTS	et retourne

Rappels sur la structure d'une piste

a) Une piste Sédoric est formée de 16, 17, 18 ou 19 secteurs de 256 octets. Entre ces secteurs de data proprement dits, se trouvent des "gaps" qui contiennent des informations utiles pour le contrôleur de lecteur.

b) Le début d'une piste (facultatif) commence par une série de 80 [#4E], 12 [#00], [#C2 #C2 #C2 #FC], puis par une série de 50 [#4E] (selon la norme IBM) ou 40 [#4E], 12 [#00], [#C2 #C2 #C2 #FC] et 40 [#4E] (Sédoric, si 16 ou 17 secteurs par piste, sinon rien), soit une économie de 50 à 146 octets.

c) Chaque secteur est alors précédé d'un champ d'identification formé de: 12 [#00], la séquence de synchronisation [#A1 #A1 #A1], [#FE pp ff ss tt CRC CRC] puis 22 octets [#4E]. Le champ de data est constitué de 12 octets [#00], la séquence de synchronisation [#A1 #A1 #A1], le marqueur de début de data [#FB] et enfin les 256 octets du secteur. Chaque secteur est suivi de 80 octets [#4E] (selon la norme IBM et 40, 30 ou 12 octets [#4E] dans le cas de Sédoric, selon le nombre de secteurs par piste), soit une économie de 12 à 42 octets par secteurs. Puis vient le secteur suivant... (NB: #pp = n°piste, #ff = n°face, #ss = n°secteur, #tt = taille (#01 pour les 256 octets du Sédoric, #02 pour les 512 octets de l'IBM PC etc...)

d) La fin de piste est marquée par un nombre très variable d'octets [#4E] (facultatif). La piste étant circulaire, toutes les valeurs entre la fin de piste et le début de piste sont sans signification.

e) Selon le nombre de secteurs par piste, la place disponible pour les "gaps" est variable. Toutes ces indications sont théoriques, lorsqu'on lit une piste et ses "gaps" avec un utilitaire spécialisé tel que NIBBLE, on obtient des différences. Le premier des 3 octets de synchronisation par exemple est toujours faux, puisque la synchronisation n'a pas encore été obtenue! De plus, la zone située entre la fin des data et les octets de synchronisation de l'en-tête du secteur suivant (soit le "gap" situé entre deux secteurs) contient souvent n'importe quoi. En fait, ni le contrôleur de drive, ni le drive lui-même, ne répondent instantanément. Il s'ensuit des bavures lors des changements d'état de la tête de lecture/écriture. C'est la raison d'être de ces "gaps", qui servent à protéger le secteur suivant. Si l'on voulait augmenter le nombre de secteurs par piste, il faudrait diminuer la taille des "gaps" et donc la fiabilité.

Soit en résumé:

Début de la piste (facultatif): 80 [#4E], 12 [#00], [#C2 #C2 #C2 #FC] et 50 [#4E] (soit 146 octets selon la norme IBM) ou 40 [#4E], 12 [#00], [#C2 #C2 #C2 #FC] et 40 [#4E] (soit 96 octets pour Sédoric).

Pour chaque secteur: 12 [#00], 3 [#A1] [#FE #pp #ff #ss #tt CRC], 22 [#4E], 12 [#00], 3 [#A1], [#FB], les 512 octets, [CRC CRC], 80 octets [#4E] (#tt = #02) (soit 141 + 512 = 653 octets selon la norme IBM) ou 12 [#00], 3 [#A1] [#FE #pp #ff #ss #01 CRC CRC], 22 [#4E], 12 [#00], 3 [#A1], [#FB], les 256 octets, [CRC CRC], 12, 30 ou 40 octets [#4E] (selon le nombre de secteurs/piste). Soit environ 256 + (72 à 100) = 328 à 356 octets pour Sédoric.

Fin de la piste (facultatif): un nombre variable d'octets [#4E].

Selon Nibble, une piste IBM compte 146 octets de début de piste + 9 secteurs de 653 octets + 257 octets de fin de piste = 6280 octets. Une piste Sédoric, formatée à 17 secteurs, compte 96 octets de début de piste + 17 secteurs de 358 octets + 98 octets de fin de piste = 6280 octets. Une piste Sédoric, formatée à 19 secteurs, compte 0 octet de début de piste + 19 secteurs de 328 octets + 48 octets de fin de piste = 6280 octets. On comprend mieux le manque de fiabilité du formatage en 19 secteurs/piste dû à la faible largeur des zones de sécurité (12 [#4E] entre chaque secteur et 48 octets entre le dernier et le premier).

Lors de l'élaboration du tampon de formatage Sédoric, les octets #C2 sont remplacés par des octets #F6, les octets #A1 sont remplacés par des octets #F5 et chaque paire de 2 octets [CRC CRC] et remplacée par un octet #F7. Comme on le voit, nombre de variantes sont utilisées, sauf la zone 22 [#4E], 12 [#00], 3 [#A1] qui est strictement obligatoire.

Formate la première face si C = 0 et la deuxième si C = 1
(voir "Rappels sur la structure d'une piste" dans la banque n°6)

73Cb	08	PHP	sauvegarde les indicateurs 6502 dont C
73Db	08	PHP	idem une 2ème fois
73Eb	AD A3 C6	LDA C6A3	
741b	85 F6	STA F6	F6 = nombre de secteurs par piste
743b	8D A4 C6	STA C6A4	
746b	EE A4 C6	INC C6A4	C6A4 = nombre de secteurs par piste + #01. Selon le nombre de secteurs par piste, la place restante pour les codes placés entre les secteurs (dans les gaps) est variable, on détermine:
749b	A0 0C	LDY #0C	Y = #0C (soit 12, valeur pour 19 secteurs/piste)
74Bb	C9 13	CMP #13	teste si A >= #13 (en fait si A = 19 valeur maxi)
74Db	B0 08	BCS C757	si oui, continue en C757 (OK pour Y)
74Fb	A0 1E	LDY #1E	sinon, Y = #1E (30, valeur pour 18 secteurs/piste)
751b	C9 12	CMP #12	teste si A >= #12 (en fait si A = 18)
753b	B0 02	BCS C757	si oui, continue en C757 (OK pour Y)
755b	A0 28	LDY #28	sinon, Y = #28 (soit 40, valeur pour A = 16 ou 17)
755b	A0 2F	LDT #2F	variante dans le cas de Stratoric V1.0
757b	8C 98 C6	STY C698	sauve Y en C698 (index de base)
75Ab	18	CLC	
75Bb	98	TYA	
75Cb	69 3C	ADC #3C	C6A1 = Y + #3C (index élargi)
75Eb	8D A1 C6	STA C6A1	
761b	AD A5 C6	LDA C6A5	
764b	85 F5	STA F5	F5 = nombre de pistes par face
766b	AD 9B C6	LDA C69B	
769b	AC 9C C6	LDY C69C	AY = #9800 (adresse présente en C69B/C69C)
76Cb	85 0A	STA 0A	0A/0B = #9800 (adresse pour préparer en RAM
76Eb	84 0B	STY 0B	une piste complète avec ses "gaps")
770b	8D 03 C0	STA C003	RWBUF = #9800 (adresse du tampon en RAM
773b	8C 04 C0	STY C004	qui sera envoyé sur la disquette)
776b	28	PLP	récupère les indicateurs 6502 dont C
777b	A9 00	LDA #00	force à 0 le registre A
779b	AA	TAX	force X à 0 (index de lecture dans la table C671)
77Ab	A8	TAY	force Y à 0 (index d'écriture dans le tampon 9800)
77Bb	2A	ROL	force le b0 de A selon C donc A = C
77Cb	85 F8	STA F8	F8 = #00 si 1ère face ou #01 si 2ème face
77Eb	28	PLP	récupère les indicateurs 6502 dont C qui passe
77Fb	6A	ROR	dans b7 de A dont l'ancien b0 est éliminé
780b	8D 01 C0	STA C001	donc b7 de PISTE porte C. En clair, si Simple face le n° de la première piste est #00, si Double face, ce n° est #80 (pas mal!)
783b	86 F7	STX F7	F7 = #00 n° du premier secteur
785b	AD A3 C6	LDA C6A3	A = nombre de secteurs par piste
788b	C9 12	CMP #12	teste si A >= #12 (c'est à dire, vaut 18 ou 19)
788b	C9 11	CMP #11	variante dans le cas de Stratoric V1.0
78Ab	B0 06	BCS C792	si oui, continue en C792

Elabore un début de piste dans le tampon de formatage

78Cb	20 DA C7	JSR C7DA	sinon, élabore un en-tête de piste de 96 octets, au début du tampon (de 9800 à 985F), selon les valeurs de la 1ère partie de la table C671 (X = #00). Ceci uniquement lors d'un formatage en 16 ou 17 secteurs par piste (l'en-tête est facultatif).

Ajuste le LL du pointeur de mise à jour

78Fb	A9 70	LDA #70	valeur pour 16 ou 17 secteurs par piste
791b	2C A9 10	BIT 10A9	et continue en C794
792b	A9 10	LDA #10	valeur pour 18 ou 19 secteurs par piste
794b	8D 9F C6	STA C69F	sauve en C69F la valeur retenue

Elabore le reste de la piste dans le tampon de formatage

797b	A2 0B	LDX #0B	dans les 2 cas, en début de boucle, X = #0B (index pour lecture de la deuxième partie de la

table C671), tandis que Y (index d'écriture dans le tampon évoluera de #00 (ou #60 si 16 ou 17 secteurs par piste) à #18FF, lorsqu'il pointera sur la fin du tampon (en B0FF).

C799b 20 DA C7 JSR C7DA écrit un secteur complet avec 256 octets [#00] encadrés par des octets de synchronisation, n° piste, n° secteur etc), dans le tampon en 9800 + Y, en utilisant les valeurs de la 2ème partie de la table C671. Selon le nombre de secteurs par piste (16, 17, 18 ou 19), le nombre total d'octets écrits sera différent (356, 356, 346 ou 328 respectivement). Cette différence porte sur le nombre de "#4E" placés en fin de secteur.

C79Cb C6 F6 DEC F6 décrémente le nombre de secteurs par piste
C79Eb D0 F7 BNE C797 reboucle en C797 tant que F6 n'est pas nul

Elabore une fin de piste dans le tampon de formatage

C7A0b A9 4E LDA #4E A = #4E
C7A2b 91 0A STA (0A),Y écrit #4E selon l'adresse en 0A/0B + Y
C7A4b C8 INY indexe la position suivante
C7A5b D0 FB BNE C7A2 et reboucle en C7A2 tant que Y n'est pas nul
C7A7b E6 0B INC 0B page suivante
C7A9b A6 0B LDX 0B pour test
C7ABb EC 9E C6 CPX C69E teste si HH atteint la valeur en C69E (#B1)
C7AEb D0 F2 BNE C7A2 sinon, reboucle jusqu'à ce que toute la fin du tampon de préparation de piste soit remplie de "#4E"

Formate pour de bon

C7B0b A2 08 LDX #08 probablement pour positionnement
C7B2b 20 CD CF JSR CFCD XRWTS routine de gestion des lecteurs, X = commande
C7B5b 20 F0 C6 JSR C6F0 met à jour les n° de piste, de face et de secteur
C7B8b A2 F0 LDX #F0 probablement commande "formater une piste"
C7BAb 20 75 DA JSR DA75 XRWTS X = commande et traite une éventuelle erreur
C7BDb A9 08 LDA #08 a = "flèche gauche" pour reculer de 2 positions
C7BFb 20 2A D6 JSR D62A XAFCAR affiche le caractère ASCII contenu dans A
C7C2b 20 2A D6 JSR D62A idem
C7C5b A2 30 LDX #30 X = "0"
C7C7b 8E 4C C0 STX C04C DEFAFF, code ASCII devant les nombres décimaux
C7CAb AD 01 C0 LDA C001 n° de PISTE formatée à afficher
C7CDb 29 7F AND #7F élimine le b7 indiquant la face
C7CFb 20 4E D7 JSR D74E affichage en décimal sur 2 digits d'un nombre A
C7D2b EE 01 C0 INC C001 n° de PISTE active suivante à écrire
C7D5b C6 F5 DEC F5 nombre de pistes par face
C7D7b D0 DC BNE C7B5 et reboucle en C7B5 tant qu'il en reste
C7D9b 60 RTS

"Copie" la table C671 + X à l'adresse 9800 + Y

C7DAb BD 71 C6 LDA C671,X lit un octet dans la table C671 à la position X
C7DDb E8 INX indexe la position suivante dans la table
C7DEb C9 FF CMP #FF l'octet lu est-il #FF? (fin de zone)
C7E0b F0 13 BEQ C7F5 si oui, simple RTS en C7F5 (seule sortie du s/p)
C7E2b 85 0C STA 0C sinon, sauve octet en 0C (nombre octets à copier)
C7E4b BD 71 C6 LDA C671,X lit un octet dans la table C671 à la position X
C7E7b E8 INX indexe la position suivante dans la table
C7E8b 91 0A STA (0A),Y copie l'octet lu dans le buffer, à la position Y
C7EAb C8 INY indexe la position suivante dans le buffer
C7EBb D0 02 BNE C7EF saute l'instruction suivante tant que Y ne dépasse pas #FF
(lorsque 256 octets ont été copiés, il faut indexer la page suivante)
C7EDb E6 0B INC 0B indexe la page suivante du buffer (incrémente HH)
C7EFb C6 0C DEC 0C décrémente le nombre d'octets à copier
C7F1b D0 F5 BNE C7E8 reboucle en C7E8 tant qu'il en reste à copier, puis
C7F3b F0 E5 BEQ C7DA reboucle en C7DA à chaque fois que le nombre voulu d'octets a été copié. Par exemple, le premier nombre d'octets à copier était #28 (40) pour X = #00, l'octet suivant #4E sera copié 40 fois à partir du début du tampon (lorsque Y = #00), puis l'octet #00 sera copié 12 fois (#0C), l'octet #F6 3 fois, l'octet #FC 1 fois et enfin l'octet #4E 40 fois. En tout, 96 octets (#60) seront mis en place dans le buffer de l'adresse 9800 à l'adresse 985F (Y = #60 à la fin).
C7F5b 60 RTS

Remarques

1) Comme indiqué dans le manuel, la commande BACKUP a été optimisée pour réduire les manipulations de disquettes lors de l'utilisation "monodrive". Pour cela, d'une part le formatage n'est fait qu'après le premier remplissage du tampon de BACKUP, d'autre part la place du tampon de formatage est récupèrée dès que possible pour le tampon de BACKUP.

2) La bogue de INIT a des effets désastreux sur la commande BACKUP qui ne peut deviner si la disquette à copier est simple ou double face.

3) On peut observer que la partie "formatage" de la commande BACKUP est la réplique exacte de la partie équivalante de

la commande INIT (banque n°6). Notamment, l'adresse du tampon de formatage est la même: 9800 qui aurait pût être portée à 9C00 (gain #400 au premier tour). De même le tampon de BACKUP commence en 0600 et aurait pût commencer en 0500 (gain #100 à tous les tours). Mais il faut avouer que ces gains n'auraient rien changé au nombre de changements de disquettes. Bravo c'est bien ficelé!

Semble être un résidu non utilisé

7F6b	E6 0B	INC 0B
7F8b	C6 0C	DEC 0C
7FAb	D0 F5	BNE C7F1
7FCb	F0 E5	BEQ C7E3
7FEb	60	RTS
7FFb	00	BRK

Banque n°3 (adresse Cxxxc): SEEK, CHANGE et MERGE

Cette banque se trouve à partir du #4C (76ème) secteur de la disquette MASTER

C400c	DC C7	EXTER	adresse des messages d'erreur externe
C402c	B8 C7	EXTMS	adresse des messages externes
C404c	4C 15 C4	JMP C415	Entrée commande SEEK
C407c	4C 25 C5	JMP C525	Entrée commande CHANGE
C40Ac	4C 95 C6	JMP C695	Entrée commande MERGE
C40Dc	4C 77 E9	JMP E977	"STRING TOO LONG"
C410c	A2 1A	LDX #1A	"INVALID STRING ERROR"
C412c	4C 7E D6	JMP D67E	incrémente X et traite l'erreur n° X

EXECUTION COMMANDE SEDORIC SEEK

Rappel de la syntaxe
SEEK (expression alphanumérique) (,S) (,M)
Recherche l'expression alphanumérique indiquée (79 caractères au maximum) dans le programme BASIC et affiche les lignes où elle est trouvée. Si l'option ",S" est spécifiée, cet affichage sera remplacé par la simple indication du nombre d'occurences de cette expression alphanumérique dans le programme, en outre la variable SK sera mise à jour avec ce nombre. Enfin, si l'option ",M" est ajoutée, SEEK retera complètement Muet: rien ne sera affiché.

Le code ASCII NUL que l'on pourrait insérer avec un CHR$(0) est interdit. Il est possible d'insérer des token BASIC.

Le joker "*" est interdit, mais on peut utiliser "£" à la place du "?" qui pourrait prêter à confusion avec la commande PRINT.

SEEK sans paramètre permet de lister une ligne à la fois, en utilisant l'expression alphanumérique indiquée lors du SEEK précédent (s'il n'y a pas encore eu de SEEK, on a droit à une belle "SYNTAX ERROR".

Variables utilisées

33/34		n° de ligne
CE/CF		pointe sur le lien suivant
F2/F3		pointe sur le début du programme BASIC
F4	LNG	longueur de la chaîne à chercher
F5/F6		nombre d'occurences de la chaîne dans le programme
F7		flag ",S" (à 1 si présent)
F8		flag ",M" (à 1 si présent)
F9		flag "argument" (à 1 si présence d'argument)
02F2		flag "LIST" (b7 à 1 pour retour au point d'appel)
C04C	DEFAFF	code ASCII à placer devant les nombres décimaux
C089/C08A	DEBBAS	(vise le lien de la première ligne)
C08B		longueur de la chaîne à chercher
C0AA/C0F8		zone de 79 octets pour garder la chaîne à chercher

Informations non documentées
Par contre, ce que le manuel ne dit pas très clairement, c'est que la chaîne doit être tokenisée, afin d'avoir la même structure que les commandes d'une ligne BASIC. Sédoric est vraiment génial compte tenu de sa taille et de la modestie du système! Exemple: soit à rechercher tous les CHR$(17). Sachant que cette séquence est codée sous la forme du token de CHR$ qui est l'octet 237 et de la chaîne "(17)", il faut faire:
- soit directement SEEK CHR$(237)+"(17)"
- soit ZZ$=CHR$(237)+"(17)" puis SEEK ZZ$
- soit enfin ZZ$=CHR$(17) puis TKEN ZZ$ et SEEK ZZ$, si vous n'avez pas le manuel de l'Atmos sous la main et que vous ignorez le token de la commande. Attention TKEN marche seulement en mode programme.

Les paramètres ",S" et ",M" peuvent être permutés. La variable SK n'est mise à jour que si le paramètre ",S" est indiqué. Le paramètre ",M" utilisé seul reste sans effet! Il n'a de sens que couplé au paramètre ",S".

La longueur maximale de la chaîne est de 79 caractères et non 78 comme indiqué dans le manuel. Il s'agit en fait non pas de caractères, mais d'octets. Ainsi le token PRINT compte non pas pour 5 caractères, mais pour un seul, l'octet #BA.

Le manuel donne quelques indications concernant les possibilités de recherche de SEEK, mais de manière très insuffisante. SEEK ne se contente pas de rechercher une châine dans le programme, mais en fait une suite d'octets quelconques pourvu que cette suite soit formulée sous forme de chaîne.

Par exemple, dans la ligne 22 PRINT"TOTO?":GETR$ les possibilités de SEEK ne se limitent pas à la chaîne TOTO. On peut rechercher: PRINT"TOTO" avec SEEK

CHR$(186)+CHR$(34)+"TOTO"

O?":	avec	SEEK "O?"+CHR$(34)+CHR$(58)
":GET	avec	SEEK CHR$(34)+CHR$(58)+CHR$(19O)

A noter que SEEK "TOTO" reconnait la chaîne "TOTO" dans la ligne 22 REM TOTO mais pas dans la ligne 22 ' TOTO qui est codée avec le token BASIC de TO alors que SEEK "TITI" reconnait "TITI" aussi bien dans 22 REM TITI que dans 22 ' TITI.

Plus curieux encore, dans la ligne 222 PRINT CHR$(41) utilisable pour afficher "(" on peut rechercher:

CHR$	avec	SEEK CHR$(237)
CHR$(	avec	SEEK CHR$(237)+"("
CHR$(4	avec	SEEK CHR$(237)+"("+CHR$(52)
CHR$(41)	avec	SEEK CHR$(237)+"("+CHR$(52)+"1)"
ou	avec	SEEK CHR$(237)+"(41"+CHR$(41)

Tous ces exemples sont un peu "forcés", mais démontrent les possibilités de SEEK. Pour éviter que de petits malins poussent le vice à inclure le #00 de début de ligne dans les chaînes à chercher et à remplacer, les auteurs de Sédoric ont été obligés d'interdire la présence de #00 dans ces chaînes.

Voici quelques token très utiles lorsqu'on veut adapter des programmes BASIC: 191 pour CALL, 192 pour !, 182 et 183 pour CLOAD et CSAVE, 230 et 231 pour PEEK et DEEK, 185 et 138 pour POKE et DOKE, 157 et 39 pour REM et '.

Notez que les n° de lignes sont codés en ASCII (SEEK"1230" pour rechercher les GOTO 1230, GOSUB 1230 etc...). De même il faut utiliser SEEK"LOAD" et SEEK"SAVE" également codé en ASCII, à la différence de CLOAD et CSAVE.

Enfin, bogue usuelle: l'option "m" en minuscule ne sera pas prise en compte et déclenchera une belle "SYNTAX ERROR"), alors que l'option "s" sera acceptée sans problème!

Utilisation en LM

Bien que d'un intérêt discutable, on peut utiliser SEEK à partir d'un programme en langage machine en faisant un JSR F154 après avoir basculé sur la RAMOV. Il est possible d'initialiser la chaîne ainsi que les flag ",S" et ",M" en écrivant dans le tampon clavier (voir les détails en annexe).

Analyse de la syntaxe et saisie des paramètres

415c	08	PHP	sauvegarde les indicateurs 6502
416c	A9 00	LDA #00	
418c	85 F5	STA F5	force à zéro F5, F6 et DEFAFF
41Ac	85 F6	STA F6	(code ASCII devant les nombres décimaux)
41Cc	8D 4C C0	STA C04C	
41Fc	46 F7	LSR F7	force à zéro le b7 de F7 (flag ",S")
421c	46 F8	LSR F8	force à zéro le b7 de F8 (flag ",M")
423c	38	SEC	
424c	66 F9	ROR F9	force à 1 le b7 de F9 (flag "présence d'argument")

NB: SEEK sans argument permet de lister une ligne à la fois selon la dernière valeur de la chaîne cherchée, qui reste en mémoire tant qu'une autre chaîne n'est pas précisée ou qu'un RESET n'est pas effectué.

426c	38	SEC	
427c	6E F2 02	ROR 02F2	force à 1 b7 de 02F2 (LIST retourne à l'appelant)
42Ac	28	PLP	récupère les indicateurs 6502 dont Z
42Bc	D0 11	BNE C43E	continue en C43E s'il y a des paramètres

Il n'y a pas de paramètre

42Dc	46 F9	LSR F9	pas de paramètre, force à 0 le b7 de F9
42Fc	AD 8B C0	LDA C08B	LNG de châine est mise a jour avec SEEK précédent
432c	85 F4	STA F4	F4 = C08B = longueur de la chaîne à chercher
434c	78	SEI	interdit les interruptions
435c	AC 89 C0	LDY C089	
438c	AE 8A C0	LDX C08A	YX = C089/C08A = DEBBAS
43Bc	4C 86 C4	JMP C486	suite forcée en C486

Il y a des paramètres

43Ec	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
441c	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
444c	85 F4	STA F4	F4 = LNG, longueur de la chaîne à chercher
446c	8D 8B C0	STA C08B	C08B = LNG, longueur de la chaîne à chercher
449c	A8	TAY	teste LNG
44Ac	F0 58	BEQ C4A4	continue en C4A4 (en fait en C500) si nulle
44Cc	C0 50	CPY #50	teste si LNG >= 80 caractères
44Ec	B0 BD	BCS C40D	si oui, continue en C40D ("STRING TOO LONG")
450c	88	DEY	indexe de 0 à LNG - 1 pour lecture de LNG octets

```
C451c  B1 91       LDA (91),Y    lecture caractère de la chaîne à chercher
C453c  F0 BB       BEQ C410      si nul, continue en C410 (INVALID STRING)
C455c  99 AA C0    STA C0AA,Y    si valable, copie ce caractère dans zone C0AA/C0F8
C458c  88          DEY           vise le caractère précédent (opère par la fin)
C459c  10 F6       BPL C451      reboucle en C451 tant qu'il en reste
C45Bc  20 9E D3    JSR D39E      XCRGOT relit le caractère à TXTPTR et le convertit en MAJUSCULE
C45Ec  F0 1C       BEQ C47C      continue en C47C s'il n'y a plus de paramètre
C460c  20 2C D2    JSR D22C      D067/ROM exige une "," lit le caractère suivant et le convertit en MAJUSCULE
C463c  C9 53       CMP #53       est-ce un "S"? (comparaison donne C = 1 si égal)
C465c  D0 09       BNE C470      sinon, poursuit l'analyse de syntaxe en C470
C467c  66 F7       ROR F7        si oui, force à 1 le b7 de F7 (flag ",S")
C469c  20 98 D3    JSR D398      XCRGET lit le caractère suivant à TXTPTR et le convertit en MAJUSCULE
C46Cc  D0 F2       BNE C460      reboucle en C460 s'il y a encore des paramètres
C46Ec  F0 0C       BEQ C47C      sinon, continue en C47C (ordre ,S,M inversable)
C470c  A9 4D       LDA #4D       caractère "M" sera recherché (bogue usuelle: le "m" en minuscule ne sera pas pris en
                                 compte et déclenchera une belle "SYNTAX ERROR")
C472c  20 2E D2    JSR D22E      JSR D067/ROM et D3A1/RAMOV demande "M" à TXTPTR, lit le caractère suivant à
                                 TXTPTR et le convertit en MAJUSCULE
C475c  08          PHP           sauvegarde les indicateurs 6502 dont Z
C476c  38          SEC
C477c  66 F8       ROR F8        force à 1 le b7 de F8 (",M" trouvé)
C479c  28          PLP           récupère les indicateurs 6502 dont Z
C47Ac  D0 E4       BNE C460      reboucle en C460 s'il y a encore des paramètres
C47Cc  A6 9B       LDX 9B
C47Ec  A4 9A       LDY 9A
C480c  8C 89 C0    STY C089      YX = C089/C08A = DEBBAS (vise 1er lien de 1ère ligne)
C483c  8E 8A C0    STX C08A
C486c  98          TYA
C487c  D0 01       BNE C48A
C489c  CA          DEX           F2/F3 = adresse du lien - 1 (vise #00 début de ligne)
C48Ac  88          DEY
C48Bc  84 F2       STY F2
C48Dc  86 F3       STX F3
C48Fc  24 F7       BIT F7        teste si le b7 de F7 est à 0 (",S" non validé)
C491c  10 01       BPL C494      si oui, saute l'instruction suivante
C493c  78          SEI           interdit les interruptions
C494c  A6 F3       LDX F3
C496c  A4 F2       LDY F2
C498c  C8          INY
C499c  D0 01       BNE C49C      YX = CE/CF = pointe sur le lien suivant
C49Bc  E8          INX
C49Cc  84 CE       STY CE
C49Ec  86 CF       STX CF
C4A0c  A0 02       LDY #02       pour lire 2 places après le #00 du début de ligne
C4A2c  B1 F2       LDA (F2),Y    lit HH de lien
C4A4c  F0 5A       BEQ C500      si nul, fin du programme BASIC, continue en C500
C4A6c  C8          INY
C4A7c  B1 F2       LDA (F2),Y
C4A9c  85 33       STA 33        copie le n° de ligne en 33/34 pour LIST
C4ABc  C8          INY
C4ACc  B1 F2       LDA (F2),Y
C4AEc  85 34       STA 34
C4B0c  A9 05       LDA #05       il y a 5 octets d'en-tête au debut de chaque ligne de programme BASIC: #00,
                                 deux octets de lien et deux octets de N° de ligne
C4B2c  18          CLC           prépare une addition
C4B3c  65 F2       ADC F2
C4B5c  85 F2       STA F2        F2/F3 = F2/F3 + #05 ou + LNG qui a été trouvée
C4B7c  90 02       BCC C4BB      F2/F3 pointe sur la 1ère instruction de la ligne
C4B9c  E6 F3       INC F3
C4BBc  A0 00       LDY #00       Y est remis à zéro à chaque nouvelle recherche
C4BDc  B1 F2       LDA (F2),Y    lit un octet de programme dans la ligne BASIC
C4BFc  D0 04       BNE C4C5      sinon nul, saute les deux instructions suivantes
C4C1c  C0 00       CPY #00       si fin de ligne atteinte, teste pointeur de chaîne
C4C3c  F0 CF       BEQ C494      si pas d'octets identiques, reboucle en C494
C4C5c  BE AA C0    LDX C0AA,Y    lit dans X un octet de la chaîne à chercher
C4C8c  E0 5F       CPX #5F       est-ce un "£"? (jocker pour SEEK)
C4CAc  F0 0D       BEQ C4D9      si oui, continue en C4D9
C4CCc  D9 AA C0    CMP C0AA,Y    sinon, compare cet octet avec octet de chaîne
C4CFc  F0 08       BEQ C4D9      continue en C4D9 si identique
C4D1c  E6 F2       INC F2        si différent, incrémente F2/F3
C4D3c  D0 E6       BNE C4BB      (pointeur dans programme BASIC)
```

```
4D5c  E6 F3       INC F3
4D7c  D0 E2       BNE C4BB      et reboucle en C4BB
4D9c  C8          INY           si identiques, incrémente Y = nombre caractères identiques
4DAc  C4 F4       CPY F4        la fin de la chaîne à chercher est-elle atteinte?
4DCc  D0 DF       BNE C4BD      sinon, reboucle en C4BD sans remettre Y à zéro
4DEc  E6 F5       INC F5
4E0c  D0 02       BNE C4E4      si oui, incrémente F5/F6

4E2c  E6 F6       INC F6        (nombre d'occurences de la chaîne dans le programme)
4E4c  A5 F4       LDA F4        chaîne complète de longueur A trouvée
4E6c  24 F7       BIT F7        teste si b7 de F7 est à 1 (paramètre ",S")
4E8c  30 C8       BMI C4B2      si oui (ne pas afficher lignes), reboucle en C4B2
4EAc  20 28 D6    JSR D628      sinon, affiche un espace, puis
4EDc  20 B4 D1    JSR D1B4      C76C/ROM exécute la commande "LIST" simplifiée
4F0c  A4 CE       LDY CE
4F2c  A6 CF       LDX CF        YX = adresse du lien suivant
4F4c  24 F9       BIT F9        teste si b7 de F9 est à 1 ("arguments présents")
4F6c  30 8E       BMI C486      si oui, reboucle en C486 (reprend début ligne suivante)
4F8c  8C 89 C0    STY C089      sinon ("pas d'argument"),
4FBc  8E 8A C0    STX C08A      sauve adresse du lien suivant en C089/C08A (pour futur SEEK)
4FEc  58          CLI           autorise les interruptions
4FFc  60          RTS           et sort de la commande SEEK
```

Fin du programme BASIC atteinte

```
500c  24 F7       BIT F7        teste si b7 de F7 est à 0 (pas de paramètre ",S")
502c  10 FA       BPL C4FE      si oui (afficher les lignes), CLI et RTS en C4FE
504c  A5 F5       LDA F5        sinon,
506c  A4 F6       LDY F6        AY = nombre d'occurances trouvées
508c  20 F5 D7    JSR D7F5      mise à jour de la variable SK (HH=Y et LL=A)
50Bc  24 F8       BIT F8        teste si b7 de F8 est à 1 (paramètre ",M")
50Dc  30 EF       BMI C4FE      si oui (ne rien afficher), CLI et RTS en C4FE
50Fc  20 06 D2    JSR D206      CBF0/ROM va à la ligne
512c  A5 F5       LDA F5
514c  A4 F6       LDY F6        AY = nombre d'occurances trouvées
516c  20 53 D7    JSR D753      affichage en décimal sur 5 digits d'un nombre AY
519c  58          CLI           autorise les interruptions
51Ac  A2 00       LDX #00       indexe le message " Founds" (bogue: avec une faute!)
51Cc  4C 64 D3    JMP D364      XAFSC affiche X+1ème message externe terminé par "caractère + 128"

51Fc  4C 10 C4    JMP C410      "INVALID STRING ERROR"

522c  4C 77 E9    JMP E977      "STRING TOO LONG ERROR"
```

EXECUTION COMMANDE SEDORIC CHANGE

Rappel de la syntaxe

CHANGE expression_alphanumérique_n°1 TO expression_alphanumérique_n°2)

Remplace une suite d'octets exprimée sous la forme de l'expression alphanumérique n°1 par une autre suite d'octet indiquée sous forme de l'expression alphanumérique n°2. Toutes les occurances de l'expression alphanumérique n°1 seront remplacées dans tout le programme BASIC. Comme avec SEEK, on peut introduire des token BASIC par exemple:

CHANGE CHR$(157) TO CHR$(39) remplace tous les REM par des ""

Le code ASCII nul (#00) est proscrit dans les chaînes, comme c'était le cas avec SEEK. La longueur des chaînes est limitée à 78 octets (et non 78 caractères comme indiqué dans le manuel pages 48 et 49). Si le caractère joker "£" est présent à la même place dans les chaînes n°1 et 2, ce caractère ne sera pas modifié. Par contre s'il n'est present que dans la chaîne n°1, il sera remplacé par le caractère indiqué à la même place dans la chaîne n°2.

Variables utilisées

```
0                 flag début de ligne BASIC
C/9D    FINBAS    fin du programme BASIC
7/C8    FINCIBL   dernier octet de l'emplacement qui recevra le bloc
9/CA    FINBLOC   dernier octet du bloc qui sera déplacé vers le haut
E/CF    DEBBLOC   premier octet du bloc qui sera déplacé vers le haut
2/F3    SOURCE    nouveau début du bloc haut
4/F5    CIBLE     point de redescente du bloc
6
7       LNG2      longueur  de la chaîne de remplacement
8       LNG1      longueur  de la chaîne cherchée
9                 différence de longueur LNG2 - LNG1 des 2 chaînes
100/C1FF  BUF1    où sera stockée la chaîne n°1
```

C200/C2FF BUF2 où sera stockée la chaîne n°2

Informations non documentées
Comme avec COPY, le TO de CHANGE doit obligatoirement être en majuscules. CHANGE"TOTO"TO"BOBO" marche, ainsi que change"TOTO"TO"BOBO", mais CHANGE"TOTO"to"BOBO" déclenchera une "SYNTAX ERROR".

La 2ème chaîne peut être vide, mais pas la 1ère: CHANGE CHR$(157) TO "" élimine tous les REM. Ainsi la ligne 30 REM devenue vide sera élimminée. Il en sera de même pour toutes les lignes vides.

Au cours du remplacement de la chaîne n°1 par la chaîne n°2, la longueur de la ligne BASIC peut être affectée. Si elle devient nulle, la ligne sera supprimée. Si elle dépasse 112 octets, une "STRING TOO LONG ERROR" sera déclanchée. Si l'ensemble du programme BASIC devient trop long une "OUT OF MEMORY ERROR" mettra fin aux remplacements. Dans les 2 cas, la viabilité du programme sera préservée: les liens de lignes sont correctement mis à jour, les remplacements de chaînes n'ont simplement pas été effectués jusqu'au bout.

Il est intéressant de noter, qu'il est possible d'entrer 78 caractères (n° de ligne inclu) dans une ligne BASIC, au clavier, mais que la ligne générée peut en contenir beaucoup plus. Par exemple, l'utilisation de "?" au lieu de PRINT permet d'aller sans problème jusqu'à 235 caractères, que l'on peut admirer avec la commande LIST. Mais en fait, la ligne BASIC réelle est beaucoup moins longue car codée avec des token BASIC: PRINT occupe 1 seul octet et non 5. Afin de ne pas avoir trop de problèmes avec la commande CHANGE, la longueur de ligne a été portée à 112 octets au lieu de 78, et cela semble fonctionner correctement.

Voir les "Informations non documentées" de la commande SEEK en ce qui concerne quelques particularités de reconnaissance des chaînes. En voici quelques autres: Pour changer tous les "CHR$(17)" en "POKE#26A,(PEEK(#26A)AND#FE)" (effacement du curseur) faire ZZ$=CHR$(17) puis TKEN ZZ$ et CHANGE ZZ$ TO POKE#26A,(PEEK(#26A)AND#FE). Rappel pour rétablir le curseur, faire un POKE#26A,(PEEK(#26A)OR#01 ces POKEs ne prennent effet qu'au prochain PRINT ou PRINT@.

Autre exemple, pour adapter la valeur de l'argument d'un WAIT (token 181), faire:
- soit directement CHANGE CHR$(181)+" 50" TO "WAIT100"
- soit ZZ$="CHR$(181)+" 50" puis change ZZ$ TO "WAIT100"
- soit enfin ZZ$="WAIT 50" puis TKEN ZZ$ et CHANGE ZZ$ TO "WAIT100", si vous n'avez pas le manuel de l'Atmos sous la main et que vous ignorz le token de la commande. Attention TKEN marche seulement en mode programme. Dans les 3 cas, il n'est pas nécessaire de coder la 2ème partie (WAIT100): la chaîne de remplacement n'a pas besoin d'être tokenisée.

Utilisation en LM
Voilà une perspective des plus farfelues: modifier un programme BASIC à l'aide de la commande CHANGE à partir d'un programme en LM! Bon, mais nous avons trop de sympatie pour les bidouilleurs pour les laisser sur leur faim. Si la banque n°3 est déjà en place, il suffira de basculer sur la RAMOV, d'initialiser BUF1, BUF2, F7 et F8, puis de faire un JSRC567. Sinon, il faut écrire les paramètres dans le tampon clavier, initialiser TXTPTR puis faire le JSR F148 (voir les détails en annexe).

Analyse de la syntaxe et saisie des paramètres

C525c	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
C528c	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
C52Bc	A8	TAY	
C52Cc	84 F8	STY F8	F8 = longueur LNG1 de la chaîne cherchée (n°1)
C52Ec	F0 0F	BEQ C53F	si nulle, continue en C53F (un BEQ C4FF aurait été beaucoup plus rapide pour arriver au même point en cas de chaîne vide)
C530c	88	DEY	indexe chaîne n°1 (ira de #00 à longueur - 1)
C531c	C0 4E	CPY #4E	LNG1 est-elle >= 78? (bogue: #4F aurait été mieux!)
C533c	B0 ED	BCS C522	si oui, "STRING TOO LONG ERROR"
C535c	B1 91	LDA (91),Y	sinon, lit octet de la chaîne n°1
C537c	99 00 C1	STA C100,Y	et le copie dans BUF1 à la position Y
C53Ac	F0 E3	BEQ C51F	si octet est nul, termine en C51F "INVALID STRING ERROR"
C53Cc	88	DEY	si octet OK, vise le précédent (opère par la fin)
C53Dc	10 F6	BPL C535	reboucle en C535 tant qu'il en reste à copier
C53Fc	A9 C3	LDA #C3	token "TO" (bogue usuelle: le "to" en minuscules ne sera pas pris en compte et déclenchera une belle "SYNTAX ERROR")
C541c	20 2E D2	JSR D22E	JSR D067/ROM et D3A1/RAMOV demande "TO" à TXTPTR, lit le caractère suivant à TXTPTR et le convertit en MAJUSCULE
C544c	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
C547c	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

```
54Ac  A8          TAY
54Bc  84 F7       STY F7        F7 = longueur LNG2 de la nouvelle chaîne (n°2)
54Dc  F0 14       BEQ C563      si nulle, continue en C563 (suite analyse syntaxe)
54Fc  88          DEY           indexe chaîne n°2 (ira de #00 à longueur - 1)
550c  C0 4E       CPY #4E       LNG2 est-elle >= 78? (re-bogue: #4F aurait été mieux!)
552c  B0 CE       BCS C522      si oui, "STRING TOO LONG ERROR"
554c  B1 91       LDA (91),Y    sinon, lit octet de la chaîne
556c  F0 C7       BEQ C51F      si octet est nul, termine en C51F "INVALID STRING ERROR"
558c  99 00 C2    STA C200,Y    si octet OK, le copie dans BUF2 à la position Y
55Bc  88          DEY           vise l'octet précédent (opère par la fin)
55Cc  10 F6       BPL C554      reboucle en C554 tant qu'il en reste à copier
55Ec  A9 00       LDA #00       force à zéro DEFAFF
560c  8D 4C C0    STA C04C      (code ASCII affiché devant les nombres décimaux)
563c  A5 F8       LDA F8        teste LNG2, longueur de la chaîne cherchée (n°1)
565c  F0 98       BEQ C4FF      simple RTS en C4FF si nulle, sinon...
```

Entrée proprement dite de la routine CHANGE
Décale le programme vers le haut sous les chaînes

```
567c  38          SEC
568c  A5 F7       LDA F7        F9 = F7 - F8 (LNG2 - LNG1)
56Ac  E5 F8       SBC F8
56Cc  85 F9       STA F9
56Ec  A5 9C       LDA 9C
570c  A4 9D       LDY 9D        C9/CA (FINBLOC) = FINBAS
572c  85 C9       STA C9        (dernier octet du bloc à déplacer vers le haut)
574c  84 CA       STY CA
576c  A4 9B       LDY 9B
578c  A6 9A       LDX 9A
57Ac  D0 01       BNE C57D
57Cc  88          DEY
57Dc  CA          DEX           F4/F5 (CIBLE) = DEBBAS - 1
57Ec  86 F4       STX F4        (point de retour ultérieur pour le bloc
580c  84 F5       STY F5        vise le #00 placé devant la 1ère ligne)
582c  86 CE       STX CE        CE/CF (DEBBLOC) = DEBBAS - 1
584c  84 CF       STY CF        (1er octet du bloc à transférer vers le haut)
586c  A4 A3       LDY A3
588c  A6 A2       LDX A2
58Ac  D0 01       BNE C58D
58Cc  88          DEY
58Dc  CA          DEX           AY = C7/C8 (FINCIBL) = A2/A3 - 1
58Ec  8A          TXA           (sous les chaînes BASIC)
58Fc  85 C7       STA C7
591c  84 C8       STY C8
593c  20 5C D1    JSR D15C      C3F4/ROM décale un bloc memoire vers le haut. En CE/CF adresse du 1er octet du bas, en C9/CA adresse du dernier octet du haut, en C7/C8 et AY adresse de la zone cible vers haut, revient avec nouveau début-#100 en C7/C8 et nouvelle fin en A0/A1 (haut tableaux).
596c  78          SEI           interdit les interruptions pendant le MOVE
597c  A4 C8       LDY C8
599c  A6 C7       LDX C7
59Bc  C8          INY           F2/F3 (SOURCE) = C7/C8 + #100 (pointeur dans le
59Cc  86 F2       STX F2        bloc haut, ajusté sur le premier #00)
59Ec  84 F3       STY F3
```

Organigramme

L'organigramme de la routine CHANGE est particulièrement complexe et contient des fossiles qui n'ont pas été éliminés après mise au point:

- en C5A0 mise à 0 du flag "ligne vide" et du nombre de caractères identiques

- en C5A4 point de rebouclage: on est sur le #00 de nouvelle ligne, on teste s'il y a une chaîne en cours d'exploration (fossile de mise au point)

- en C5A7 on teste si le flag "ligne vide" est à 1, si oui, on recule de 5 pas

- en C5B6 on teste si fin programme BASIC atteinte. Si oui, on termine en C632

- en C5BC on ajuste F6 à #70 qui est une longueur de ligne batarde, sorte de "garde-fou" permettant d'allonger un peut la ligne classique du BASIC avec CHANGE" sans déclencher de "SYNTAX ERROR" à tout bout de champ

- en C5CA les 5 octets d'en-tête sont descendus avec mise à jour des pointeurs SOURCE et CIBLE et le flag "ligne vide" est mis à 1 (début de ligne)

- en C5D0 début recherche chaîne, met Y à 0 (nombre de caractères identiques)

- en C5D2 descend un octet qui sera surchargé si besoin par la nouvelle chaîne. On teste cet octet: s'il est nul (nouvelle ligne) reboucle en C5A4 s'il est différent de l'octet corresp de la chaîne cherchée, met flag "ligne vide" à 0, décrémente le garde- fou F6, que l'on teste (éventuel STRING TOO LONG), si OK reprend la recherche en C5D0

- en C5EF octet lu est identique à l'octet corresp de la chaîne cherchée incrémente Y (nombre caractères identiques), teste si Y = LNG1 (fini, tous les caractères sont trouvés), sinon reprend l'examen en C5D2

- en C5FF teste s'il y a assez de place en mémoire pour opérer le changement sinon, termine la redescente du programme par simple recopie, restaure les liens de ligne et "OUT OF MEMORY ERROR"
- en C608 il y a assez de place en RAM, met flag "ligne vide" à 0, copie la nouvelle chaîne dans le bloc du bas, octet par octet en décrémentant le garde-fou F6, (test à chaqu fois avec éventuel "STRING TOO LONG ERROR"),

- en C61A la chaîne n°2 étant en place, mise à jour des pointeurs SOURCE et CIBLE avec LNG1 et LNG2 et reprend au début en C5A4

Ajuste les flags pour nouvelle ligne et chaîne en cours nulle

C5A0c 46 00 LSR 00 force à zéro le b7 de 00 qui sert de flag pour détecter la présence d'une ligne vide. Ce b7 est toujours à zéro sauf en début de ligne, après descente des 5 octets d'en-tête où il passe à 1. Si le programme rencontre au moins un caractère différent de #00 (c'est à dire si la ligne n'est pas vide), ce flag est remis à 0. Sinon, il reste à 1 et lors de la détection de la nouvelle ligne, le pointeur CIBLE est reculé de 5 pas, ce qui revient à eliminer la ligne vide.

C5A2c A0 00 LDY #00 force Y à zéro (contiendra le nombre de caractères trouvés identiques dans la chaîne cherchée et dans la chaîne explorée)

On est au début d'une ligne de programme BASIC

C5A4c 98 TYA teste la valeur de Y

C5A5c D0 3D BNE C5E4 continue en C5E4 si Y n'est pas nul. Il s'agit d'un résidu de mise au point: à l'origine CHANGE devait probablement pouvoir manipuler toute chaîne d'octets, y compris les 5 octets d'en-tête de ligne. Mais devant les difficultés rencontrées cette possibilité a été éliminée, ainsi que l'atteste la détection du 00 dans l'analyse de syntaxe initiale. Les 2 lignes C5A4 et C5A5 sont à la limite de la bogue.

C5A7c 24 00 BIT 00 teste si le b7 de 00 est nul (début ligne normal)

C5A9c 10 0B BPL C5B6 si oui, continue en C5B6

C5ABc A5 F4 LDA F4 sinon, il faut remettre le pointeur CIBLE sur le

C5ADc 38 SEC #00 de début de ligne (ligne précédente est vide)

C5AEc E9 05 SBC #05 F4/F5 = F4/F5 - #05 (#00 de début + 2 octets

C5B0c 85 F4 STA F4 de lien + 2 octets de numéro de ligne)

C5B2c B0 02 BCS C5B6

C5B4c C6 F5 DEC F5

Teste si la fin du programme BASIC est atteinte

C5B6c A0 02 LDY #02 indexe HH du lien situé 2 pas après #00 de début

C5B8c B1 F2 LDA (F2),Y lit HH du lien

C5BAc F0 76 BEQ C632 si nul (FINBAS), continue en C632 (fin programme)

Initialise la recherche pour une nouvelle ligne

C5BCc A2 70 LDX #70 F6 sert de "garde-fou" pour limiter la longueur de la nouvelle

C5BEc 86 F6 STX F6 ligne a 112 caractères au lieu de 79. Cette marge supplémentaire est justifiée par les allongements possibles obtenus avec CHANGE, afin de réduire les SYNTAX ERRORs, dans la mesure où l'interpréteur BASIC ne bloque pas. F6 sera décrémenté à chaque fois qu'un nouvel octet est ajouté à la ligne en cours dans le bloc du bas.

C5C0c C8 INY

C5C1c B1 F2 LDA (F2),Y

C5C3c C8 INY 33/34 = numéro de la ligne en cours d'analyse

C5C4c 85 33 STA 33 (pour affichage éventuel)

C5C6c B1 F2 LDA (F2),Y

C5C8c 85 34 STA 34

C5CAc 20 7D C6 JSR C67D descend les 5 octets d'en-tête de SOURCE vers CIBLE

C5CDc 38 SEC

C5CEc 66 00 ROR 00 force à 1 le b7 de 00 (ligne est actuellement vide)

Début de recherche de la chaîne n°1 dans la ligne en cours

C5D0c A0 00 LDY #00 réinitialise nombre caractères trouvés identiques

Descend et teste un octet

C5D2c B1 F2 LDA (F2),Y lit octet de ligne BASIC dans le bloc du haut

C5D4c 91 F4 STA (F4),Y et l'écrit dans le bloc du bas

C5D6c F0 CC BEQ C5A4 si c'est un #00 de nouvelle ligne, reboucle en C5A4

C5D8c BE 00 C1 LDX C100,Y sinon lit dans X octet de la chaîne 1 (ancienne)

```
5DBc  E0 5F       CPX #5F      est-ce un "£"? (joker pour CHANGE)
5DDc  F0 10       BEQ C5EF     si oui, continue directement en C5EF
5DFc  D9 00 C1    CMP C100,Y   sinon, l'octet lu dans le bloc du haut est-il égal à l'octet lu dans la chaîne 1 (ancienne)?
5E2c  F0 0B       BEQ C5EF     si oui, continue en C5EF (trouvé!)

      L'octet descendu est différent de l'octet corresp du modèle
5E4c  46 00       LSR 00       sinon, force à zéro le b7 de 00 (la ligne de programme BASIC en cours contient au moins 1
      octet)
5E6c  C6 F6       DEC F6       et décrémente F6
                                (nombre d'octets pouvant encore être ajoutés à la nouvelle ligne, avant que ça fasse trop
      long)
5E8c  F0 4E       BEQ C638     continue en C638 lorsque F6 atteint zéro
5EAc  20 5A C6    JSR C65A     sinon, incrémente F2/F3 (SOURCE) et F4/F5 (CIBLE)
5EDc  D0 E1       BNE C5D0     reprise forcée en C5D0

      L'octet descendu est identique à l'octet corresp du modèle
      Teste s'il y a assez de place pour écrire la nouvelle chaîne
5EFc  C8          INY          trouvé! incrémente le nombre d'octets identiques
5F0c  C4 F8       CPY F8       Y a t-il atteint la longueur de la chaîne 1
5F2c  D0 DE       BNE C5D2     sinon, reboucle en C5D2
5F4c  A5 F4       LDA F4
5F6c  18          CLC
5F7c  65 F9       ADC F9       si oui, calcule AX = F4/F5 + F9
5F9c  A6 F5       LDX F5       (F9 = différence LNG2 - LNG1)
5FBc  90 01       BCC C5FE
5FDc  E8          INX

                                Teste si place en RAM pour écrire nouvelle chaîne à la place de l'ancienne
5FEc  E4 F3       CPX F3       teste si X (HH bloc bas) < F3 (HH bloc haut)
600c  90 06       BCC C608     si oui (OK), continue en C608
602c  D0 50       BNE C654     si X > F3, continue en C654 (en fait C667)
604c  C5 F2       CMP F2       si X = F3, teste si A >= F2 (LL blocs bas et haut)
606c  B0 4C       BCS C654     si oui, continue en C654 (en fait C667)

      Copie la nouvelle chaîne à la place de l'ancienne dans le bloc du bas
608c  A4 F7       LDY F7       OK, Y = LNG2, longueur de la nouvelle chaîne
60Ac  F0 19       BEQ C625     continue en C625 si cette longueur est nulle
60Cc  46 00       LSR 00       sinon, force à zéro le b7 de 00 (au moins 1 octet)
60Ec  B9 00 C2    LDA C200,Y   lit un octet de la nouvelle chaîne danss BUF2, à la position Y
611c  91 F4       STA (F4),Y   écrit cet octet dans le bloc du bas
613c  C6 F6       DEC F6       et décrémente F6 (nombre octets encore ajoutables)
615c  F0 21       BEQ C638     continue en C638 lorsque F6 atteint 0 (trop longue)
617c  88          DEY          vise l'octet précédent (opère par la fin)
618c  10 F4       BPL C60E     et reboucle en C60E tant qu'il en reste à copier

      Met SOURCE et CIBLE à jour selon LNG1 et LNG2
61Ac  A5 F7       LDA F7       fini,
61Cc  18          CLC
61Dc  65 F4       ADC F4       mise à jour de CIBLE = F4/F5 + F7
61Fc  85 F4       STA F4       (F7 = LNG2, longueur de la chaîne 2)
621c  90 02       BCC C625
623c  E6 F5       INC F5
625c  A5 F8       LDA F8
627c  18          CLC
628c  65 F2       ADC F2       mise à jour de SOURCE = F2/F3 + F8
62Ac  85 F2       STA F2       (F8 = LNG1, longueur de la chaîne 1)
62Cc  90 A2       BCC C5D0
62Ec  E6 F3       INC F3
630c  D0 9E       BNE C5D0     reprise forcée en C5D0 car HH de SOURCE jamais nul

      Fin du programme BASIC atteinte
632c  91 F4       STA (F4),Y   écrit A = #00 dans HH du lien de fin de programme
634c  58          CLI          autorise les interruptions
635c  4C B4 E0    JMP E0B4     restaure les liens de lignes et les pointeurs
                                puis quitte CHANGE si appelé de C5BA ou retourne si appelé de C673

      Lorsque F6 atteint zéro: nouvelle ligne trop longue
638c  A0 00       LDY #00      force Y à zéro
63Ac  20 60 C6    JSR C660     incrémente SOURCE
63Dc  B1 F2       LDA (F2),Y   lit octet de ligne BASIC dans le bloc du haut
63Fc  D0 F7       BNE C638     reprend en C638 si pas nul (cherche fin de ligne)
```

```
C641c  20 67 C6   JSR C667    si fin de ligne trouvée,
C644c  A2 02      LDX #02     indexe le message "LINE"
C646c  20 64 D3   JSR D364    XAFSC affiche X+1ème message externe terminé par "caractère + 128"
C649c  A5 33      LDA 33
C64Bc  A4 34      LDY 34      AY = numéro de ligne
C64Dc  20 53 D7   JSR D753    affichage en décimal sur 5 digits d'un nombre AY
C650c  58         CLI         autorise les interruptions
C651c  4C 0D C4   JMP C40D    "STRING TOO LONG ERROR"

       Il n'y a pas assez de place pour écrire la nouvelle chaîne
C654c  20 67 C6   JSR C667    termine la descente sans mise à jour des chaînes
C657c  4C 6C D1   JMP D16C    "OUT OF MEMORY ERROR"

       Incrémente F2/F3 (SOURCE) et F4/F5 (CIBLE)
C65Ac  E6 F4      INC F4
C65Cc  D0 02      BNE C660    Incrémente CIBLE
C65Ec  E6 F5      INC F5
C660c  E6 F2      INC F2
C662c  D0 02      BNE C666    Incrémente SOURCE
C664c  E6 F3      INC F3
C666c  60         RTS

       Termine la descente sans mise à jour des chaînes
C667c  A0 00      LDY #00
C669c  B1 F2      LDA (F2),Y  lit octet de ligne BASIC dans le bloc du haut
C66Bc  91 F4      STA (F4),Y  écrit cet octet dans le bloc du bas
C66Dc  D0 09      BNE C678    continue en C678 si pas nul (nouvelle ligne)
C66Fc  A0 02      LDY #02     si nouvelle ligne,
C671c  B1 F2      LDA (F2),Y  lit HH du lien suivant
C673c  F0 BD      BEQ C632    si nul (FINBAS), continue en C632
                              (restaure les liens de lignes et retourne à la dernière adresse empilée c'est à dire en C657)
C675c  20 7D C6   JSR C67D    sinon, copie 5 octets d'en-tête de SOURCE vers CIBLE
C678c  20 5A C6   JSR C65A    incrémente SOURCE et CIBLE
C67Bc  D0 EA      BNE C667    reprise forcée en C667 car HH de CIBLE jamais nul

       Copie 5 octets d'en-tête de SOURCE vers CIBLE
C67Dc  A2 04      LDX #04     pour copier 5 octets (X = 4 à 0)
C67Fc  A0 00      LDY #00     index nul
C681c  20 5A C6   JSR C65A    incrémente SOURCE et CIBLE
C684c  B1 F2      LDA (F2),Y
C686c  91 F4      STA (F4),Y  copie 5 octets du bloc haut vers le bloc bas
C688c  CA         DEX
C689c  10 F6      BPL C681    reboucle en C681 tant qu'il en reste
C68Bc  60         RTS
```

EXECUTION COMMANDE SEDORIC MERGE

Rappel de la syntaxe

MERGE FICHCOMPL (,L)

Voilà une commande très utile, bien plus performante que COPYM CLOAD,J et LOAD,J En effet, elle permet de mélanger les lignes du programme BASIC présent en RAM avec celles d'un programme FICHCOMPL, chargé à partir d'une disquette, pour former un nouveau programme en RAM. FICHCOMPL (fichier complémentaire) représente un nom de fichier non ambigu et l'option ",L" est utilisée pour inhiber le listing qui s'affiche normalement au cours de l'opération.

Les variables sont conservées, mais pas les fonctions définies par DEF FN. MERGE peut être utilisé aussi bien en mode direct qu'en mode programme, mais dans ce dernier cas, il faudra veiller à ce qu'aucune ligne ne soit inserée anvant la ligne où se trouve MERGE.

Variables utilisées

```
33/34                 n° de la ligne en cours
9A/9B      DEBBAS     début du programme BASIC
9C/9D      DEBVAR     début des variables BASIC
9E/9F      DEBTAB     début des tableaux BASIC
A0/A1      FINTAB     fin des tableaux BASIC
C7/C8                 adresse de la cible vers le haut, puis adresse du nouveau début - #100
C9/CA                 adresse de la fin du bloc à déplacer vers le haut
CE/CF                 adresse du début du bloc à déplacer vers le haut
           PTRBAS     pointeur dans le bloc du bas (programme BASIC déjà en RAM)
F6                    nombre d'octets d'instructions propr dites à copier (longueur de la ligne)
F8/F9      PTRHAUT    pointeur dans le bloc du haut (programme complémentaire)
026A                  indicateurs du status console
```

016			flag "banque changée" (b7 à 1 si changée)
027		POSNMX	position du nom de fichier dans le secteur de catalogue
04E		VSALO1	code pour SAve/LOad (b6 = 1 si ",A" et b7 = 1 si ",J")
04C		DEFAFF	code ASCII à mettre devant les nombres décimaux
074			flag "listing" (b7 à 1 si OFF)
052/C053		DESALO	adresse de début du fichier (adresse de chargement)
100/C1FF		BUF1	
300/C3FF		BUF3	

Informations non documentées

Bogue usuelle: le "l" en minuscule ne sera pas pris en compte et déclenchera une belle "SYNTAX ERROR").

Utilisation en LM

Il faut écrire les paramètres dans le tampon clavier, initialiser TXTPTR puis faire un JSR F13C (voir les détails en annexe).

Traitement des erreurs

(L'entrée proprement dite est en C695)

68Cc	4C DD E0	JMP E0DD	"FILE NOT FOUND ERROR"
68Fc	4C E0 E0	JMP E0E0	"FILE TYPE MISMATCH ERROR"

Fini

692c	68	PLA	élimine 2 octets de la pile et retourne
693c	68	PLA	
694c	60	RTS	

Analyse de la syntaxe et saisie des paramètres

695c	20 4F D4	JSR D44F	XNF lit nom de fichier non-ambigu à TXTPTR et l'écrit dans BUFNOM
698c	08	PHP	sauvegarde les indicateurs 6502 dont Z
699c	48	PHA	sauve A qui contient le caractère suivant
69Ac	2C 16 C0	BIT C016	teste si b7 du flag "banque changée" est à 0
69Dc	10 0A	BPL C6A9	si oui (banque pas changée), continue en C6A9
69Fc	A2 03	LDX #03	indexe le message "LOAD"
6A1c	20 64 D3	JSR D364	XAFSC affiche X+1ème message externe terminé par "caractère + 128"
6A4c	20 48 D6	JSR D648	affiche "DISC IN DRIVE "lettre du lecteur" AND PRESS RETURN" puis demande un "ESC" (C = 1) ou un "RETURN" (C = 0)
6A7c	B0 E9	BCS C692	si "ESC", dépile 2 octets et termine en C692
6A9c	68	PLA	récupère A
6AAc	28	PLP	récupère les indicateurs 6502 dont Z
6ABc	18	CLC	pour flag C074 "listing" ON par défaut
6ACc	F0 09	BEQ C6B7	continue en C6B7 s'il n'y a plus de paramètres
6AEc	20 2C D2	JSR D22C	D067/ROM exige une "," lit le caractère suivant et le convertit en MAJUSCULE
6B1c	A9 4C	LDA #4C	caractère "L" (pour inhiber le listing)
6B3c	20 2E D2	JSR D22E	JSR D067/ROM et D3A1/RAMOV demande "L" à TXTPTR, lit le caractère suivant à TXTPTR et le convertit en MAJUSCULE
6B6c	38	SEC	pour flag C074 "listing" OFF (présence de ",L")
6B7c	6E 74 C0	ROR C074	force le b7 de C074 selon C flag "listing"

Cherche FICHCOMPL dans le catalogue

6BAc	20 2D DB	JSR DB2D	vérifie que la disquette en place est bien une disquette Sédoric, 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
6BDc	F0 CD	BEQ C68C	continue en C68C si pas trouvé (FILE NOT FOUND)

Charge dand BUF1 le descripteur principal de FICHCOMPL

6BFc	BD 0C C3	LDA C30C,X	lit dans BUF3 (secteur de catalogue) les
6C2c	BC 0D C3	LDY C30D,X	coordonnées AY du premier descripteur du fichier
6C5c	20 5D DA	JSR DA5D	XPBUF1 Charge dans BUF1 le secteur Y de la piste A
6C8c	2C 03 C1	BIT C103	teste si le le b7 du 4ème octet de BUF1 est nul (flag "type de fichier" indique qu'il s'agit d'un fichier non BASIC)
6CBc	10 C2	BPL C68F	si oui, continue en C68F (FILE TYPE MISMATCH)

Vérifie qu'il y a assez de place pour MERGEr

6CDc	AD 06 C1	LDA C106	si fichier BASIC, calcule:
6D0c	ED 04 C1	SBC C104	AY = adresse fin - adresse début = longueur de FICHCOMPL
6D3c	48	PHA	(A reste sur la pile)
6D4c	AD 07 C1	LDA C107	
6D7c	ED 05 C1	SBC C105	
6DAc	A8	TAY	
6DBc	18	CLC	pour addition
6DCc	68	PLA	puis calcule:

```
C6DDc  65 A0      ADC A0      AY =   adresse fin actuelle (fin des tableaux)
C6DFc  48         PHA         + longueur du programme à merger
C6E0c  98         TYA         + #100 de sécurité
C6E1c  65 A1      ADC A1
C6E3c  A8         TAY
C6E4c  C8         INY
C6E5c  68         PLA
C6E6c  20 64 D1   JSR D164    C444/ROM vérifie que l'adresse AY est en dessous des chaînes,
                              "OUT OF MEMORY" si AY trop haut, zone C7/CF n'est pas affectée
```

Charge FICHCOMPL une "page" au-dessus des tableaux

```
C6E9c  A5 A0      LDA A0
C6EBc  A4 A1      LDY A1
C6EDc  C8         INY         C052/C053 (DESALO) =
C6EEc  8D 52 C0   STA C052    A0/A1 (fin des tableaux) + #100 de sécurité
C6F1c  8C 53 C0   STY C053    (adresse où sera chargé FICHCOMPL)
C6F4c  20 E6 DF   JSR DFE6    XDEFLO force les valeurs par défaut pour XLOADA
                              (remet à zéro VSALO0, VSALO1 et LGSALO)
C6F7c  A2 40      LDX #40     0100 0000, b6 à 1 pour option ",A" placé dans
C6F9c  8E 4E C0   STX C04E    VSALO1 pour charger le fichier à l'adresse DESALO
C6FCc  AE 27 C0   LDX C027    reprend X = POSNMX
C6FFc  20 EA E0   JSR E0EA    lit fichier selon X = POSNMX, VSALO0, VSALO1, DESALO
```

Initialise les pointeurs bas et haut, flags etc...

```
C702c  AD 6A 02   LDA 026A
C705c  48         PHA         empile les indicateurs de status console
C706c  20 40 D7   JSR D740    "CURSEUR OFF" (curseur caché = vidéo normale)
C709c  A5 9A      LDA 9A
C70Bc  A4 9B      LDY 9B
C70Dc  85 CE      STA CE      CE/CF = PTRBAS (début du programme BASIC)
C70Fc  84 CF      STY CF      (pointeur dans bloc du bas, c'est à dire dans le
C711c  AD 52 C0   LDA C052    programme initialement présent en mémoire)
C714c  AC 53 C0   LDY C053
C717c  85 F8      STA F8      F8/F9 = PTRHAUT (adresse début fichier chargé)
C719c  84 F9      STY F9      (pointeur dans le bloc du haut: FICHCOMPL)
C71Bc  78         SEI         interdit les interruptions
C71Cc  A2 20      LDX #20     X = "espace" placé dans
C71Ec  8E 4C C0   STX C04C    DEFAFF, code ASCII devant les nombres décimaux
C721c  86 F5      STX F5      force le b7 de F5 à zéro (flag "ligne non trouvée)
C723c  2C 74 C0   BIT C074    teste si le flag C074 "listing" est OFF
C726c  30 05      BMI C72D    si oui, saute les deux instructions suivantes
C728c  A2 01      LDX #01     sinon, indexe le message "Merging line:"
C72Ac  20 64 D3   JSR D364    XAFSC affiche X+1ème message externe terminé par "caractère + 128"
```

Teste si fin du programme haut (FICHCOMPL) est atteinte

```
C72Dc  A0 01      LDY #01     pour indexer HH du lien de ligne BASIC programme haut
C72Fc  B1 F8      LDA (F8),Y  teste si octet à PTRHAUT + 1 est nul (fin du programme)
C731c  D0 11      BNE C744    continue en C744 si ce n'est pas le cas
C733c  68         PLA
C734c  8D 6A 02   STA 026A    récupère le status console
C737c  58         CLI         autorise les interruptions
C738c  24 F5      BIT F5      teste si le flag "ligne trouvée" est à 0
C73Ac  10 05      BPL C741    si oui, continue en C741
C73Cc  A2 1B      LDX #1B     sinon, "LINES ALREADY EXISTS ERROR" (bogue: avec 1 faute!)
C73Ec  4C 7E D6   JMP D67E    incrémente X et traite l'erreur n° X

C741c  4C 06 D2   JMP D206    CBF0/ROM va à la ligne
```

Lecture (et affichage si besoin) du n° de ligne suivante du programme haut

```
C744c  C8         INY
C745c  B1 F8      LDA (F8),Y
C747c  85 33      STA 33
C749c  48         PHA
C74Ac  C8         INY
C74Bc  B1 F8      LDA (F8),Y  AY = 33/34 = n° de la ligne BASIC du programme haut
C74Dc  85 34      STA 34      (c'est le n° qu'il faudra chercher
C74Fc  A8         TAY         dans le programme bas)
C750c  68         PLA
C751c  2C 74 C0   BIT C074    teste si le flag C074 "listing" est OFF
C754c  30 08      BMI C75E    si oui, saute les trois instructions suivantes
C756c  20 53 D7   JSR D753    affichage en décimal sur 5 digits d'un nombre AY
```

759c A2 05 LDX #05 pour se replacer au début des 5 digits
75Bc 20 69 EE JSR EE69 affiche X fois "flèche gauche"

Calcule la longueur de la ligne à MERGEr

75Ec A0 03 LDY #03
760c C8 INY recherche le début de ligne suivante du programme haut
761c B1 F8 LDA (F8),Y (calcule le nombre d'octets qu'il faudra copier)
763c D0 FB BNE C760
765c 84 F6 STY F6 sauve Y dans F6 (nombre d'octets instructions propr dites)
767c A0 01 LDY #01 pour indexer le HH du lien
769c 20 A4 D1 JSR D1A4 C6C3/ROM cherche dans le programme du bas, à partir du pointeur courant CE/CF, l'adresse de la ligne BASIC portant le même n° que celle qui est en train d'être MERGEe, située dans le bloc du haut (FICHCOMPL) Si trouve, retourne avec C = 1 et adresse en CE/CF (1er octet de lien)
76Cc 90 04 BCC C772 si pas trouvé, saute les 2 instructions suivantes
76Ec 66 F5 ROR F5 le b7 de F5 est mis à 1 (flag "trouvé")
770c 30 39 BMI C7AB suite forcée C7AB (ajuste PTRHAUT p sauter ligne)

Remonte la fin du bloc du bas pour pouvoir inserer une ligne

772c A5 A0 LDA A0 ligne non trouvée, CE/CF contient adresse de la
774c A4 A1 LDY A1 ligne suivante, c'est à dire l'adresse du 1er octet
776c 85 C9 STA C9 du bas du bloc qu'il faut remonter pour faire place
778c 84 CA STY CA à la ligne qui sera inserée
77Ac 38 SEC C9/CA = AY = A0/A1 (fin des tableaux)
77Bc 65 F6 ADC F6 adresse du dernier octet du haut du bloc
77Dc 85 C7 STA C7 qu'il faut remonter
77Fc 48 PHA C7/C8 = AY = AY + F6 + #01
780c 98 TYA F6 = nombre d'octets à copier = longueur de ligne
781c 69 00 ADC #00 C7/C8 = adresse cible vers le haut
783c 85 C8 STA C8
785c A8 TAY
786c 68 PLA
787c 20 5C D1 JSR D15C C3F4/ROM décale un bloc memoire vers le haut. En CE/CF adresse du 1er octet du bas, en C9/CA adresse dernier octet du haut, en C7/C8 et AY adresse cible vers haut, "OUT OF MEMORY ERROR" si adresse cible > bas des chaînes A2/A3 revient avec nouveau début - #100 en C7/C8 et nouvelle fin en A0/A1 (haut des tableaux)

Mise à jour des pointeurs 9C/9D et 9E/9F
(début des variables et début des tableaux)

78Ac A2 02 LDX #02
78Cc 38 SEC
78Dc B5 9C LDA 9C,X l'adresse présente en 9E/9F (début des tableaux)
78Fc 65 F6 ADC F6 est incrémentée du contenu de F6 + 1
791c 95 9C STA 9C,X
793c 90 02 BCC C797
795c F6 9D INC 9D,X
797c CA DEX l'adresse présente en 9C/9D (début des variables)
798c CA DEX est incrémentée du contenu de F6 + 1
799c 10 F1 BPL C78C (reboucle une fois en C78C)
79Bc A4 F6 LDY F6 nombre d'octets à copier
79Dc B1 F8 LDA (F8),Y lit octet à PTRHAUT
79Fc 91 CE STA (CE),Y écrit octet à PTRBAS
7A1c 88 DEY octet précédent (opère par la fin)
7A2c 10 F9 BPL C79D reboucle tant qu'il en reste
7A4c A5 CE LDA CE
7A6c A4 CF LDY CF AY = adresse du nouveau bloc
7A8c 20 8C D1 JSR D18C C563/ROM restaure les liens à partir de l'adresse AY

Mise à jour de PTRHAUT

7ABc 38 SEC
7ACc A5 F6 LDA F6
7AEc 65 F8 ADC F8 F8/F9 = F8/F9 + F6 + #01
7B0c 85 F8 STA F8
7B2c 90 02 BCC C7B6
7B4c E6 F9 INC F9
7B6c 4C 2D C7 JMP C72D reprise forcée en C72D

Messages externes de la banque n°3 (C7B9 à C7DC)

7B9c 20 46 6F 75 6E 64 73 0A **8D**
■FoundsLF**CR** (NB: Avec une belle faute car le participe passé ne prend jamais la forme plurielle en anglais!)
7C2c 0A 0D 4D 65 72 67 69 6E 67 20 6C 69 6E 65 **BA**
LFCRMerging line**:**
7D1c 0A 0D 4C 49 4E 45 20 **BA**

03 LFCRLINE **:**
C7D9c 4C 4F 41 **C4**
04 LOA**D**

Messages d'erreur externe de la banque n°3 (C7DD à C7FE)
C7DDc 49 4E 56 41 4C 49 44 20 53 54 52 49 4E **C7**
01 INVALID STRIN**G**
C7EBc 4C 49 4E 45 53 20 41 4C 52 45 41 44 59 20 45 58 49 53 54 **D3**
02 LINES ALREADY EXIST**S** (NB: Avec une belle faute d'accord!)
C7FFc 00 BRK (inutilisé)

Banque n°4 (adresse Cxxxd): COPY

Cette banque se trouve à partir du #51 (81ème) secteur de la disquette MASTER

400d	00 00	EXTER	adresse des messages d'erreur externe (néant)
402d	8F C7	EXTMS	adresse des messages externes

EXECUTION COMMANDE SEDORIC COPY

Rappel de la syntaxe:
a) **COPY** (ou **COPYO) NFA source TO NFA cible** (**,C**) (**,N**) il doit y avoir correspondance entre les jokers du NFA source et ceux du NFA cible ou alors le NFA cible doit être *.* ou omis (ce qui revient au même).
b) **COPYM NFA source TO NF cible** (**,C**) (**,N**) les jokers ne sont pas autorisés, dans le fichier cible.
Dans les deux cas, l'option ",C" permet que soit demandée confirmation pour chacun des fichiers corresp à un NFA source comportant des jockers. L'option ",N" inhibe la demande de changement de disquette lorsqu'on travaille avec un seul lecteur. Par exemple, COPY"F1"TO"F2",N affiche: LOAD DISCS FOR COPY FROM A TO A AND PRESS RETURN et effectue la copie en une seule fois sans demander les disquettes source et cible.

Variables utilisées

0/01	n° de PISTE et n° de SECTEUR actifs en lecture
2/03	n° de PISTE et n° de SECTEUR actifs en écriture
4	initialisé à #00 (b7 à zéro si 1ère passe de lecture)
5	n° du message d'erreur
6	initialisé à #80
7	b6 à 1 si option ",C" (confirmation demandée avant copie)
	b7 à 1 si option ",N" (inhibition de demande changement de disquette)
A/0B	nombre de secteurs restant à sauver
6	b6 à 1 si COPYM (et à 0 si COPY ou COPYO)
	b7 à 1 si COPY (et à 0 si COPYM ou COPYO)
4	b7 à 1 s'il y a au moins un "?" dans le nom de fichier source
5/F6	RWBUF
7/F8	nombre de secteurs à charger
9	POSNMX puis flag "lecture/écriture" (b7 à 1 si écriture)
025	POSNMP
026	POSNMS
027	POSNMX
08C	position dans la liste des coordonnées des secteurs à charger
08D/C08E	nombre de secteurs restant à charger
090/C09C	NFA source
09D/C09A	NFA cible

Informations non documentées
La dénomination COPYM (pour MERGE = mélanger) n'est pas très bonne. COPYJ (pour JOINT = joindre) aurait été plus judicieuse. EN effet, les fichiers sont mis bout à bout, un peu comme avec CLOAD,J ou LOAD,J et non pas mixés comme avec la commande MERGE. Il faut encore noter que les commandes CLOAD,J LOAD,J et MERGE concernent uniquement des fichiers BASIC, alors que COPY,M opère avec des fichiers de tous types.

Comme avec CHANGE, le TO de COPY doit obligatoirement être en majuscules. COPY"TOTO"TO"BOBO" marche, ainsi que copy"TOTO"TO"BOBO", mais COPY"TOTO"to"BOBO" déclenchera un SYNTAX ERROR.

Il semble que le NFA peut être omis (dans ce cas, tous les fichiers du drive courant seront copiés). Curieusement le "TO" devrait pouvoir être omis. Finalement, le NFA cible peut aussi être omis (le drive courant sera utilisé comme drive cible) sauf bien sûr avec COPYM qui réclame un nom de fichier non ambigu. Donc COPY et COPYO tout court devraient pouvoir marcher. Voir ce qui se passe alors avec COPY,N.

Attention, l'ordre des options est important: si on tape ",C,N" comme indiqué dans le manuel, l'option ",N" annule l'option ",C". Pour avoir ces deux options, il faut indiquer ",N,C"!

Utilisation en LM
Si la banque n°4 est déjà en place, il suffira de basculer sur la RAMOV, d'initialiser l'adresse 07 avec le flag "inhibition demande disquette/confirmation", l'adresse 16 avec le flag COPY, COPYO ou COPYM, la zone C090/C09CV avec NFA source, zone C09D/C09E avec NFA cible (dans les deux cas utiliser un ou des "?" au lieu de "*"), puis de faire un JSR C567. Sinon, il faut écrire les paramètres dans le tampon clavier, initialiser TXTPTR puis faire le JSR F148 (voir les détails en annexe).

Analyse de la syntaxe et saisie des paramètres

404d	C9 4D	CMP #4D	est-ce un "M" ? (Merge)
406d	F0 07	BEQ C40F	si oui, continue en C40F
408d	C9 4F	CMP #4F	est-ce un "O" ? (Over)
40Ad	D0 09	BNE C415	sinon, continue en C415 (COPY tout court)

```
C40Cd  A0 00      LDY #00      Y = #00 pour flag "COPYO" (b7 et b6 à zéro)
C40Ed  2C A0 40   BIT 40A0     continue en C411
C40Fd  A0 40      LDY #40      Y = #40 pour flag "COPYM" (b7 à zéro et b6 à 1)
C411d  20 98 D3   JSR D398     XCRGET lit le caractère suivant à TXTPTR et le convertit en MAJUSCULE
C414d  2C A0 80   BIT 80A0     continue en C417
C415d  A0 80      LDY #80      Y = #80 pour flag "COPY" (b7 à 1 et b6 à zéro)
C417d  84 16      STY 16       16 porte donc le flag "COPY" ou "COPYO" ou "COPYM"
C419d  A9 00      LDA #00      force à zéro le flag 07 (pas de confirmation, pas
C41Bd  85 07      STA 07       d'inhibition de demande de changement de disquette)
C41Dd  20 51 D4   JSR D451     XNFA lit un nom de fichier ambigu à TXTPTR et l'écrit dans BUFNOM
C420d  20 DE DF   JSR DFDE     vérifie mode TEXT (sinon "DISP TYPE MISMATCH ERROR")
C423d  A2 0C      LDX #0C      index pour copie de 13 octets (drive+nom+extension)
C425d  BD 28 C0   LDA C028,X   lit un octet dans BUFNOM
C428d  9D 90 C0   STA C090,X   et le copie dans zone C090/C09C (NFA source)
C42Bd  CA         DEX          octet précédent
C42Cd  10 F7      BPL C425     reboucle tant qu'il en reste
C42Ed  20 9E D3   JSR D39E     XCRGOT relit le caractère à TXTPTR et le convertit en MAJUSCULE
C431d  F0 09      BEQ C43C     s'il n'y a plus de paramètres, continue en C43C
C433d  C9 2C      CMP #2C      est-ce une ","?
C435d  F0 05      BEQ C43C     si oui, continue en C43C (paramètre omis)
C437d  A9 C3      LDA #C3      sinon, A = token "TO"
C439d  20 2E D2   JSR D22E     JSR D067/ROM et D3A1/RAMOV demande "TO" à TXTPTR,
                               lit le caractère suivant à TXTPTR et le convertit en MAJUSCULE
C43Cd  20 51 D4   JSR D451     XNFA lit un nom de fichier ambigu à TXTPTR et l'écrit dans BUFNOM
C43Fd  48         PHA          sauve A sur la pile (paramètre suivant)
C440d  A2 0C      LDX #0C      index pour copie de 13 octets (drive+nom+extension)
C442d  BD 28 C0   LDA C028,X   lit un octet dans BUFNOM
C445d  9D 9D C0   STA C09D,X   et le copie dans zone C09D/C0A9 (NFA ou NF cible)
C448d  24 16      BIT 16       teste si b6 du flag "COPY*" est à zéro
C44Ad  50 07      BVC C453     si oui (ce n'est pas COPYM), continue en C453
C44Cd  C9 3F      CMP #3F      si COPYM, l'octet lu est-il un "?"?
C44Ed  D0 03      BNE C453     sinon (OK), saute l'instruction suivante
C450d  4C AC D5   JMP D5AC     "INVALID FILE NAME"
C453d  CA         DEX          indexe l'octet précédent
C454d  10 EC      BPL C442     et reboucle s'il en reste à copier
C456d  68         PLA          récupère A (paramètre suivant)
C457d  F0 1E      BEQ C477     continue en C477 s'il n'y a plus de paramètres
C459d  20 2C D2   JSR D22C     D067/ROM exige une "," lit le caractère suivant et le convertit en MAJUSCULE
C45Cd  C9 43      CMP #43      est-ce un "C"? (confirmation demandée)
C45Ed  D0 08      BNE C468     sinon, continue en C468
C460d  A5 07      LDA 07       si oui, force le b6 de 07 à 1
C462d  09 40      ORA #40
C464d  85 07      STA 07
C466d  D0 0A      BNE C472     et suite forcée en C472
C468d  C9 4E      CMP #4E      est-ce un "N"? (inhibition de demande de changement de disquette)
                               NB: #4E = 0100 1110 sera shifté vers la gauche en 1001 1100.
C46Ad  F0 03      BEQ C46F     si oui, saute l'instruction suivante
C46Cd  4C 23 DE   JMP DE23     sinon (ni ",C" ni ",N"), "SYNTAX ERROR"
C46Fd  0A         ASL          force à 1 le b7 de 07, mais force aussi à 0 le b6
C470d  85 07      STA 07       donc option ",N" annule option ",C" sauf si ",N,C"!
C472d  20 98 D3   JSR D398     XCRGET lit le caractère suivant à TXTPTR et le convertit en MAJUSCULE
C475d  D0 E2      BNE C459     reprend en C459 s'il y a encore un paramètre

       Initialise drive source = drive actif et BUFNOM avec NFA source
C477d  20 58 FE   JSR FE58     copie NFA source dans BUFNOM et vérifie les jockers
                               (revient avec b7 de F4 à 1 s'il y a au moins un "?" dans le NFA source)
C47Ad  AD 90 C0   LDA C090     n° du drive source
C47Dd  8D 00 C0   STA C000     devient DRIVE actif

       Force à 1 le b7 du flag "demande changement disquette inhibée" si multidrive
C480d  24 07      BIT 07       teste si le b7 de 07 est déjà à 1 (demande inhibée)
C482d  30 0B      BMI C48F     si oui, continue directement en C48F
C484d  CD 9D C0   CMP C09D     le drive source est-il identique au drive cible?
C487d  F0 06      BEQ C48F     si oui, continue en C48F (on laisse le flag à zéro)
C489d  A5 07      LDA 07       sinon, force à 1 le b7 de 07 (si multidrive, seule
C48Bd  09 80      ORA #80      la demande initiale sera affichée et les demandes
C48Dd  85 07      STA 07       ultérieures seront inhibées d'office)

       Demande initiale de disquette(s)
       Soit       LOAD DISCS FOR COPY FROM X TO X CRLF AND PRESS 'RETURN' LFLF
                  si b7 de 07 à 1 (multidrive ou monodrive avec demande inhibée)
```

```
      ou              LOAD SOURCE DISC IN DRIVE X CRLF AND PRESS 'RETURN'LFLF
                      si b7 de 07 à zéro (uniquement monodrive avec demande non inhibée)
48Fd  20 06 D2        JSR D206        CBF0/ROM va à la ligne
492d  24 07           BIT 07          teste si le b7 de 07 est à 1
494d  30 0C           BMI C4A2        si oui, continue en C4A2 (multidrive ou monodrive avec demande inhibée: demande la ou
      les disquettes uniquement au départ) Sinon (monodrive et demande non inhibee), demande d'abord la disquette source.
496d  A2 00           LDX #00         indexe le message "LOAD SOURCE"
498d  20 64 D3        JSR D364        XAFSC affiche X+1ème message externe terminé par "caractère + 128"
49Bd  20 48 D6        JSR D648        affiche "DISC IN DRIVE"lettre du lecteur" AND PRESS RETURN'
                                        puis demande un 'ESC' (C = 1) ou un 'RETURN' (C = 0)
49Ed  90 2B           BCC C4CB        si "RETURN", continue en C4CB
4A0d  58              CLI             si "ESC", autorise les interruptions
4A1d  60              RTS             et retourne

      Demande initiale uniquement
4A2d  A2 03           LDX #03         index le message "LOAD DISCS FOR COPY FROM "
4A4d  20 64 D3        JSR D364        XAFSC affiche X+1ème message externe terminé par "caractère + 128"
4A7d  AD 90 C0        LDA C090        n° du lecteur source
4AAd  20 0E D6        JSR D60E        convertit n° lecteur en lettre et l'affiche
4ADd  A2 04           LDX #04         indexe le message " TO "
4AFd  20 64 D3        JSR D364        XAFSC affiche X+1ème message externe terminé par "caractère + 128"
4B2d  AD 9D C0        LDA C09D        n° du lecteur cible
4B5d  20 0E D6        JSR D60E        convertit n° lecteur en lettre et l'affiche
4B8d  20 06 D2        JSR D206        CBF0/ROM va à la ligne
4BBd  A2 0D           LDX #0D         indexe " AND PRESS 'RETURN'"
4BDd  20 6C D3        JSR D36C        affiche X+1ème message de zone CEE7 terminé par "caractère + 128"
4C0d  20 69 D6        JSR D669        demande un 'ESC' (C = 1) ou un 'RETURN' (C = 0)
4C3d  B0 DB           BCS C4A0        si "ESC", continue en C4A0 (simple CLI et RTS)
4C5d  20 06 D2        JSR D206        CBF0/ROM va à la ligne
4C8d  20 06 D2        JSR D206        CBF0/ROM va à la ligne

      Recherche le fichier source
4CBd  20 2D DB        JSR DB2D        vérifie que la disquette en place est bien une disquette Sédoric, 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
4CEd  D0 03           BNE C4D3        si trouvé, saute l'instruction suivante
4D0d  4C DD E0        JMP E0DD        si pas trouvé, "FILE NOT FOUND ERROR"
4D3d  86 F9           STX F9          sauve POSNMX dans F9
4D5d  24 07           BIT 07          teste si le b6 de 07 est à zéro
4D7d  50 2B           BVC C504        si oui (sans confirmation), continue en C504

      Demande s'il faut copier le fichier source
4D9d  20 B4 DA        JSR DAB4        affiche nom de fichier présent à POSNMX dans BUF3
4DCd  A2 0A           LDX #0A         indexe " (Y)es or (N)o:CRLF"
4DEd  20 6C D3        JSR D36C        affiche X+1ème message de zone CEE7 terminé par "caractère + 128"
4E1d  58              CLI             autorise les interruptions
4E2d  20 02 D3        JSR D302        JSR EB78/ROM si touche frappée alors N = 1 et A = code ASCII corresp, sinon N = 0
4E5d  20 A1 D3        JSR D3A1        minMAJ convertit en MAJUSCULE le caractère dans A
4E8d  C9 1B           CMP #1B         est-ce un "ESC"?
4EAd  F0 B4           BEQ C4A0        si oui, CLI et RTS en C4A0
4ECd  C9 4E           CMP #4E         est-ce un "N"?
4EEd  D0 03           BNE C4F3        sinon, saute l'instruction suivante
4F0d  4C 77 C5        JMP C577        si oui (on ne copie pas), continue en C577
4F3d  C9 59           CMP #59         est-ce un "Y"?
4F5d  D0 EB           BNE C4E2        sinon, reprend en C4E2 (nouvelle saisie de touche)
4F7d  20 2A D6        JSR D62A        XAFCAR affiche le caractère ASCII contenu dans A
4FAd  20 06 D2        JSR D206        CBF0/ROM va à la ligne
4FDd  24 07           BIT 07          teste si le b7 de 07 est à 1
4FFd  30 03           BMI C504        si oui (inhibe demande disc cible) saute l'instruction suivante
501d  20 06 D2        JSR D206        CBF0/ROM va à la ligne

      Copie le fichier source
504d  AD 25 C0        LDA C025        empile POSNMP
507d  48              PHA
508d  AD 26 C0        LDA C026        empile POSNMS
50Bd  48              PHA
50Cd  AD 27 C0        LDA C027        empile POSNMX
50Fd  48              PHA             (coordonnées pour fichier source)
510d  20 8D C5        JSR C58D        transfère les secteurs d'un fichier
513d  68              PLA
514d  A8              TAY             récupère POSNMX dans Y et dans F9
```

```
C515d  85 F9        STA F9
C517d  68           PLA
C518d  8D 02 C0     STA C002      récupère POSNMS dans SECTEUR et dans C026
C51Bd  8D 26 C0     STA C026
C51Ed  68           PLA           récupère POSNMP dans A
C51Fd  90 03        BCC C524      saute l'instruction suivante si C = 0
C521d  4C A0 C4     JMP C4A0      si C = 1, simple CLI et RTS en C4A0
C524d  8D 01 C0     STA C001      sinon, copie POSNMP dans n° de PISTE active
C527d  8D 25 C0     STA C025      et dans C025 (POSNMP)
C52Ad  AD 90 C0     LDA C090      n° de drive source
C52Dd  8D 00 C0     STA C000      devient n° de drive DRIVE actif
C530d  24 16        BIT 16        teste si b6 du flag "COPY*" est à zéro
C532d  50 06        BVC C53A      si oui (ce n'est pas COPYM), saute les 2 instructions suivantes
C534d  8C 27 C0     STY C027      si COPYM, copie POSNMX dans C027
C537d  20 73 DA     JSR DA73      XPRSEC lit un secteur selon DRIVE, PISTE, SECTEUR et RWBUF
C53Ad  A6 05        LDX 05        n° de message
C53Cd  D0 05        BNE C543      si X <> #00, saute les deux instructions suivantes
C53Ed  20 06 D2     JSR D206      CBF0/ROM va à la ligne
C541d  10 17        BPL C55A      si b7 de X est nul, continue en C55A
C543d  20 B4 DA     JSR DAB4      affiche nom de fichier présent à POSNMX dans BUF3
C546d  A6 05        LDX 05        indexe le message à afficher
C548d  24 16        BIT 16        teste si b6 du flag "COPY*" est à zéro
C54Ad  50 02        BVC C54E      si oui (ce n'est pas COPYM), continue en C54E
C54Cd  A2 07        LDX #07       si COPYM, X = #07
C54Ed  E0 09        CPX #09       teste si X >= #09
C550d  B0 05        BCS C557      si oui, continue en C557, sinon...
C552d  20 64 D3     JSR D364      XAFSC affiche X+1ème message externe terminé par "caractère + 128"
C555d  30 03        BMI C55A      et continue en C55A
C557d  20 6C D3     JSR D36C      affiche X+1ème message de zone CEE7 terminé par "caractère + 128"
C55Ad  20 58 FE     JSR FE58      copie NFA source dans BUFNOM
                                  et vérifie les jockers (revient avec b7 de F4 à 1 s'il y a au moins un "?" dans le NFA source)
C55Dd  24 F4        BIT F4        teste si le b7 de F4 est nul (pas de "?" dans source)
C55Fd  10 26        BPL C587      si oui, continue en C587
C561d  24 07        BIT 07        teste si le b7 de 07 est à 1
C563d  30 0D        BMI C572      si oui (inhibe demande disquette), continue en C572
C565d  20 06 D2     JSR D206      CBF0/ROM va à la ligne
C568d  A2 00        LDX #00       indexe le message "LOAD SOURCE"
C56Ad  20 64 D3     JSR D364      XAFSC affiche X+1ème message externe terminé par "caractère + 128"
C56Dd  20 48 D6     JSR D648      affiche "DISC IN DRIVE"lettre du lecteur" AND PRESS RETURN"
                                  puis demande un "ESC" (C = 1) ou un "RETURN" (C = 0)
C570d  B0 4A        BCS C5BC      si "ESC", continue en C5BC (simple CLI et RTS)
C572d  20 73 DA     JSR DA73      XPRSEC lit un secteur selon DRIVE, PISTE, SECTEUR et RWBUF
C575d  F0 06        BEQ C57D      si pas trouvé, saute les 2 instructions suivantes
C577d  20 2A D6     JSR D62A      XAFCAR affiche le caractère ASCII contenu dans A
C57Ad  20 06 D2     JSR D206      CBF0/ROM va à la ligne
C57Dd  A5 F9        LDA F9        A = POSNMX (ancienne valeur)
C57Fd  20 44 DB     JSR DB44      X = POSNMX pour "entrée" suivante et reprend la comparaison
C582d  F0 03        BEQ C587      si pas trouvé, saute l'instruction suivante
C584d  4C D3 C4     JMP C4D3      si trouvé, reprend en C4D3
C587d  58           CLI           autorise les interruptions
C588d  A2 05        LDX #05       indexe le message "LFCRCopy completeLFCR"
C58Ad  4C 64 D3     JMP D364      XAFSC affiche X+1ème message externe terminé par "caractère + 128"

       Transfert des secteurs d'un fichier: Initialise variables pour lecture
C58Dd  78           SEI           interdit les interruptions
C58Ed  A9 00        LDA #00
C590d  85 0A        STA 0A        force 0A/0B à zéro
C592d  85 0B        STA 0B        force VSALO0 à zéro (pour indiquer ni ",V" ni ",N")
C594d  8D 4D C0     STA C04D      force 04 à zéro (b7 à 0 si 1ère passe lecture,
C597d  85 04        STA 04        et b6 à 0 si 1ère passe écriture)
C599d  85 F5        STA F5        LL de RWBUF à zéro pour former B400 ultérieurement
C59Bd  A9 80        LDA #80
C59Dd  85 06        STA 06        force le b7 de 06 à 1 et les autres bits à zéro
C59Fd  AD 90 C0     LDA C090      n° de drive source
C5A2d  8D 00 C0     STA C000      devient n° de DRIVE actif
C5A5d  46 F9        LSR F9        force à zéro le b7 de F9 (flag "lecture")
C5A7d  24 04        BIT 04        teste si le b7 de 04 est nul (1ère passe de lecture)
C5A9d  10 13        BPL C5BE      si oui, continue en C5BE, sinon passe ultérieure...

       Demande éventuellement la disquette source
```

```
5ABd 24 07      BIT 07      teste si le b7 de 07 est à 1
5ADd 30 1A      BMI C5C9    si oui (inhibe demande disquette), continue en C5C9
5AFd 20 06 D2   JSR D206    CBF0/ROM va à la ligne
5B2d A2 00      LDX #00     indexe le message "LOAD SOURCE"
5B4d 20 64 D3   JSR D364    XAFSC affiche X+1ème message externe terminé par "caractère + 128"
5B7d 20 48 D6   JSR D648    affiche "DISC IN DRIVE"lettre du lecteur" AND PRESS RETURN"
                            puis demande un "ESC" (C = 1) ou un "RETURN" (C = 0)
5BAd 90 0D      BCC C5C9    si "RETURN", continue en C5C9
5BCd 58         CLI         si "ESC", autorise les interruptions
5BDd 60         RTS         et retourne

        1ère passe en lecture: prend les coordonnées du 1er descripteur
5BEd AE 27 C0   LDX C027    X = POSNMX
5C1d BD 0C C3   LDA C30C,X  lit PISTE dans BUF3 à la position POSNMX + #0C
5C4d BC 0D C3   LDY C30D,X  lit SECTEUR dans BUF3 à la position POSNMX + #0D
                            (ce sont les coordonnées PISTE/SECTEUR du premier descripteur du fichier)
5C7d D0 04      BNE C5CD    secteur jamais nul = suite forcée en C5CD

        Passe ultérieure en lecture: reprend les coordonnées du descripteur en cours
5C9d A5 00      LDA 00      récupère le n° de PISTE active en lecture
5CBd A4 01      LDY 01      récupère le n° de SECTEUR actif en lecture
5CDd 20 5D DA   JSR DA5D    XPBUF1 Charge dans BUF1 le secteur Y de la piste A
5D0d A0 B4      LDY #B4     (c'est à dire le descripteur du fichier source)
5D2d 8C 04 C0   STY C004    HH de RWBUF = #B4 pour former l'adresse B400
5D5d 84 F6      STY F6      force le b7 de F6 à 1 pour indiquer que l'on va charger le premier descripteur
5D7d AE 04 C0   LDX C004    X = HH du RWBUF courant
5DAd E0 05      CPX #05     teste si la limite inférieure (#05) est atteinte
5DCd F0 1D      BEQ C5FB    si oui, continue en C5FB, sinon...
5DEd 20 73 DA   JSR DA73    XPRSEC lit un secteur selon DRIVE, PISTE, SECTEUR et RWBUF
5E1d 06 06      ASL 06      récupère le b7 de 06 dans C qui sera donc toujours nul, sauf lors du 1er tour,
                            pour indiquer qu'on vient de charger en RAM le 1et descripteur du fichier
5E3d 20 0C C7   JSR C70C    lecture des secteurs du fichier dans la limite de la place disponible en RAM
5E6d AC 04 C0   LDY C004    HH de RWBUF
5E9d 8C 8F C0   STY C08F    position actuelle du pointeur sauvegardée en C08F
5ECd B0 0D      BCS C5FB    continue en C5FB (il reste des secteurs à charger)
5EEd A8         TAY         Y = pointeur dans la liste des decripteurs
5EFd 20 2A E2   JSR E22A    teste si Y est OK, puis charge éventuellement le descripteur suivant
                            et enfin met Y à jour pour viser le descripteur suivant
5F2d 78         SEI         interdit les interruptions
5F3d B0 05      BCS C5FA    continue en C5FA si C = 1 (il n'y a plus de descript)
5F5d 38         SEC
5F6d 66 06      ROR 06      force à 1 le b7 de 06
5F8d 30 DD      BMI C5D7    reprise forcée en C5D7 si b7 = 1

        Il n'y a plus de descripteur
5FAd 18         CLC         force C = zéro

        Initialise le drive cible
5FBd 66 04      ROR 04      C -> b7 (0 il n'y a plus de descrip, 1 il en reste)
5FDd AD 9D C0   LDA C09D    n° de drive cible
600d 8D 00 C0   STA C000    devient DRIVE actif
603d 24 07      BIT 07      teste si le b7 de 07 est à 1
605d 30 0A      BMI C611    si oui (inhibe demande disquette), continue en C611
607d A2 01      LDX #01     sinon, indexe le message "LOAD TARGET"
609d 20 64 D3   JSR D364    XAFSC affiche X+1ème message externe terminé par "caractère + 128"
60Cd 20 48 D6   JSR D648    affiche "DISC IN DRIVE"lettre du lecteur" AND PRESS RETURN"
                            puis demande un "ESC" (C = 1) ou un "RETURN" (C = 0)
60Fd B0 AB      BCS C5BC    si "ESC", continue en C5BC (simple CLI et RTS)
611d 38         SEC
612d 66 F9      ROR F9      force à 1 le b7 de F9 (flag "écriture")
614d 24 04      BIT 04      teste si le b6 de 04 à 0 (1ère passe en écriture)
616d 50 0A      BVC C622    si oui, continue en C622, sinon, reprend les
618d A5 02      LDA 02      coordonnées du descripteur en cours:
61Ad A4 03      LDY 03      A = piste selon 02 et Y = secteur selon 03
61Cd 20 5D DA   JSR DA5D    XPBUF1 Charge dans BUF1 le secteur Y de la piste A
61Fd 38         SEC         force C = 1
620d F0 4B      BEQ C66D    suite forcée en C66D
622d AE 27 C0   LDX C027    X = POSNMX
625d A0 00      LDY #00     index d'exploration
627d B9 9E C0   LDA C09E,Y  lit un octet du nom de fichier cible
62Ad C9 3F      CMP #3F     est-ce un "?"?
```

```
C62Cd  D0 03      BNE C631     sinon, saute l'instruction suivante
C62Ed  BD 00 C3   LDA C300,X   si oui, lit un octet à la position X de BUF3
C631d  99 29 C0   STA C029,Y   et le copie dans BUFNOM à la position Y
C634d  E8         INX          octet cible suivant
C635d  C8         INY          octet BUFNOM suivant
C636d  C0 0C      CPY #0C      12 caractères ont-ils été copiés?
C638d  D0 ED      BNE C627     sinon, reboucle en C627
C63Ad  A9 00      LDA #00      si oui, force à zéro PSDESP (coorrdonnées du descripteur principal)
                               et NSTOTP (nombre de secteurs totaux + PROT/UNPROT)
C63Cd  99 29 C0   STA C029,Y   c'est à dire les octets de C035 à C038
C63Fd  C8         INY          octet suivant (de #0C à #0F soit 4 octets)
C640d  C0 10      CPY #10      teste si valeur limite de Y atteinte
C642d  D0 F8      BNE C63C     sinon, reboucle en C63C
C644d  20 2D DB   JSR DB2D     vérifie que la disquette en place est bien une disquette Sédoric, 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
C647d  F0 1D      BEQ C666     continue en C666 si fichier n'existe pas sur cible
C649d  24 16      BIT 16       si existe, teste les b7 et b6 de 16 (flag COPY*)
C64Bd  30 0F      BMI C65C     si b7 = 1 (COPY), continue en C65C, sinon...
C64Dd  50 05      BVC C654     si b6 = 0 (COPYO), continue en C654
C64Fd  20 07 DB   JSR DB07     si b6 = 1 (COPYM), XCABU copie la ligne d'"entrée" de catalogue à POSNMX (BUF3)
       dans BUFNOM (cette mise à jour inclu PSDESP coordonnées descripteur principal et NSTOTP nombre de secteurs totaux
       + PROT)
C652d  F0 12      BEQ C666     suite forcée en C666 si Z = 1
C654d  20 64 E2   JSR E264     DELète le fichier indexé à POSNMX dans BUF3
C657d  90 0A      BCC C663     continue en C663 si C = 0 (pas d'erreur)
C659d  A9 00      LDA #00      A = #00 pour message "LFCRTRACK:"
C65Bd  2C A9 0E   BIT 0EA9     et continue en C65E
C65Cd  A9 0E      LDA #0E      A = #0E pour message "ALREADY EXISTSLFCR"
C65Ed  85 05      STA 05       sauve A dans 05 (n° du message d'erreur)
C660d  18         CLC          force C = 0 pour indiquer l'existence d'une erreur
C661d  58         CLI          autorise les interruptions
C662d  60         RTS          et retourne

       Pas d'erreur
C663d  A9 06      LDA #06      A = #06
C665d  2C A9 08   BIT 08A9     et continue en C668
C666d  A9 08      LDA #08      A = #08
C668d  85 05      STA 05       sauve A dans 05
C66Ad  38         SEC
C66Bd  66 06      ROR 06       force à 1 le b7 de 06
C66Dd  A9 00      LDA #00
C66Fd  A0 B4      LDY #B4      F5/F6 = #B400 (RWBUF)
C671d  85 F5      STA F5
C673d  84 F6      STY F6
C675d  8C 04 C0   STY C004     HH de RWBUF = #B4
C678d  90 04      BCC C67E     saute les deux instructions suivantes si C = 0
C67Ad  24 06      BIT 06       teste si le b7 de 06 est à zéro
C67Cd  10 59      BPL C6D7     si oui, continue en C6D7
C67Ed  AD 04 C0   LDA C004
C681d  48         PHA          empile HH de RWBUF
C682d  08         PHP          sauvegarde les indicateurs 6502
C683d  90 15      BCC C69A     continue en C69A si C = 0
C685d  66 06      ROR 06       si C = 1, force à 1 le b7 de 06
C687d  20 6C DC   JSR DC6C     XLIBSE cherche un secteur libre dans la bitmap dans BUF2,
C68Ad  8D 00 C1   STA C100       retourne avec A = n° de piste et Y = n° de secteur (sinon "DISK FULL ERROR")
C68Dd  8C 01 C1   STY C101     AY = coordonnées du descripteur suivant
C690d  20 15 DD   JSR DD15     XDETSE Libère le secteur AY sur le bitmap
C693d  A5 02      LDA 02
C695d  A4 03      LDY 03       AY = coordonnées du
C697d  20 91 DA   JSR DA91     XSBUF1 sauve BUF1 à la piste A et le secteur Y
C69Ad  A0 0B      LDY #0B      index pour copier 12 octets
C69Cd  B1 F5      LDA (F5),Y   lit un octet selon F5/F6 + Y
C69Ed  99 4E C0   STA C04E,Y   et l'écrit dans la zone C04F à C059
                               (LGSALO, FTYPE, DESALO, FISALO, EXSALO et NSRSAV)
C6A1d  88         DEY          indexe l'octet précédent
C6A2d  D0 F8      BNE C69C     reboucle en C69C tant qu'il en reste
C6A4d  AD 58 C0   LDA C058
C6A7d  48         PHA          empile LL de NSRSAV
C6A8d  AC 59 C0   LDY C059     Y = HH de NSRSAV
C6ABd  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/C05B), les coordonnées du 1er secteur descripteur dans PSDESC (C05C/C05D), le nombre de descripteurs utilisés dans NSDESC (C05E) et 1er descripteur en place

```
6AEd 68          PLA          récupère LL de NSRSAV
6AFd 18          CLC          prépare une addition pour calculer dans 0A/0B le nombre total de secteurs
6B0d 65 0A       ADC 0A
6B2d 85 0A       STA 0A       0A = 0A + LL de NSRSAV
6B4d 48          PHA
6B5d A5 0B       LDA 0B
6B7d 6D 5B C0    ADC C05B
6BAd 85 0B       STA 0B       0B = 0B + HH de NSRSAV
6BCd 68          PLA
6BDd 6D 5E C0    ADC C05E
6C0d 85 0A       STA 0A
6C2d 90 02       BCC C6C6
6C4d E6 0B       INC 0B
6C6d 28          PLP          récupère les indicateurs 6502
6C7d B0 0A       BCS C6D3     continue en si C = 1
6C9d AD 5C C0    LDA C05C
6CCd AC 5D C0    LDY C05D
6CFd 85 08       STA 08       08/09 = C05C/C05D (PSDESC)
6D1d 84 09       STY 09
6D3d 68          PLA
6D4d 8D 04 C0    STA C004     récupère HH de RWBUF
6D7d 06 06       ASL 06
6D9d 20 0C C7    JSR C70C
6DCd AC 04 C0    LDY C004
6DFd 84 F6       STY F6       F6 = HH de RWBUF
6E1d 88          DEY
6E2d CC 8F C0    CPY C08F     teste si Y >= C08F
6E5d B0 97       BCS C67E     si oui, reprend en C67E
6E7d 24 04       BIT 04       sinon, teste si le b7 de 04 est à zéro
6E9d 10 03       BPL C6EE     si oui, saute l'instruction suivante
6EBd 4C 9F C5    JMP C59F     sinon, reprend en C59F
6EEd A5 08       LDA 08
6F0d A4 09       LDY 09       C05C/C05D (PSDESC) = 08/09
6F2d 8D 5C C0    STA C05C
6F5d 8C 5D C0    STY C05D
6F8d A5 0A       LDA 0A
6FAd A4 0B       LDY 0B
6FCd A2 00       LDX #00
6FEd 8E 5E C0    STX C05E     force C05E à zéro
701d 8D 5A C0    STA C05A
704d 8C 5B C0    STY C05B     C05A/C05B = 0A/0B
707d 20 81 DF    JSR DF81     Mise à jour du nombre de secteurs totaux du fichier
70Ad 18          CLC
70Bd 60          RTS
```

Lecture/écriture des secteurs: Initialise Y = position dans liste et F8/F8 = nombre de secteurs à charger

```
70Cd 90 0A       BCC C718     continue en C718 si C = zéro
70Ed A9 0A       LDA #0A      sinon (C = 1) on est au début de la 1ère passe...
710d AE 0A C1    LDX C10A     A = 10 position initiale dans liste des coordonnées
713d AC 0B C1    LDY C10B     XY = nombre de secteurs à charger
716d B0 09       BCS C721     suite forcée en C721
718d AE 8D C0    LDX C08D     on est au début d'une passe ultérieure...
71Bd AC 8E C0    LDY C08E     XY = nombre de secteurs à charger
71Ed AD 8C C0    LDA C08C     A = position dans la liste des coordonnées
721d E8          INX
722d D0 01       BNE C725     incrémente le nombre de secteurs à transférer
724d C8          INY          pour avoir valeur correcte en début de boucle C72D
725d 86 F7       STX F7       et sauve le résultat en F7/F8
727d 84 F8       STY F8
729d A8          TAY          Y = position dans la liste des coordonnées
72Ad 20 7C C7    JSR C77C     copie PISTE et SECTEUR en 00/01 ou 02/03 selon b7 de F9
                              c'est à dire selon qu'il s'agit d'un cycle lecture ou écriture secteurs
```

Charge les secteurs en RAM (dans la limite de la place disponible)
ou les écrits sur la disquette cible (selon flag "lecture/écriture")

```
72Dd A5 F7       LDA F7
72Fd D0 02       BNE C733     décrémente F7/F8, le nombre de secteurs à transférer
731d C6 F8       DEC F8
733d C6 F7       DEC F7
```

C735d CE 04 C0 DEC C004 décrémente HH de RWBUF
C738d AE 04 C0 LDX C004
C73Bd E0 05 CPX #05 teste s'il atteint la limite inférieure (#05)
C73Dd F0 2B BEQ C76A si oui, continue en C76A (sauve si besoin les valeurs de Y en C08C et de F7/F8 en C08D/C08E)
C73Fd A5 F7 LDA F7 sinon,
C741d 05 F8 ORA F8 teste s'il reste des secteurs à transférer
C743d F0 24 BEQ C769 sinon, continue en C769 (CLC et RTS)
C745d 20 28 E2 JSR E228 si oui, ajuste Y = Y + 2 pour viser les coordonnées du prochain secteur 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
C748d C0 02 CPY #02 teste si Y = #02
C74Ad D0 03 BNE C74F sinon, saute l'instruction suivante, si oui...
C74Cd 20 7C C7 JSR C77C copie PISTE et SECTEUR en 00/01 ou 02/03 selon b7 de F9
c'est à dire selon qu'il s'agit d'un cycle lecture ou écriture secteurs
C74Fd 24 F9 BIT F9 teste si le b7 de F9 est à 1 (écriture)
C751d 30 05 BMI C758 si oui, saute les 2 instructions suivantes
C753d 20 50 E2 JSR E250 sinon, lit les coordonnées du prochain secteur à charger
et le charge selon DRIVE PISTE SECTEUR et RWBUF
C756d F0 D5 BEQ C72D reprend en C72D si pas d'erreur
C758d B9 00 C1 LDA C100,Y mise à jour du n° de PISTE active
C75Bd 8D 01 C0 STA C001 et du n° de SECTEUR actif
C75Ed B9 01 C1 LDA C101,Y selon les valeurs lues dans la
C761d 8D 02 C0 STA C002 liste des secteurs à charger
C764d 20 A4 DA JSR DAA4 XSVSEC écrit un secteur selon DRIVE, PISTE, SECTEUR et RWBUF
C767d F0 C4 BEQ C72D reprend en C72D si pas d'erreur

Il n'y a plus de secteurs à transférer
C769d 18 CLC flag pour indiquer qu'il n'y en a plus

Sauve si nécessaire Y et F7/F8 et retourne
C76Ad 24 F9 BIT F9 teste si le b7 de F9 est à zéro
C76Cd 10 0D BPL C77B si oui, simple RTS en C77B, sinon...
C76Ed 8C 8C C0 STY C08C sauve Y en C08C (position dans liste coordonnées)
C771d A5 F7 LDA F7 et F7/F8 en C08D/C08E
C773d A4 F8 LDY F8 (nombre de secteurs restant à charger)
C775d 8D 8D C0 STA C08D
C778d 8C 8E C0 STY C08E
C77Bd 60 RTS

Sauve les coordonnées du descripteur en cours en 00/01 ou 02/03 selon le flag "lecture/écriture"
C77Cd AD 01 C0 LDA C001 n° de PISTE active
C77Fd AE 02 C0 LDX C002 n° de SECTEUR actif
C782d 24 F9 BIT F9 teste si le b7 de F9 est à 1 (cycle écriture)
C784d 30 05 BMI C78B si oui, continue en C78B
C786d 85 00 STA 00 si le b7 est à zéro, sauve PISTE en 00
C788d 86 01 STX 01 et SECTEUR en 01 (pour lecture secteur)
C78Ad 60 RTS
C78Bd 85 02 STA 02 si le b7 est à 1, sauve PISTE en 02
C78Dd 86 03 STX 03 et SECTEUR en 03 (pour écriture secteur)
C78Fd 60 RTS

Messages externes de la banque n°4 (C790 à C7FF)
C790d 4C 4F 41 44 20 53 4F 55 52 43 **C5**
01 LOAD SOURC**E**
C79Bd 4C 4F 41 44 20 54 41 52 47 45 **D4**
02 LOAD TARGE**T**
C7A6d **BF**
03 **?**
C7A7d 4C 4F 41 44 20 44 49 53 43 53 20 46 4F 52 20 43 4F 50 59 20 46 52 4F 4D **A0**
04 LOAD DISCS FOR COPY FROM■
C7C0d 20 54 4F **A0**
05 ■TO■
C7C4d 0A 0D 43 6F 70 79 20 63 6F 6D 70 6C 65 74 65 0A **8D**
06 LFCRCopy completeLF**CR**
C7D5d 20 4F 56 45 52 57 52 49 54 54 45 4E 0A **8D**
07 ■OVERWRITTENLF**CR**
C7E3d 20 41 50 50 45 4E 44 45 44 0A **8D**
08 ■APPENDEDLF**CR**
C7EEd 20 43 52 45 41 54 45 44 0A **8D**
09 ■CREATEDLF**CR**

```
7F8d  C6 0C                 le reste semble n'avoir aucun sens...
7FAd  D0 F5     BNE C7F1    ce n'est pas une adresse de branchement
7FCd  F0 E5     BEQ C7E3    idem
7FEd  60        RTS
7FFd  00        BRK
```

Banque n°5 (adresse Cxxxe):
SYS DNAME DTRACK TRACK INIST DNUM DSYS DKEY VUSER

Cette banque se trouve à partir du #56 (86ème) secteur de la disquette MASTER

C400e	00 00	EXTER	adresse des messages d'erreur externe (néant)
C402e	F0 C6	EXTMS	adresse des messages externes
C404e	4C 5A C5	JMP C55A	Entrée commande SYS
C407e	4C 1F C4	JMP C41F	Entrée commande DNAME
C40Ae	4C 3E C4	JMP C43E	Entrée commande DTRACK
C40De	4C 46 C4	JMP C446	Entrée commande TRACK
C410e	4C 09 C5	JMP C509	Entrée commande INIST
C413e	4C D8 C4	JMP C4D8	Entrée commande DNUM
C416e	4C 22 C5	JMP C522	Entrée commande DSYS
C419e	4C FD C5	JMP C5FD	Entrée commande DKEY
C41Ce	4C 2A C6	JMP C62A	Entrée commande VUSER

EXECUTION COMMANDE SEDORIC DNAME

Rappel de la syntaxe
DNAME (lecteur)
Edite le nom de la disquette présente dans le drive indiqué ou dans le drive courant.

Variables utilisées

D0	longueur de la chaîne dans la zone des chaînes sous HIMEM
D1/D2	adresse de la chaîne dans la zone des chaînes sous HIMEM
F2	longueur de la chaîne à saisir par XLINPU
F3	options E, S, C, J, K pour XLINPU
F3/F4	adresse de lecture dans BUF1 (secteur système)
F4	mode de sortie de XLINPU
F8	on se demande bien à quoi il sert (utilisé par s/p C68E)
F9	position de la chaîne dans BUF1
C000/C004	DRIVE, PISTE, SECTEUR, RWBUF
C016	flag "banque changée"
C075	caractère à utiliser pour matérialiser la fenêtre XLINPU
C100/C1FF	BUF1

Informations non documentées
La chaîne ne peut comporter que 21 caractères au maximum. Ces caractères peuvent être quelconques, y compris des attributs vidéo que l'on peut entrer avec CTRL/Z puis une lettre de @ à W.

Utilisation en LM
Si la banque n°5 est déjà en place, et que l'on veut opérer sur le drive courant, un simple JSR F145 (précédé d'un passage sous RAMOV) suffit. Sinon, il faut écrire la lettre désignant le lecteur (A à D) dans le tampon clavier, initialiser TXTPTR puis faire le JSR F148 (voir les détails en annexe).

Affichage du nom actuel

C41Fe	20 8E C6	JSR C68E	valide drive et affiche "LFCRDisc name:" suivi du nom de la disquette (complété avec des espaces si besoin pour faire 21 caractères)
C422e	A2 15	LDX #15	X = 21
C424e	20 69 EE	JSR EE69	affiche X fois "flèche gauche" (retourne au début)
C427e	A9 15	LDA #15	A = 21 caractères à saisir
C429e	A0 88	LDY #88	Y = 1000 1000 = options ",S" et ",E" de LINPUT, c'est à dire interdit toute sortie avec les flèches et interdit l'affichage initial du caractère de remplissage afin de ne pas effacer le nom affiché

Saisie du nouveau nom

C42Be	A2 09	LDX #09	X = 9 = position où écrire de DNAME dans BUF1
C42De	20 D6 C5	JSR C5D6	Saisit la chaîne et la copie dans BUF1
C430e	A4 F4	LDY F4	teste si mode de sortie = 1
C432e	88	DEY	c'est à dire ESC
C433e	F0 06	BEQ C43B	si oui, continue en C43B
C435e	20 06 D2	JSR D206	CBF0/ROM va à la ligne
C438e	20 A4 DA	JSR DAA4	XSVSEC écrit un secteur selon DRIVE, PISTE, SECTEUR et RWBUF
C43Be	4C 06 D2	JMP D206	CBF0/ROM va à la ligne

EXECUTION COMMANDE SEDORIC DTRACK

Rappel de la syntaxe

DTRACK (lecteur) (,PA(;F))(,PB(;F))(,PC(;F))(,PD(;F))

Avec PA = nombre de pistes/face pour le drive A (de 0 à 99, 0 étant utilisé pour indiquer que le lecteur n'est pas connecté), PB idem pour le drive B etc... et F = "S" ou "D" selon qu'il s'agit d'un lecteur à Simple ou à Double tête. Modifie la configuration des lecteurs A à D sur la disquette présente dans le drive indiqué ou à défaut, présente dans le drive courant. Les paramètres de certains drives peuvent être omis (s'il n'y a pas besoin de les modifier), mais il faut quand même mettre les virgules si l'on veut modifier le ou les drives suivants: DTRACK,,42 modifie seulement la configuration du drive B.

Variables utilisées

2		index pour écrire dans BUF1 (selon le n° du drive)
7		b7 à 0 pour ";S" et à 1 pour ";D"
9		LL totalisateur pour calcul nombre de secteurs/face
016		flag "banque changée"
072		flag TRACK/DTRACK (b7 à 0 si TRACK à 1 si DTRACK)
100/C1FF	BUF1	pour charger le secteur système
200/C2FF	BUF2	pour charger la bitmap

Informations non documentées

La commande DTRACK sans paramètres reste sans effet à part charger la banque n°5, ce qui peut être interessant pour une utilisation dans un programme en LM des commandes présentes dans cette banque.

Le ";S" ne sert à rien, car il est pris de toute façon par défaut de plus ";X" (ou tout autre caractère sauf ";D") fait le même effet!

Enfin, notons que la vérification de la validité du nombre de secteurs par face indiqué comme paramètre est plus que farfelue!

Le manuel (page 42) n'indique pas à quoi servent les commande TRACK et DTRACK. Cependant, on apprend page 36, qu'en absence d'indication du nombre de pistes par face et du nombre de faces, la commande INIT prend par défaut les valeurs présentes en mémoire et modifiables à l'aide de la commande TRACK (et donc aussi par DTRACK). Ces valeurs peuvent être connues à l'aide des commandes SYS et DSYS. En fait, les commandes TRACK et DTRACK ne servent pas à grand chose. En effet, INIT se contente déjà de prendre 17 comme nombre de secteurs par piste par défaut et pourrait aussi prendre 42 pistes par face et simple face. Personne ne fait confiance à ces 2 valeurs par défaut présentes en mémoire, car elles dépendent de la disquette utilisée au démarrage, disquette bien souvent quelconque. Pour se risquer à utiliser INIT sans indiquer ces valeurs, il faudrait d'abord faire un SYS, puis éventuellement un TRACK dont la syntaxe est pénible! Il est bien plus simple de taper directement INIT A,17,42,D. Nous pensons donc qu'il faudrait modifier la commande INIT pour quelle utilise 42 et S par défaut au lieu des valeurs stockées dans la TABDRV en C039/C03C. Il serait ainsi possible de récupérer la place dégagée par la supression des commandes TRACK et DTRACK.

Utilisation en LM

Etant donné la complexité de la syntaxe, il semble raisonnable d'écrire les paramètres dans le tampon clavier, d'initialiser TXTPTR puis de faire un JSR F139 (voir les détails en annexe).

Saisie du paramètre "lecteur" et initialise flag "DTRACK"

43Ee	20 0D E6	JSR E60D	valide drive indiqué à TXTPTR, sinon valide DRVDEF
441e	20 DB C6	JSR C6DB	demande la disquette cible
444e	38	SEC	C = 1 = flag DTRACK
445e	24 18	BIT 18	et continue en C447

EXECUTION COMMANDE SEDORIC TRACK

Rappel de la syntaxe

TRACK (PA(;F))(,PB(;F))(,PC(;F))(,PD(;F))

Avec PA = nombre de pistes/face pour le drive A (de 0 à 99, 0 étant utilisé pour indiquer que le lecteur n'esst pas connecté), PB idem pour le drive B etc... et F = "S" ou "D" selon qu'il s'agit d'un lecteur à Simple ou à Double tête. Modifie en mémoire la configuration des lecteurs A à D. Les paramètres de certains drives peuvent être omis (s'il n'y a pas besoin de les modifier), mais il faut quand même mettre les virgules si l'on veut modifier le ou les drives suivants: TRACK,,42 modifie seulement la configuration du drive C. NB: Il y a une erreur dans le manuel page 42 à propos de la signification de ";S".

Variables utilisées

2		index pour écrire dans BUF1 (selon le n° du drive)
7		b7 à 0 pour ";S" et à 1 pour ";D"
9		LL totalisateur pour calcul nombre de secteurs/face
039/C03C	TABDRV	table de configuration des lecteurs
072		flag TRACK/DTRACK (b7 à 0 si TRACK à 1 si DTRACK)

Informations non documentées

La commande TRACK sans paramètres reste sans effet à part charger la banque n°5, ce qui peut être interessant pour une utilisation dans un programme en LM des commandes présentes dans cette banque.

Le ";S" ne sert à rien, car il est pris de toute façon par défaut de plus ";X" (ou autre sauf ";D") fait le même effet! Attention, bogue usuelle: le "d" en minuscule ne sera pas pris en compte et déclenchera une belle "SYNTAX ERROR".

Enfin, notons que la vérification de la validité du nombre de secteurs par face indiqué comme paramètre est plus que farfelue!
Voir les "Informations non documentées" de la commande DTRACK concernant l'utilité des commandes TRACK et DTRACK.

Utilisation en LM
Etant donné la complexité de la syntaxe, il semble raisonnable d'écrire les paramètres dans le tampon clavier, d'initialiser TXTPTR puis de faire un JSR F130 (voir les détails en annexe).

C446e	18	CLC	C = 0 = flag "TRACK"
C447e	A0 00	LDY #00	
C449e	84 F2	STY F2	F2 = 0 index pour écrire dans BUF1
C44Be	AA	TAX	teste s'il y a des paramètres
C44Ce	F0 52	BEQ C4A0	si pas de paramètre, simple RTS
	Suite de l'analyse de syntaxe et saisie des paramètres		
C44Ee	6E 72 C0	ROR C072	copie C dans b7 de C072 (0 si TRACK, 1 si DTRACK)
C451e	10 09	BPL C45C	continue en C45C si TRACK, sinon...
C453e	20 4C DA	JSR DA4C	XPMAP prend le secteur de bitmap, vérifie le format
C456e	20 D4 C6	JSR C6D4	prend le secteur système dans BUF1
C459e	20 2C D2	JSR D22C	D067/ROM exige une "," place TXTPTR sur l'octet suivant
C45Ce	C9 2C	CMP #2C	le caractère suivant est-il une ","? (caractère omis)
C45Ee	F0 2B	BEQ C48B	si oui, continue en C48B
C460e	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 pistes par face)
C463e	C9 3B	CMP #3B	le caractère suivant est-il un ";"? (annonce une indication de lecteur double face)
C465e	D0 0B	BNE C472	sinon, continue en C472, si oui...
C467e	20 98 D3	JSR D398	XCRGET lit le caractère suivant à TXTPTR et le convertit en MAJUSCULE
C46Ae	A9 44	LDA #44	caractère "D"
C46Ce	20 2E D2	JSR D22E	JSR D067/ROM et D3A1/RAMOV demande "D" à TXTPTR, lit le caractère suivant à TXTPTR et le convertit en MAJUSCULE
C46Fe	A9 80	LDA #80	pour double face
C471e	2C A9 00	BIT 00A9	continue en C474
C472e	A9 00	LDA #00	pour simple face dans tous les autres cas
C474e	85 F7	STA F7	sauve A dans b7 de F7 (0 = simple, 1 = double face)
C476e	8A	TXA	et reprend X dans A (nombre de pistes par face)
C477e	20 A1 C4	JSR C4A1	vérifie si le nombre de secteurs par piste et le nombre de secteurs par disquette sont corrects (enfin "essaye" de vérifier!)
C47Ae	F0 02	BEQ C47E	saute l'instruction suivante si le nombre de pistes par face est nul (unconnected drive)
C47Ce	05 F7	ORA F7	force b7 de A selon b7 de F7 (1 = double face)
C47Ee	A4 F2	LDY F2	index d'écriture
C480e	99 00 C1 Y	STA C100,Y	écrit le nombre de pistes/face avec l'indice du nombre de faces (b7) dans BUF1 à la position
C483e	2C 72 C0	BIT C072	teste b7 de C072 (0 = TRACK, 1 = DTRACK)... c'est un peu tard, le STA C100,Y n'était peut être pas nécessaire!
C486e	30 03	BMI C48B	si DTRACK, saute l'instruction suivante
C488e	99 39 C0	STA C039,Y	si TRACK, écrit nombre de pistes/face dans TABDRV
	Teste s'il y a encore des drives à configurer		
C48Be	E6 F2	INC F2	indexe le drive suivant
C48De	A5 F2	LDA F2	
C48Fe	C9 04	CMP #04	teste si A >= 4
C491e	B0 05	BCS C498	si oui (fini), continue en C498, sinon...
C493e	20 9E D3	JSR D39E	XCRGOT relit le caractère à TXTPTR et le convertit en MAJUSCULE
C496e	D0 C1	BNE C459	si pas fin des paramètres, reboucle en C459
C498e	2C 72 C0	BIT C072	teste b7 de C072 (0 = TRACK, 1 = DTRACK)
C49Be	10 03	BPL C4A0	simple RTS si TRACK, sinon...
C49De	4C A4 DA	JMP DAA4	XSVSEC écrit un secteur selon DRIVE, PISTE, SECTEUR et RWBUF
C4A0e	60	RTS	
	Vérifie si les nombres de secteurs par piste et par disquette sont corrects		
C4A1e	48	PHA	sauve A sur la pile (nombre de pistes par face)
C4A2e	A8	TAY	teste si A est nul ("unconnected")
C4A3e	F0 2E	BEQ C4D3	si oui, continue en C4D3 (dépile A et retourne)
C4A5e	A0 00	LDY #00	sinon, force Y à 0 (initialise HH du totalisateur)
C4A7e	C9 15	CMP #15	teste si A < 21 pistes
C4A9e	90 2A	BCC C4D5	si oui, continue en C4D5 ("ILLEGAL QUANTITY ERROR")

```
4ABe  C9 64      CMP #64      teste si A >= 100 pistes
4ADe  B0 26      BCS C4D5     si oui, continue en C4D5 ("ILLEGAL QUANTITY ERROR")
4AFe  A9 00      LDA #00
4B1e  85 F9      STA F9       force F9 à 0 (initialise LL du totalisateur)
4B3e  CA         DEX          décrémente le nombre de pistes par face (seule variable dans ce qui suit) rebouclera donc
```
autant de fois qu'il y a de pistes par face. Terminera avec un pseudo nombre de secteurs par face basé sur 17 secteurs par piste. Le minimum possible de secteurs par face est 21 pistes x 16 secteurs/piste = 336. Le maximum possible est de 99 pistes x 19 secteurs/piste = 1881 secteurs/face. Le calcul se base a priori sur 17 secteurs par piste et donne donc un résultat compris entre 357 et 1683.

```
4B4e  30 0C      BMI C4C2     si X < #00, continue en C4C2
4B6e  A9 11      LDA #11      sinon A = 17 secteurs par piste d'office!
4B8e  18         CLC          prépare une addition
4B9e  65 F9      ADC F9
4BBe  85 F9      STA F9       F9 = F9 + #11
4BDe  90 F4      BCC C4B3     reboucle en C4B3 tant que total < #100 (F9 et A représentent le LL du résultat)
4BFe  C8         INY          sinon, incrémente Y (représente le HH du résultat)
4C0e  D0 F1      BNE C4B3     reprise forcée en C4B3 (HH jamais nul)
4C2e  24 F7      BIT F7       teste le b7 de F7 (à 1 si Double face)
4C4e  10 06      BPL C4CC     si b7 = 0 (Simple face), continue en C4CC
4C6e  0A         ASL          si b7 = 1 (Double face), A = A x 2 (avec retenue dans C)
4C7e  48         PHA          et empile le résultat
4C8e  98         TYA
4C9e  2A         ROL          multiplie Y par 2 en récupérant C dans b0
4CAe  A8         TAY
4CBe  68         PLA          récupère A (LL du résultat sur deux octets)
```
A ce stade, AY est le nombre total T de secteurs formatés de la disquette. Le résultat obtenu peut donc aller de 714 à 3366. Le minimum ne pose pas de problème, mais le maximum risque de dépasser la barre fatidique des 1919 secteurs (erreur dans le manuel page 37 à propos de INIT et du nombre maximum de secteurs).

```
4CCe  C0 07      CPY #07      teste si Y < 7 (et donc T < 1792). Si HH < 7, le nombre de secteurs est au maximum de #6FF
```
soit 1791. Si HH = 7, le nombre de secteurs peut atteindre #7FF soit 2047, ce qui est trop. Dans ce cas, on teste le b7 du LL. S'il est à 1, le nombre de secteurs dépasse #77F soit 1919, ce qui est trop. On voit que ce test est falacieux:
1) par l'a priori de 17 secteurs par piste et
2) parce que si HH > 7 et que le b7 du LL est à 0, on dépasse, mais ce n'est pas détecté (exemple 2048 = #800). On pourrait carrement supprimer ce test en mettant un RTS en C4AF ce qui permettrait de récupérer 37 octets pour un meilleur usage.

```
4CEe  90 03      BCC C4D3     si oui, continue en C4D3 (dépile et RTS)
4D0e  AA         TAX          teste si b7 de A est à 1 (T >= 1920)
4D1e  30 02      BMI C4D5     si oui, "ILLEGAL QUANTITY ERROR"
4D3e  68         PLA          récupère A (nombre de pistes par face)
4D4e  60         RTS          et retourne

4D5e  4C 20 DE   JMP DE20     "ILLEGAL QUANTITY ERROR"
```

Version 2.0 GB

Le sous-programme C4B3 à été modifié (31 octets différents) pour tenir compte de la nouvelle capicité maximum des disquettes Sédoric: 3840 secteurs par disquette. Hélas cette vérification n'était pas nécessaire puisque les nombres maximum de secteurs par piste (19) et de piste par face (99) ont déjà été vérifiés. Or 19 x 99 x 2 = 3762 qui est donc toujours inférieur aux 3840 secteurs autorisés!

```
4B3e  CA         DEX          début de la boucle de calcul du nombre de secteurs par face basé sur 17 secteurs par piste.
```
Le totalisateur n'est plus F9 (LL) et Y (HH) mais A (LL) et F9 (HH).

```
4B4e  30 09      BMI C4BF     si calcul pour 1 face terminé, continue en C4C2
4B6e  18         CLC          prépare une addition
4B7e  69 11      ADC #11      17 secteurs par piste toujours d'office!
4B9e  90 02      BCC C4BD     saute l'instruction suivante si pas de retenue
4BBe  E6 F9      INC F9       HH = HH + 1 report de la retenue
4BDe  D0 F4      BNE C4B3     reboucle en tant que total < #100
4BFe  24 F7      BIT F7       teste le b7 de F7 (à 1 si Double face)
4C1e  10 03      BPL C4C6     si Simple face, saute les 2 instructions suivantes
4C3e  0A         ASL          si Double face, A = A x 2 (avec retenue dans C)
4C4e  26 F9      ROL F9       reportée dans F9, lui aussi multiplié par 2
4C6e  A4 F9      LDY F9       pour tester si HH < #0F
4C8e  C0 0F      CPY #0F      c'est à dire nombre total de secteurs < 3840
4CAe  90 05      BCC C4D1     si oui OK, continue en C4D1
4CCe  D0 05      BNE C4D3     si > 3840, "ILLEGAL QUANTITY ERROR" en C4D3
4CEe  AA         TAX          si HH = #0F, vérifie que LL = #00
4CFe  D0 02      BNE C4D3     si ce n'est pas le cas, "ILLEGAL QUANTITY ERROR"
4D1e  68         PLA          récupère A (nombre de pistes par face)
4D2e  60         RTS          et retourne
4D3e  4C 20 DE   JMP DE20     "ILLEGAL QUANTITY ERROR"
4D5e  20 DE      DE20         bogue: le saut à "ILLEGAL QUANTITY ERROR" appelé depuis C4A9 et C4AD ne
```
marche plus car il manque le C4 (JMP)

Plus que jamais, on pourrait carrement supprimer ce test non seulement falacieux, mais inutile et bogué en mettant un RTS en C4AF ce qui permettrait de récupérer de la place pour un meilleur usage.

EXECUTION COMMANDE SEDORIC DNUM

Rappel de la syntaxe

DNUM (lecteur)(,DEFNUM)(,DEFPAS)

Avec DEFNUM = n° de la première ligne par défaut et DEFPAS = "pas" d'incrémentation par défaut. Ces paramètres sont utilisés par RENUM et par la numérotation automatique avec FUNCT+RETURN. DNUM modifie ces valeurs par défaut sur la disquette présente dans le drive indiqué ou à défaut, dans le drive courant.

Variables utilisées

C000	DRIVE	n° de lecteur actif
C016		flag "banque changée"
C100/C1FF	BUF1	pour charger le secteur système

Informations non documentées

La commande DNUM sans paramètre reste sans effet à part charger la banque n°5, ce qui peut être interessant pour une utilisation dans un programme en LM des commandes présentes dans cette banque.

Si l'indication de lecteur est omise, il faut commencer par une virgule (exemple DNUM ,1000,10), voire deux si l'on veut modifier uniquement DEFPAS (exemple DNUM,,10)... sinon SYNTAX ERROR!

Utilisation en LM

Comme bien souvent, le plus simple est d'écrire les paramètres dans le tampon clavier, initialiser TXTPTR puis faire un JSR F12A (voir les détails en annexe).

Analyse de la syntaxe et saisie des paramètres

C4D8e	20 0D E6	JSR E60D	valide drive indiqué à TXTPTR, sinon valide DRVDEF
C4DBe	F0 C3	BEQ C4A0	simple RTS si pas de paramètre
C4DDe	20 DB C6	JSR C6DB	demande la disquette cible
C4E0e	20 D4 C6	JSR C6D4	copie le secteur système dans BUF1
C4E3e	20 2C D2	JSR D22C	D067/ROM exige une "," place TXTPTR sur l'octet suivant
C4E6e	C9 2C	CMP #2C	le caractère suivant est-il une ","? (un paramètre omis)
C4E8e	F0 0E	BEQ C4F8	si oui, continue en C4F8, sinon...
C4EAe	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)
C4EDe	8C 05 C1	STY C105	écrit ce nombre dans BUF1, aux positions #05 et #06
C4F0e	8D 06 C1	STA C106	(DEFNUM = départ de RENUM par défaut)
C4F3e	20 9E D3	JSR D39E	XCRGOT relit le caractère à TXTPTR et le convertit en MAJUSCULE
C4F6e	F0 0E	BEQ C506	termine en C506 s'il n'y a plus de paramètre
C4F8e	20 2C D2	JSR D22C	D067/ROM exige une "," place TXTPTR sur l'octet suivant
C4FBe	F0 09	BEQ C506	termine en C506 s'il n'y a plus de paramètre
C4FDe	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)
C500e	8C 07 C1	STY C107	écrit ce nombre dans BUF1, aux positions #07 et #08
C503e	8D 08 C1	STA C108	(DEFPAS = "PAS" de RENUM par défaut)
C506e	4C A4 DA	JMP DAA4	XSVSEC écrit un secteur selon DRIVE, PISTE, SECTEUR et RWBUF

EXECUTION COMMANDE SEDORIC INIST

Rappel de la syntaxe

INIST (lecteur)

Edite les instructions à exécuter lors du boot et se trouvant déjà sur la disquette présente dans le drive indiqué ou, à défaut, dans le drive courant.

Variables utilisées

D0	longueur de la chaîne dans la zone des chaînes sous HIMEM
D1/D2	adresse de la chaîne dans la zone des chaînes sous HIMEM
F2	longueur de la chaîne à saisir par XLINPU
F3	options E, S, C, J, K pour XLINPU
F3/F4	adresse de lecture dans BUF1 (secteur système)
F4	mode de sortie de XLINPU
F8	on se demande bien à quoi il sert (utilisé par s/p C690)
F9	position de la chaîne dans BUF1
C000/C004	DRIVE, PISTE, SECTEUR, RWBUF
C016	flag "banque changée"
C075	caractère à utiliser pour matérialiser la fenêtre
C100/C1FF	BUF1

Informations non documentées
La chaîne ne peut comporter que 60 caractères au maximum. Curieusement l'exemple donné dans le manuel laisse penser qu'on peut utiliser une syntaxe du type INIST (lecteur)(,chaîne alphanumérique de commandes). Il n'en est rien, la saisie de la chaîne se fait dans un deuxième temps à l'aide d'un masque de type LINPUT. Dans les limites de la syntaxe BASIC et Sédoric, ces caractères peuvent être quelconques, y compris des attributs vidéo que l'on peut entrer avec CTRL/Z puis une lettre de @ à W.

Utilisation en LM
Si la banque n°5 est déjà en place, et que l'on veut opérer sur le drive courant, un simple JSR F12D (précédé d'un passage sous RAMOV) suffit. Il est possible de mettre la banque 5 en place par un JSR F139 sous RAMOV. Sinon, il faut écrire la lettre désignant le lecteur (A à D) dans le tampon clavier, initialiser TXTPTR puis faire le JSR F12D (voir les détails en annexe).

Affichage de la liste actuelle des commandes

509e	20 90 C6	JSR C690	valide le drive indiqué ou le drive par défaut et affiche "LFCRInit statement:" suivi des instructions de démarage (chaîne complètée à 60 caractères avec des espaces)
50Ce	A2 3C	LDX #3C	X = 60
50Ee	20 69 EE	JSR EE69	affiche X fois "flèche gauche" (retourne au début)
511e	A9 3C	LDA #3C	A = 60 (nombre de caractères à saisir)
513e	A0 88	LDY #88	Y = 1000 1000 = options ",S" et ",E" de LINPUT, c'est à dire interdit toute sortie avec les flèches et interdit l'affichage initial du caractère de remplissage afin de ne pas effacer la chaîne affichée
515e	A2 1E	LDX #1E	X = 30 (position de INIST dans BUF1)
517e	20 D6 C5	JSR C5D6	saisit une chaîne et la copie dans BUF1
51Ae	A4 F4	LDY F4	teste si le mode de sortie est 1
51Ce	88	DEY	c'est à dire sortie par ESC
51De	F0 B5	BEQ C4D4	si oui, continue en C4D4 (simple RTS)
51Fe	4C A4 DA	JMP DAA4	XSVSEC écrit un secteur selon DRIVE, PISTE, SECTEUR et RWBUF

EXECUTION COMMANDE SEDORIC DSYS

Rappel de la syntaxe
DSYS (lecteur)
Affiche la configuration complète de la disquette présente dans le drive indiqué ou, à défaut, dans le drive courant: nom de la disquette DNAME, instruction de démarage INIST, nombre de pistes par face et nombre de faces pour chaque lecteur ou lecteurs non connectés selon TABDRV, valeurs de DEFNUM et DEFPAS, type de clavier au démarage (ACCENT SET/OFF et AZ/QWERTY).

Variables utilisées

0	longueur de la chaîne dans la zone des chaînes sous HIMEM
1/D2	adresse de la chaîne dans la zone des chaînes sous HIMEM
2	longueur de la chaîne à saisir par XLINPU
3/F4	adresse de lecture dans BUF1 (secteur système)
4	mode de sortie de XLINPU
7	n° du lecteur
8	on se demande bien à quoi il sert (utilisé par s/p C68E)
3/F9	adresse de la table des lecteurs TABDRV
9	position de la chaîne dans BUF1
000/C004	DRIVE, PISTE, SECTEUR, RWBUF
016	flag "banque changée"
04C	DEFAFF, code ASCII devant les nombres décimaux
100/C1FF	BUF1 pour charger le secteur système

Informations non documentées
Néant

Utilisation en LM
Si la banque n°5 est déjà en place, et que l'on veut opérer sur le drive courant, un simple JSR F127 (précédé d'un passage sous RAMOV) suffit. Il est possible de mettre la banque 5 en place par un JSR F139 sous RAMOV. Sinon, il faut écrire la lettre désignant le lecteur (A à D) dans le tampon clavier, initialiser TXTPTR puis faire le JSR F127 (voir les détails en annexe).

Affichage du nom de la disquette DNAME

522e	20 8E C6	JSR C68E	valide le drive indiqué ou le drive courant, charge le secteur système dans BUF1 et affiche "LFCRDisc name:" suivi du nom de la disquette (21 caractères)
525e	20 06 D2	JSR D206	CBF0/ROM va à la ligne

Affichage des instructions de démarage INIST

528e	20 8A C6	JSR C68A	re-valide le drive courant, re-charge le secteur système dans BUF1 et affiche "LFCRInit statement:" suivi des instructions de démarage (chaîne complètée à 60 caractères avec des espaces)

```
C52Be  20 06 D2   JSR D206    CBF0/ROM va à la ligne
```

Affiche la configuration des lecteurs, DEFNUM et DEFPAS

```
C52Ee  A9 00      LDA #00     AY = C100 début du secteur système chargé dans BUF1
C530e  A0 C1      LDY #C1     c'est à dire la table des drives TABDRV: nombre de pistes par face ou 0 si "unconnected"
C532e  20 5E C5   JSR C55E    affiche "Drive A:" suivi de "unconnected" ou du nombre de pistes par face et si "single
```

sided" ou "double sided". Idem pour B, C, et D. Affiche "Num origin:" suivi de la valeur DEFNUM; "Num step:" suivi de la valeur DEFPAS

Affiche la configuration du clavier

```
C535e  A2 05      LDX #05     indexe le message: "LFCRKeyboard:"
C537e  20 64 D3   JSR D364    XAFSC affiche X+1ème message externe terminé par "caractère + 128"
C53Ae  A2 0C      LDX #0C     indexe le message: "ACCENT "
C53Ce  20 64 D3   JSR D364    XAFSC affiche X+1ème message externe terminé par "caractère + 128"
C53Fe  A2 0D      LDX #0D     indexe le message: "SET,"
C541e  2C 04 C1   BIT C104    teste si le b6 de "type de clavier" est à 1
C544e  70 01      BVS C547    si oui (ACCENTSET), saute l'instruction suivante
C546e  E8         INX         indexe le message: "OFF,"
C547e  20 64 D3   JSR D364    XAFSC affiche X+1ème message externe terminé par "caractère + 128"
C54Ae  A2 0F      LDX #0F     indexe le message: "AZ"
C54Ce  2C 04 C1   BIT C104    teste si le b7 de "type de clavier" est à 1
C54Fe  30 01      BMI C552    si oui (AZERTY), saute l'instruction suivante
C551e  E8         INX         indexe le message: "QW"
C552e  20 64 D3   JSR D364    XAFSC affiche X+1ème message externe terminé par "caractère + 128"
C555e  A2 11      LDX #11     indexe le message: "ERTYLFCR"
C557e  4C 64 D3   JMP D364    XAFSC affiche X+1ème message externe terminé par "caractère + 128"
```

EXECUTION COMMANDE SEDORIC SYS

Rappel de la syntaxe

SYS tout court

Affiche une partie de la configuration du Sédoric présent en RAMOV: lecteurs non connectés ou nombre de pistes par face et nombre de faces pour chaque lecteur selon TABDRV, valeurs de DEFNUM et DEFPAS.

Variables utilisées

```
F7          n° du lecteur, sert d'index dans TABDRV
F8/F9       adresse de la table des lecteurs TABDRV
C039        TABDRV
C04C        DEFAFF, code ASCII devant les nombres décimaux
```

Informations non documentées

Cette commande aurait aussi bien pu afficher le type de clavier en cours (ACCENT SET/OFF et AZ/QWERTY) comme le fait la commande DSYS!

Utilisation en LM

Simple: on passe sous RAMOV et on fait un JSR F15A (voir annexe).

Affiche la configuration courante des lecteurs

```
C55Ae  A9 39      LDA #39     F8/F9 = C039 (TABDRV en RAMOV si l'on est entré par
C55Ce  A0 C0      LDY #C0     SYS ou F8/F9 = C100 (TABDRV du secteur système si
C55Ee  85 F8      STA F8      l'on vient de DSYS)
C560e  84 F9      STY F9
C562e  A9 30      LDA #30     A = "0"
C564e  8D 4C C0   STA C04C    DEFAFF, code ASCII devant les nombres décimaux
C567e  A0 00      LDY #00     Y = lecteur A
C569e  84 F7      STY F7      sauve le numéro de lecteur
C56Be  A2 06      LDX #06     indexe le message: "LFCRDrive "
C56De  20 64 D3   JSR D364    XAFSC affiche X+1ème message externe terminé par "caractère + 128"
C570e  A5 F7      LDA F7      numéro de lecteur
C572e  20 0E D6   JSR D60E    convertit n° lecteur en lettre et l'affiche
C575e  A9 3A      LDA #3A     A = ":"
C577e  20 2A D6   JSR D62A    XAFCAR affiche le caractère ASCII contenu dans A
C57Ae  20 28 D6   JSR D628    affiche un espace
C57De  A4 F7      LDY F7      numéro de lecteur = index de lecture
C57Fe  B1 F8      LDA (F8),Y  lit octet à l'adresse indiquée en F8/F9 + Y
C581e  D0 07      BNE C58A    si "connected", continue en C58A
C583e  A2 0B      LDX #0B     si "unconnected", indexe le message: "unconnected "
C585e  20 64 D3   JSR D364    XAFSC affiche X+1ème message externe terminé par "caractère + 128"
C588e  30 1B      BMI C5A5    suite forcée en C5A5
C58Ae  48         PHA         sauve A (valeur lue dont b7 à 1 si Double face)
```

```
58Be  29 7F      AND #7F      masque 0111 1111, force b7 à zéro
58De  20 4E D7   JSR D74E     affichage en décimal sur 2 digits d'un nombre A
590e  A2 07      LDX #07      indexe le message " tracks "
592e  20 64 D3   JSR D364     XAFSC affiche X+1ème message externe terminé par "caractère + 128"
595e  68         PLA          récupère A
596e  30 03      BMI C59B     si b7=1 (Double face), continue en C59B
598e  A2 08      LDX #08      indexe le message "single "
59Ae  2C A2 09   BIT 09A2     continue en C59D
59Be  A2 09      LDX #09      indexe le message "double "
59De  20 64 D3   JSR D364     XAFSC affiche X+1ème message externe terminé par "caractère + 128"
5A0e  A2 0A      LDX #0A      indexe le message "sided "
5A2e  20 64 D3   JSR D364     XAFSC affiche X+1ème message externe terminé par "caractère + 128"
5A5e  A4 F7      LDY F7       numéro de lecteur
5A7e  C8         INY          lecteur suivant (numéro 3 = maximum)
5A8e  C0 04      CPY #04      teste si numéro de lecteur < 4
5AAe  90 BD      BCC C569     si oui, reboucle en C569

      Affiche les valeurs courantes de DEFNUM et DEFPAS
5ACe  A9 20      LDA #20      sinon, A = espace
5AEe  8D 4C C0   STA C04C     DEFAFF, code ASCII devant les nombres décimaux
5B1e  20 06 D2   JSR D206     CBF0/ROM va à la ligne
5B4e  A2 03      LDX #03      indexe le message "LFCRNum orign:"
5B6e  20 64 D3   JSR D364     XAFSC affiche X+1ème message externe terminé par "caractère + 128"
5B9e  A0 05      LDY #05      index de lecture
5BBe  20 CB C5   JSR C5CB     lit et affiche sur 5 digits la valeur de DEFNUM
5BEe  A2 02      LDX #02      indexe le message "LFCRNum step :"
5C0e  20 64 D3   JSR D364     XAFSC affiche X+1ème message externe terminé par "caractère + 128"
5C3e  A0 07      LDY #07      index de lecture
5C5e  20 CB C5   JSR C5CB     lit et affiche sur 5 digits la valeur de DEFPAS
5C8e  4C 06 D2   JMP D206     CBF0/ROM va à la ligne

      Lit et affiche en décimal sur 5 digits
5CBe  B1 F8      LDA (F8),Y   lit octet à l'adresse indiquée en F8/F9 + Y
5CDe  48         PHA          et l'empile
5CEe  C8         INY          octet suivant
5CFe  B1 F8      LDA (F8),Y   lit octet à l'adresse indiquée en F8/F9 + Y
5D1e  A8         TAY          et le passe dans Y
5D2e  68         PLA          récupère A
5D3e  4C 53 D7   JMP D753     affichage en décimal sur 5 digits d'un nombre AY

      Saisit une chaîne de A caractères et
      la copie à partir de la Xème position dans BUF1

      Variables utilisées
0                 longueur de la chaîne dans la zone des chaînes sous HIMEM
1/D2              adresse de la chaîne dans la zone des chaînes sous HIMEM
2                 longueur de la chaîne à saisir
3                 options E, S, C, J, K pour XLINPU
4                 mode de sortie de XLINPU
9                 position de la chaîne dans BUF1
075               caractère à utiliser pour matérialiser la fenêtre

      Sous-programme commun à plusieurs commandes
5D6e  85 F2      STA F2       longueur de la chaîne à saisir
5D8e  86 F9      STX F9       position de la chaîne dans BUF1
5DAe  84 F3      STY F3       options E, S, C, J, K pour LINPUT
5DCe  A9 20      LDA #20      sauvegarde du caractère "espace" pour XLINPU
5DEe  8D 75 C0   STA C075     caractère à utiliser pour matérialiser la fenêtre
5E1e  20 36 ED   JSR ED36     XLINPU (routine de saisie de chaîne). Au retour, F4 contient le mode de sortie, D0 et
      D1/D2 donnent la longueur et l'adresse de la chaîne dans la zone de stockage des chaînes sous HIMEM.
5E4e  20 3E D7   JSR D73E     "CURSEUR ON" (curseur visible = vidéo inverse)
5E7e  A6 F4      LDX F4       mode de sortie de XLINPU
5E9e  CA         DEX          teste si mode de sortie est différent de 1 ("ESC")
5EAe  D0 01      BNE C5ED     si oui, saute l'instruction suivante
5ECe  60         RTS          simple RTS si "ESC"
5EDe  A0 00      LDY #00      Y = index de lecture dans la chaîne saisie
5EFe  A6 F9      LDX F9       X = index d'écriture dans BUF1
5F1e  B1 D1      LDA (D1),Y   lecture caractère dans la chaîne saisie
5F3e  9D 00 C1   STA C100,X   écriture dans BUF1
5F6e  E8         INX          suivant en écriture
5F7e  C8         INY          suivant en lecture
```

```
C5F8e  C4 F2     CPY F2      teste si fini (nombre copié = longueur chaîne)
C5FAe  D0 F5     BNE C5F1    sinon, reboucle en C5F1
C5FCe  60        RTS
```

EXECUTION COMMANDE SEDORIC DKEY

Rappel de la syntaxe
DKEY (lecteur)(,A)(,S)

Avec ",A" pour avoir un clavier AZERTY (sinon QWERTY par défaut) et ",S" pour avoir les caractères accentués (sinon ACCENT OFF par défaut).

DKEY modifie ces valeurs sur la disquette présente dans le drive indiqué ou à défaut, dans le drive courant. DKEY tout court ou DKEY (lecteur) imposent QWERTY et ACCENT OFF.

Variables utilisées

C000	DRIVE actif
C001	PISTE active
C002	SECTEUR actif
C003/C004	RWBUF
C016	flag "banque changée"
C100/C1FF	BUF1

Informations non documentées
DKEY,S,A est possible et même toutes autre combinaison telle que DKEY,S,A,S ou DKEY,S,S,S sans SYNTAX ERROR! Il faut le faire! Enfin, (bogue usuelle) le "d" en minuscule ne sera pas pris en compte et déclenchera une belle "SYNTAX ERROR", alors que "a" est accepté sans problème!

Utilisation en LM
Comme très souvent, le plus simple est d'écrire les paramètres dans le tampon clavier, initialiser TXTPTR puis faire un JSR F12A (voir les détails en annexe).

Analyse de la syntaxe et saisie des paramètres

```
C5FDe  20 CE C6   JSR C6CE    valide drive à TXTPTR ou valide DRVDEF, charge secteur système dans BUF1
C600e  20 DB C6   JSR C6DB    demande la disquette à modifier
C603e  A9 00      LDA #00     A = 0 (clavier QWERTY et ACCENT OFF par défaut)
C605e  F0 18      BEQ C61F    et suite forcée en C61F
C607e  20 2C D2   JSR D22C    D067/ROM exige une "," lit le caractère suivant et le convertit en MAJUSCULE
C60Ae  C9 41      CMP #41     est-ce un "A"?
C60Ce  D0 07      BNE C615    sinon, continue l'analyse de syntaxe en C615
C60Ee  20 98 D3   JSR D398    XCRGET lit le caractère suivant à TXTPTR et le convertit en MAJUSCULE
C611e  A9 80      LDA #80     masque 1000 0000 pour forcer AZERTY
C613e  D0 07      BNE C61C    et suite forcée en C61C
C615e  A9 53      LDA #53     caractère "S" (bogue usuelle: le "s" en minuscule ne sera pas pris en compte
                              et déclenchera une belle "SYNTAX ERROR")
C617e  20 2E D2   JSR D22E    JSR D067/ROM et D3A1/RAMOV demande "S" à TXTPTR,
                              lit le caractère suivant à TXTPTR et le convertit en MAJUSCULE
C61Ae  A9 40      LDA #40     masque 0100 0000 pour forcer ACCENT SET
C61Ce  0D 04 C1   ORA C104    force à 1 bits de C104 selon les bits à 1 de A
C61Fe  8D 04 C1   STA C104    type de clavier (b7=1 AZERTY, b6=1 ACCENT SET)
C622e  20 9E D3   JSR D39E    XCRGOT relit le caractère à TXTPTR et le convertit en MAJUSCULE
C625e  D0 E0      BNE C607    reboucle en C607 si fin des paramètres pas atteinte
C627e  4C A4 DA   JMP DAA4    si fin des paramètres atteinte, XSVSEC écrit un secteur selon DRIVE, PISTE, SECTEUR
                              et RWBUF et retourne
```

EXECUTION COMMANDE SEDORIC VUSER

Rappel de la syntaxe
VUSER tout court
Affiche les chaînes de définitions des 16 commandes redéfinissables utilisateurs stockées en RAMOV de C880 à C97F. Voir un exemple de l'utilisation de cette commande dans le "Non documenté" de la commande ACCENT.

Variables utilisées

12/13	adresse du début de la ligne à l'écran
F7	à 1 tant qu'aucun caractère significatif n'a pas été rencontré
F8	index de lecture dans la table des commandes redéfinissables
F9	n° de code de fonction de #00 à #0F
0269	n° de colonne TEXT = abscisse X du curseur
C04C	DEFAFF, code ASCII à mettre devant les nombres à afficher
C880/C97F	table des commandes redéfinissables

Informations non documentées
Un espace est affiché devant les codes de contrôle (ASCII inférieur à 32).

Utilisation en LM
Bon, ce coup-là, c'est vraiment simple: on bascule sur la RAMOV et on fait un JSR F121.

Entrée de la commande

62Ae	20 06 D2	JSR D206	CBF0/ROM va à la ligne
62De	A2 04	LDX #04	indexe le message "LFCRUser fonctions:LFCR"
62Fe	20 64 D3	JSR D364	XAFSC affiche X+1ème message externe terminé par "caractère + 128"
632e	A9 30	LDA #30	A = "0"
634e	8D 4C C0	STA C04C	DEFAFF, code ASCII devant les nombres décimaux
637e	A2 00	LDX #00	F9=0 compteur du nombre de commandes affichées
639e	86 F9	STX F9	c'est à dire le n° de code de fonction de #00 à #0F
63Be	86 F8	STX F8	F8=0 index lecture table commandes redéfinissables
63De	A9 80	LDA #80	au début de chaque ligne de commnde,
63Fe	85 F7	STA F7	force à 1 le b7 de F7, (reste à 1 tant qu'un caractère significatif n'a pas été rencontré)
641e	20 06 D2	JSR D206	CBF0/ROM va à la ligne
644e	A5 F9	LDA F9	A = n° de code de fonction de #00 à #0F
646e	20 4E D7	JSR D74E	affichage en décimal sur 2 digits d'un nombre A
649e	A9 3A	LDA #3A	A = ":"
64Be	20 2A D6	JSR D62A	XAFCAR affiche le caractère ASCII contenu dans A
64Ee	20 28 D6	JSR D628	affiche un espace pour séparer l'en-tête de la ligne de commande
651e	A6 F8	LDX F8	index lecture table commandes redéfinissables
653e	BD 80 C8	LDA C880,X	lit octet dans table commandes redéfinissables
656e	08	PHP	sauvegarde les indicateurs 6502 dont N (à1 si dernier caractère d'une ligne)
657e	29 7F	AND #7F	masque 0111 1111 pour forcer b7 à 0
659e	C9 20	CMP #20	est-ce un "espace"? (positionne C à 1 si A >= #20)
65Be	D0 04	BNE C661	sinon, continue en C661
65De	24 F7	BIT F7	si oui, teste si b7 de F7 est à 1 (mis à 1 au début de chaque ligne de commande)
65Fe	30 1B	BMI C67C	si oui, continue en C67C (saute l'affichage des 1er espaces)
661e	85 F7	STA F7	F7 reçoit le caractère dont le b7 à été mis à 0
663e	B0 14	BCS C679	si caractère affichable, continue en C679
665e	20 28 D6	JSR D628	si code de CTRL, affiche un espace (sera devant le code de CRTL)
668e	A5 F7	LDA F7	récupère le caractère
66Ae	09 40	ORA #40	masque 0100 0000 pour forcer b6 à 1 (conversion du code de CTRL de #00 à #1F en lettre de @ à £)
66Ce	20 2A D6	JSR D62A	XAFCAR affiche le caractère ASCII contenu dans A
66Fe	AC 69 02	LDY 0269	n° de colonne TEXT = abscisse X du curseur
672e	88	DEY	case précédente
673e	09 80	ORA #80	masque 1000 0000 pour forcer b7 à 1 (vidéo inverse)
675e	91 12	STA (12),Y	affiche caractère en vidéo inverse
677e	D0 03	BNE C67C	saut forcé de l'instruction suivante
679e	20 2A D6	JSR D62A	XAFCAR affiche le caractère ASCII contenu dans A
67Ce	28	PLP	récupère les indicateurs 6502 dont N
67De	30 03	BMI C682	continue en C682 si b7=1 (c'était le dernier caractère de la ligne)
67Fe	E8	INX	index de lecture table commandes redéfinissables
680e	D0 D1	BNE C653	reprise forcée en C653
682e	E6 F9	INC F9	nombre de lignes de commande affichées
684e	E8	INX	index de lecture table commandes redéfinissables
685e	D0 B4	BNE C63B	reboucle en C63B (commande suivante) tant que toute la table n'a pas été affichée, soit 256 octets
687e	4C 06 D2	JMP D206	CBF0/ROM va à la ligne

Affiche INIST

Variables utilisées par le s/p C68A

3/F4	adresse de lecture dans BUF1
3	on se demande bien à quoi il sert
016	flag "banque changée"
100/C1FF	BUF1

Sous-programme commun à plusieurs commandes

68Ae	18	CLC	C = 0 flag INITIAL STATEMENTS, "instructions au
68Be	08	PHP	démarrage" sauvegarde les indicateurs 6502 dont C
68Ce	90 11	BCC C69F	suite forcée en C69F

Affiche DNAME ou INIST

Variables utilisées par les s/p C68E et C690

3/F4	adresse de lecture dans BUF1

F8 on se demande bien à quoi il sert
C016 flag "banque changée"
C100/C1FF BUF1

Sous-programme commun à plusieurs commandes

C68Ee	38	SEC	C = 1 flag DISC NAME, "nom de la disquette"
C68Fe	24 18	BIT 18	continue en C691
C690e	18	CLC	C = 0 flag INITIAL STATEMENTS, "instructions au
C691e	08	PHP	démarrage" sauvegarde les indicateurs 6502 dont C
C692e	20 0D E6	JSR E60D	valide drive indiqué à TXTPTR, sinon valide DRVDEF
C695e	2C 16 C0	BIT C016	teste si b7 du flag "banque changée" est à 0
C698e	10 05	BPL C69F	si oui (banque pas changée), saute les 2 instructions suivantes
C69Ae	20 DB C6	JSR C6DB	sinon (banque changée), demande disquette cible
C69De	B0 2E	BCS C6CD	simple RTS si touche "ESC" pressée
C69Fe	20 D4 C6	JSR C6D4	charge secteur système dans BUF1 (au retour F4=#C2)
C6A2e	A9 00	LDA #00	
C6A4e	85 F8	STA F8	force F8 à zéro (on se demande bien pourquoi!)
C6A6e	C6 F4	DEC F4	décrémente HH de l'adresse de lecture qui revient à #C1
C6A8e	28	PLP	récupère les indicateurs 6502 dont C
C6A9e	90 08	BCC C6B3	continue en C6B3 si "INIST"
C6ABe	A9 14	LDA #14	si "DNAME", A = 20 pour lire 21 caractères
C6ADe	A2 00	LDX #00	pour 1er message "LFCRDisc name:"
C6AFe	A0 09	LDY #09	LL de l'adresse de lecture dans BUF1
			(en C109 débute le nom de la disquette, ce nom comporte 21 caractères)
C6B1e	D0 06	BNE C6B9	suite forcée en C6B9
C6B3e	A9 3B	LDA #3B	si "INIST", A = 59 pour lire 60 caractères
C6B5e	A2 01	LDX #01	pour 2ème message "LFCRInit statement:"
C6B7e	A0 1E	LDY #1E	LL de l'adresse de lecture dans BUF1
			(en C11E débutent les instructions au démarrage qui comportent 60 caractères)
C6B9e	84 F3	STY F3	F3 = LL de l'adresse de lecture dans BUF1
C6BBe	48	PHA	sauve A
C6BCe	20 64 D3	JSR D364	XAFSC affiche X+1ème message externe terminé par "caractère + 128"
C6BFe	68	PLA	récupère A
C6C0e	AA	TAX	compteur du nombre d'octets restant à lire
C6C1e	A0 00	LDY #00	index pour lecture
C6C3e	B1 F3	LDA (F3),Y	lit octet à l'adresse en F3/F4 + Y
C6C5e	20 2A D6	JSR D62A	XAFCAR affiche le caractère ASCII contenu dans A
C6C8e	C8	INY	octet suivant
C6C9e	CA	DEX	nombre d'octets restant à lire
C6CAe	10 F7	BPL C6C3	reboucle en C6C3 tant qu'il en reste
C6CCe	18	CLC	fini, force C à zéro et retourne
C6CDe	60	RTS	

Valide DRIVE, charge le secteur système dans BUF1

Variables utilisées

C000	DRIVE	n° du DRIVE actif
C100/C1FF	BUF1	pour charger le secteur système

Sous-programme commun à plusieurs commandes

C6CEe	20 0D E6	JSR E60D	valide drive indiqué à TXTPTR, sinon valide DRVDEF
C6D1e	8C 00 C0	STY C000	devient DRIVE actif
C6D4e	A9 14	LDA #14	piste 20
C6D6e	A0 01	LDY #01	secteur 1, c'est à dire secteur système
C6D8e	4C 5D DA	JMP DA5D	XPBUF1 Charge dans BUF1 le secteur Y de la piste A

Demande la disquette cible

Variable utilisée

C016	flag "banque changée"

Sous-programme commun à plusieurs commandes

C6DBe	2C 16 C0	BIT C016	teste si le b7 du flag "banque changée" est à zéro
C6DEe	10 10	BPL C6F0	si oui (banque pas changée), simple RTS en C6F0
C6E0e	A2 12	LDX #12	sinon (banque changée), indexe le message "LOAD"
C6E2e	20 64 D3	JSR D364	XAFSC affiche X+1ème message externe terminé par "caractère + 128"
C6E5e	20 48 D6	JSR D648	affiche "DISC IN DRIVE "lettre du drive" AND PRESS RETURN"
			puis demande un 'ESC' (C = 1) ou un 'RETURN' (C = 0)
C6E8e	58	CLI	autorise les interruptions
C6E9e	90 05	BCC C6F0	simple RTS en C6F0 si "RETURN"
C6EBe	68	PLA	si "ESC", élimine l'adresse de retour de la pile, puis

6ECe 68 PLA le JMP D206 execture la routine CBF0 "va à la ligne" en ROM
6EDe 4C 06 D2 JMP D206 et retourne à l'avant-dernier appelant

6F0e 60 RTS

Messages externes de la banque n°5 (C6F1 à C792)

6F1e 0A 0D 44 69 73 63 20 6E 61 6D 65 **BA**
LFCRDisc name**:**
6FDe 0A 0D 49 6E 69 74 20 73 74 61 74 65 6D 65 6E 74 **BA**
LFCRInit statement**:**
70Ee 0A 0D 4E 75 6D 20 73 74 65 70 20 20 **BA**
LFCRNum step■■**:**
71Be 0A 0D 4E 75 6D 20 6F 72 69 67 69 6E **BA**
LFCRNum origin**:**
728e 0A 0D 55 73 65 72 20 66 6F 6E 63 74 69 6F 6E 73 3A 0A **8D**
LFCRUser fonctions:LF**CR**
73Be 0A 0D 4B 65 79 62 6F 61 72 64 **BA**
LFCRKeyboard**:**
746e 0A 0D 44 72 69 76 65 **A0**
LFCRDrive■
74Ee 20 74 72 61 63 6B 73 **A0**
■tracks■
756e 73 69 6E 67 6C 65 **A0**
single■
75De 64 6F 75 62 6C 65 **A0**
double■
764e 73 69 64 65 64 **A0**
sided■
76Ae 75 6E 63 6F 6E 6E 65 63 74 65 64 **A0**
unconnected■
776e 41 43 43 45 4E 54 **A0**
ACCENT■
77De 53 45 54 **AC**
SET**,**
781e 4F 46 46 **AC**
OFF**,**
785e 41 **DA**
A**Z**
787e 51 **D7**
Q**W**
789e 45 52 54 59 0A **8D**
ERTYLF**CR**
78Fe 4C 4F 41 **C4**
LOA**D**

Le reste n'a aucun sens ici. Il s'agit en fait de la fin de la banque n°2 qui est absolument identique de C793b à C7FFb. C'est donc le résidu d'une compilation antérieure. Les #6D (109) octets corresp sont récupèrables pour ajouter une petite commande à la banque n°5.

Banque n°6 (adresse Cxxxf): INIT

Cette banque se trouve à partir du #5B (91ème) secteur de la disquette MASTER

C400f	00 00	EXTER	adresse des messages d'erreur externe (néant)
C402f	06 C4	EXTMS	adresse des messages externes D'autres messages ont été placés dans une zone supplémentaire de C6B0 à C6F8!

EXECUTION COMMANDE SEDORIC INIT

L'entrée de la commande INIT se fait théoriquement en C404 où se trouve en fait un JMP C482 qui est le début proprement dit!

Rappel de la syntaxe

INIT (lecteur(,NS(,NP(,NF))))

Où "lecteur" est une lettre de A à D, "NS" est le nombre de secteurs par piste (de 16 à 19), "NP" est le nombre de piste par face (de 21 à 99, mais en fait les lecteurs ne vont que jusqu'à 83 au maximum) et "NF" le nombre de faces ("S" pour simple face, "D" pour double face).

Les paramètres ne sont pas obligatoires, mais ils ne peuvent être ajoutés que si le ou les paramètres précédents sont présents, sauf "lecteur" qui peut être omis. On ne peut donc pas utiliser une succession de virgules comme avec DTRACK par exemple. Les syntaxes suivantes sont acceptées: INIT

INIT A
INIT,16
INIT,18,41
INIT,19,40,D

mais pas INIT A,,42,S
ni INIT A,,,S

Variables utilisées

0A/0B		pointeur dans le tampon de formatage
0C		nombre d'octets à copier
D0		longueur de la chaîne saisie par XLINPU
D1/D2		adresse de la chaîne saisie par XLINPU
F2		longueur de la chaîne à saisir
F3		options pour XLINPU
F4		mode de sortie de XLINPU
F5		nombre de pistes/face
F6		nombre de secteurs restant à écrire dans une piste
F7		n° de secteur à écrire dans le tampon de formatage
F8		n° de face (#00 si première, 01 si deuxième face) longueur de la chaîne à saisir
F9		nombre de secteurs à copier sur la disquette position de la chaîne saisie dans BUF1
3000/B100		secteurs en attente d'être recopié sur la nouvelle disquette
C000		DRIVE
C001		PISTE
C002		SECTEUR
C003/C004		RWBUF
C039		TABDRV, MODCLA, DEFNUM et DEFPAS
C075		caractère de remplissage
C100/C1FF		BUF1
C200/C2FF		BUF2
C474		table des secteurs réservés
C67A/C6A3		table de formatage
C6A4/C6AF		table des variables internes
C6A1		index de base pour gaps
C6A4f	00 98	adresse pour préparer en RAM une piste complète avec ses "gaps"
C6A6f	00 B1	adresse de fin de ce tampon de préparation de piste
C6A8f	70	#10 si 18 ou 19 secteurs/piste ou #70 si 16 ou 17 secteurs/piste
C6A9f	98	#98 est le HH de l'adresse du tampon de formatage
C6AAf	64	index élargi pour "gaps" = index de base + #3C
C6ABf	01	HH de cet index élargi (#100 octets pour les data)
C6ACf	11	nombre de secteurs par piste (repris dans F6)

6ADf 12 nombre de secteurs par piste + #01
6AEf 00 nombre de pistes/face (repris dans F5) [et nombre de faces]
6AFf 00 nombre de faces: #00 si une face et #01 si deux faces
6B0/C6F8 autres messages

Informations non documentées

Le nombre maximum de secteurs par disquette n'est pas de 1920 comme indiqué dans le manuel, mais de 1919. L'indicateur de double face est bogué ce qui pose quelques problèmes (voir plus loin). Le nom de la disquette (DNAME) peut comporter 21 caractères. Les instructions de démarrage (INIST) peuvent aller jusqu'à 60 caractères, si cela ne suffit pas, il faut utiliser un ficher de BOOT genre "BONJOUR" ou "MENU".

La commande INIT est sévèrement boguée. Le paramètre ",D" provoque bien un formatage en Double face, mais l'indicateur de Double face (le b7 de l'octet n°#09 de la bitmap) n'est pas mis à jour, ainsi qu'en témoigne le directory, qui indique désespérément "S/" au lieu de "D/". Pour contourner ce problème, il y a trois solutions:

1) Forcer "manuellement" le b7 du 10ème octet du 2ème secteur de la piste 20 à l'aide d'un éditeur de disquette (genre BDDISK). Par ex. pour une disquette formatée en 42 pistes, double face, il faut remplacer le #2A par #AA.

2) Utiliser 2 fois de suite la commande INIT. La première fois, il faut répondre Y à la demande "Format (Y/N):" et continuer sans remplir les champs DNAME et INIST. La seconde fois, il faut répondre N à la demande "Format (Y/N):" et continuer en remplissant les champs DNAME et INIST.

3) Ne pas utiliser de paramètre ",D" avec INIT, mais mettre préalablement à jour les valeurs par défaut à l'aide de TRACK ou de DTRAK (dans ce cas, il faut rebooter).

En effet, la fameuse bogue de INIT ne se produit pas si TABDRV (C039/C03C) est à jour, ce que l'on peut vérifier à l'aide de la commande SYS: le lecteur utilisé doit être marqué "double sided".

Utilisation en LM
Il faut écrire les paramètres dans le tampon clavier, initialiser TXTPTR puis faire le JSR F148 (voir les détails en annexe).

Version 2.0 GB
La commande INIT a été modifiée dans la version 2.0 GB. Ces modifications seront décrites plus loin, à la fin de l'analyse de la commande INIT.

Analyse de la syntaxe et saisie des paramètres
404f 4C 82 C4 JMP C482 qui est le début proprement dit

Messages externes de la banque n°6

407f 0D 0A 46 6F 72 6D 61 74 20 28 59 2F 4E 29 BA
CRLFFormat (Y/N)**:**

416f 0D 0A 0A 4E 61 6D 65 3A 20 20 20 20 20 20 20 20 20 20 20 20 20 58 58 2F 58 58 2F 58 D8
2 CRLFLFName:■■■■■■■■■■■■■■XX/XX/X**X**

433f 0D 0A 49 6E 69 74 20 61 6E 6F 74 68 65 72 20 64 69 73 63 20 28 59 2F 4E 29 BA
3 CRLFInit another disc (Y/N)**:**

44Df 0D 0A 0A 4D 61 73 74 65 72 20 64 69 73 63 20 28 59 2F 4E 29 BA
4 CRLFLFMaster disc (Y/N)**:**

462f 0D 0A 0A 49 6E 69 74 20 73 74 61 74 65 6D 65 6E 74 BA
5 CRLFLFInit statement**:**

Table des secteurs réservés
474f 01 02 03 04 07 0A 0D 10 les secteurs n°#01, #02, #03, #04, #07, #0A, #0D et #10 de la piste n°#14 sont réservés pour le Système, la Bitmap et le Catalogue

47Cf 4C 20 DE JMP DE20 "ILLEGAL QUANTITY ERROR"

47Ff 4C 23 DE JMP DE23 "SYNTAX ERROR"

Début proprement dit: analyse de syntaxe

Le s/p C482/C4DD valide le drive indiqué à TXTPTR ou le drive par défaut DRVDEF (C009), évalue le nombre de secteurs/piste indiqué à TXTPTR ("ILLEGAL QUANTITY ERROR" si <16 ou >19). Si aucun nombre n'est indiqué, prend 17 par défaut. Sauve le nombre de secteurs/piste en C6AC.

Puis il consulte TABDRV (C039/C03C), la table des drives actifs et y lit le nombre de pistes/face corresp au lecteur qui sera

utilisé (si besoin, cette valeur sera prise par défaut). Le s/p évalue alors, s'il existe, le nombre de pistes/face indiqué à TXTPTR, en vérifie la validité ("ILLEGAL QUANTITY ERROR" si <21 ou si >99) et prend ce nombre à la place de la valeur par défaut pour le sauver en C6AE.

Le s/p lit dans la table des drives actifs TABDRV (C039/C03C), le nombre de faces corresp au lecteur qui sera utilisé (si besoin, cette valeur sera prise par défaut). Puis il regarde s'il y a ",D" ou ",S" à TXTPTR ("SYNTAX ERROR" si autre chose). Si "D" doit être retenu, force à 1 le b7 de C6AE. Le nombre de pistes/face est donc codé par les bits b0 à b6 et le nombre de faces par b7 (Simple si b7 à zéro, Double si b7 à 1).

C482f	A2 11	LDX #11	nombre de secteurs par piste par défaut
C484f	20 0D E6	JSR E60D	valide drive indiqué à TXTPTR, sinon valide DRVDEF
C487f	F0 0E	BEQ C497	continue en C497 s'il n'y a plus de paramètres
C489f	20 2C D2	JSR D22C	D067/ROM exige une "," place TXTPTR sur l'octet suivant
C48Cf	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 secteurs par piste)
C48Ff	E0 10	CPX #10	teste si nombre de secteurs par piste < #10 (16)
C491f	90 E9	BCC C47C	si oui, "ILLEGAL QUANTITY ERROR"
C493f	E0 14	CPX #14	teste si nombre de secteurs par piste >= #14 (20)
C495f	B0 E5	BCS C47C	si oui, "ILLEGAL QUANTITY ERROR"
C497f	8E AC C6	STX C6AC	sauve nombre de secteurs par piste valide dans C6AC
C49Af	AC 00 C0	LDY C000	n° de DRIVE actif
C49Df	B9 39 C0	LDA C039,Y	lit le nombre de pistes par face et le nombre de faces corresp à ce lecteur dans la table des drives actifs TABDRV
C4A0f	29 7F	AND #7F	force à zéro le b7 qui code le nombre de faces
C4A2f	AA	TAX	X reçoit donc le nombre de pistes/face par défaut
C4A3f	20 9E D3	JSR D39E	XCRGOT relit le caractère à TXTPTR et le convertit en MAJUSCULE
C4A6f	F0 0E	BEQ C4B6	continue en C4B6 s'il n'y a plus de paramètres
C4A8f	20 2C D2	JSR D22C	D067/ROM exige une "," place TXTPTR sur l'octet suivant
C4ABf	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 pistes par face)
C4AEf	E0 15	CPX #15	teste si nombre de pistes par face < #15 (21)
C4B0f	90 CA	BCC C47C	si oui, "ILLEGAL QUANTITY ERROR"
C4B2f	E0 64	CPX #64	teste si nombre de pistes par face >= #64 (100)
C4B4f	B0 C6	BCS C47C	si oui, "ILLEGAL QUANTITY ERROR"
C4B6f	8E AE C6	STX C6AE	sauve nombre de pistes par face valide dans C6AE
C4B9f	AE 00 C0	LDX C000	n° de DRIVE actif
C4BCf	BC 39 C0	LDY C039,X	relit le nombre de pistes par face et le nombre de faces corresp à ce lecteur dans la table des drives actifs TABDRV
C4BFf	20 9E D3	JSR D39E	XCRGOT relit le caractère à TXTPTR et le convertit en MAJUSCULE
C4C2f	F0 11	BEQ C4D5	continue en C4D5 s'il n'y a plus de paramètres
C4C4f	20 2C D2	JSR D22C	D067/ROM exige une "," lit le caractère suivant et le convertit en MAJUSCULE
C4C7f	A0 FF	LDY #FF	le b7 de Y est à 1 si "Double face" (par défaut)
C4C9f	C9 44	CMP #44	le caractère lu est-il un "D"?
C4CBf	F0 05	BEQ C4D2	si oui, continue en C4D2
C4CDf	C8	INY	sinon, le b7 de Y passe à 0 ("Simple face")
C4CEf	C9 53	CMP #53	le caractère lu est-il un "S"?
C4D0f	D0 AD	BNE C47F	sinon, "SYNTAX ERROR"
C4D2f	20 98 D3	JSR D398	XCRGET lit le caractère suivant à TXTPTR et le convertit en MAJUSCULE
C4D5f	98	TYA	flag "Double face"/"Simple face"
C4D6f	29 80	AND #80	dont tous les bits sauf le b7 sont forcés à zéro
C4D8f	0D AE C6	ORA C6AE	si ce b7 est à 1, force à 1 le b7 de C6AE
C4DBf	8D AE C6	STA C6AE	le nombre de pistes/face est donc codé par les bits b0 à b6 et le nombre de faces par b7 (Simple si b7 à zéro, Double si b7 à 1)

Commence l'élaboration d'un secteur de bitmap

Le s/p C4DE/C4F1, remplit le buffer BUF2 de #00, puis de #FF à partir de la position #10 (17ème octet), initialise à #01 le nombre de secteurs de directory (à la position #08) et à #FF le 1er octet de la bitmap qui doit toujours contenir cette valeur.

C4DEf	20 D1 DA	JSR DAD1	XVBUF2 Rempli BUF2 de 0 (bitmap)
C4E1f	A9 FF	LDA #FF	valeur par défaut pour remplir le secteur de bitmap
C4E3f	A2 10	LDX #10	à partir de la position #10 (17ème octet)
C4E5f	9D 00 C2	STA C200,X	écrit un #FF dans BUF2 à la position X
C4E8f	E8	INX	indexe la position suivante et
C4E9f	D0 FA	BNE C4E5	reboucle tant que la fin de BUF2 n'est pas atteinte
C4EBf	E8	INX	X = #01 en sortie de boucle

```
4ECf  8E 08 C2    STX C208      nombre de secteurs de directory = 1
4EFf  8D 00 C2    STA C200      le 1er octet de bitmap doit toujours contenir #FF
```

Calcule le nombre total de secteurs

Le s/p C4F2/C50F, calcule AY = nombre de secteurs/piste (C6AC) que multiplie le nombre de pistes/face (b0 à b6 de C6AE) et multiplie le résultat par deux si "double face" (b7 de C6AE à 1).

```
4F2f  AD AE C6    LDA C6AE      nombre de pistes par face et nombre de faces
4F5f  29 7F       AND #7F       élimine b7 (nombre de faces)
4F7f  AA          TAX           X = nombre de pistes par face
4F8f  A9 00       LDA #00       mise à zéro LL du totalisateur
4FAf  A8          TAY           mise à zéro HH du totalisateur

4FBf  18          CLC           calcule AY =
4FCf  6D AC C6    ADC C6AC      nombre de secteurs par piste (C6AC)
4FFf  90 01       BCC C502      que multiplie
501f  C8          INY           nombre de pistes par face (X)

502f  CA          DEX           = nombre de secteurs par face
503f  D0 F6       BNE C4FB
505f  2C AE C6    BIT C6AE      teste s'il y a deux faces
508f  10 06       BPL C510      continue en C510 si "Simple face"
50Af  0A          ASL
50Bf  48          PHA
50Cf  98          TYA           si "double face", multiplie le résultat par deux
50Df  2A          ROL
50Ef  A8          TAY
50Ff  68          PLA
```

Teste la validité de AY = nombre total de secteurs

Le s/p C510/C51E teste si AY < 1920 (sinon "ILLEGAL QUANTITY ERROR"). Bogue dans le manuel, page 37: le nombre total de secteurs ne peut excéder 1919 et non 1920. Ceci est fâcheux, car cette dernière valeur aurait été bien utile avec les disquettes 3,5 pouces que l'on aurait pu formater en 16 secteurs/piste, 60 pistes/face et double face (16 X 60 X 2 = 1920).

Le s/p C519/C51E écrit le nombre total de secteurs AY dans BUF2 aux positions 02/03 (nombre de secteurs libres). Ce s/p a été modifié dans la Version 2.0 GB (2 octets différents).

```
510f  C0 07       CPY #07       teste si Y < #07 (soit AY < #0700 donc < 1792)
512f  90 05       BCC C519      si oui, continue en C519 (avec AY valable)
514f  D0 9E       BNE C4B4      si Y > #07 (ce qui correspond à AY > #07FF soit AY > 2047,
                                continue en C4B4 ("ILLEGAL QUANTITY ERROR")
516f  AA          TAX           si Y = #07, teste si le b7 de A est à 1, c'est à dire si AY >= #0780 ou 1920 en décimal (bogue
```

dans le manuel page 37 où il est écrit que le nombre total de secteurs ne peut exéder 1920, alors qu'en réalité, il ne peut exéder 1919, ce qui est facheux, car avec les disquettes 3,5 pouces on aurait pu formater au maximum, c'est à dire 16 secteurs par piste, 60 pistes par face et double face (16 X 60 X 2 = 1920).

```
517f  30 9B       BMI C4B4      "ILLEGAL QUANTITY ERROR" si AY >= 1920

519f  8D 02 C2    STA C202      ecrit le nombre total de secteurs dans BUF2
51Cf  8C 03 C2    STY C203      aux positions #02/03 (nombre de secteurs libres)
```

Ce sous-programme a été modifié dans la Version 2.0 GB (2 octets différents): voir à la fin de la banque n°6.

Formate la disquette?

Le s/p C51F/C52B affiche "CRLFFormat (Y/N):", saisit une touche, si "Y", formate la ou les faces (en C64A), sinon passe directement à la suite.

```
51Ff  A2 00       LDX #00       indexe le message n° #01 "CRLFFormat (Y/N):"
521f  20 64 D3    JSR D364      XAFSC affiche X+1ème message externe terminé par "caractère + 128"
524f  20 0F C6    JSR C60F      saisie de touche, retourne avec C = 1 si "Y"
527f  90 03       BCC C52C      saute l'instruction suivante si ce n'est pas "Y"
529f  20 4A C6    JSR C64A      si "Y", formate la ou les faces
```

Elabore un secteur système et l'écrit sur la disquette

Le s/p C52C/C566 :

- rempli le buffer BUF1 de #00,
- affiche le message: "CRLFLFName:■■■■■■■■■■■■■■■XX/XX/XX",
- appelle le s/p XLINPU pour saisir DNAME (21 caractères maxi),

- affiche le message "CRLFLFInit statement**:**",
- appelle le s/p XLINPU pour saisir INIST (60 caractères maxi),
- copie ces chaînes dans BUF1, à partir de la position #09,
- copie les 9 octets de TABDRV (C039/C03C), MODCLA (C03D), DEFNUM (C03E/C03F) et DEFPAS (C040/C041) dans BUF1 aux positions #00/08 et sauve BUF1 au secteur Y = 1 de la piste A = 20.

C52Cf	20 CE DA	JSR DACE	XVBUF1 Rempli BUF1 de 0
C52Ff	A2 01	LDX #01	indexe le message "CRLFLFName:■■■■■■■■■■■■XX/XX/X**X**"
C531f	20 64 D3	JSR D364	XAFSC affiche X+1ème message externe terminé par "caractère + 128"
C534f	A9 20	LDA #20	"espace" devient le
C536f	8D 75 C0	STA C075	caractère de remplissage pour LINPUT
C539f	A2 15	LDX #15	X = 21 pour retourner au début de la chaîne
C53Bf	20 69 EE	JSR EE69	affiche X fois "flèche gauche"
C53Ef	A0 88	LDY #88	Y = 1000 1000, options ",S" et ",E" de LINPUT C'est à dire interdit toute sortie avec les flèches et interdit l'affichage initial du caractère de remplissage, afin de ne pas effacer la chaîne affichée.
C540f	A9 15	LDA #15	A = 21 catactères à saisir
C542f	A2 09	LDX #09	X = position de DNAME dans BUF1
C544f	20 24 C6	JSR C624	saisit la chaîne et la copie dans BUF1
C547f	A2 08	LDX #08	pour copier les 9 octets de TABDRV (C039/C03C), MODCLA (C03D), DEFNUM (C03E/C03F) et DEFPAS (C040/C041)

C549f	BD 39 C0	LDA C039,X	lit un octet dans la zone C039 à C041
C54Cf	9D 00 C1	STA C100,X	et le copie dans BUF1 aux positions #00 à #08
C54Ff	CA	DEX	indexe l'octet précédent (opère par la fin)
C550f	10 F7	BPL C549	et reboucle tant qu'il en reste à copier
C552f	A2 04	LDX #04	indexe le message "CRLFLFInit statement**:**"
C554f	20 64 D3	JSR D364	XAFSC affiche X+1ème message externe terminé par "caractère + 128"
C557f	A9 3C	LDA #3C	A = 60 catactères à saisir
C559f	A2 1E	LDX #1E	X = position de INIST dans BUF1
C55Bf	A0 80	LDY #80	Y = 1000 0000, options ",S" de LINPUT C'est à dire interdit toute sortie avec les flèches.
C55Df	20 24 C6	JSR C624	saisit la chaîne et la copie dans BUF1
C560f	A9 14	LDA #14	piste 20 secteur 1
C562f	A0 01	LDY #01	secteur système (en-tête disquette)
C564f	20 91 DA	JSR DA91	XSBUF1 sauve BUF1 à la piste A et le secteur Y

Elabore le 1er secteur de directory et l'écrit sur la disquette

Le s/p C567/C575 rempli BUF1 de #00, copie #10 (qui vise le 17ème octet du secteur de directory) en C102 (n° de l'octet de la 1ère entrée libre de directory) et finalement sauve BUF1 au secteur Y = 4 de la piste A = 20.

C567f	20 CE DA	JSR DACE	XVBUF1 Rempli BUF1 de 0
C56Af	A9 10	LDA #10	vise le 17ème octet du secteur de directory
C56Cf	8D 02 C1	STA C102	n° de l'octet de la 1ère entrée libre de directory
C56Ff	A9 14	LDA #14	piste 20 secteur 4
C571f	A0 04	LDY #04	premier secteur de directory
C573f	20 91 DA	JSR DA91	XSBUF1 sauve BUF1 à la piste A et le secteur Y

Continue l'élaboration du secteur de bitmap et adapte le 2ème secteur en attente en RAM

Le s/p C576/C5B3 copie le nombre de pistes/face + nombre de faces (C6AE) dans BUF2 à la position #09. C'est ici que se trouve "la" bogue de INIT: le nombre de face n'est correctement mis à jour que si on ne formate pas! En effet, le b7 de C6AE est forcé à zéro par la routine de formatage C64A/C65F et ne retrouve jamais sa valeur initiale. Puis le s/p copie le nombre de pistes/face dans BUF2 à la position #06 et le nombre de secteurs/piste (C6AC) dans BUF2 à la position #07, affiche le message "CRLFLFMaster disc (Y/N)**:**", saisit une touche. Si "Y", affiche "M" et initialise F9 à #5E (94 secteurs à copier), sinon, affiche "S" et initialise F9 à #08 (8 secteurs à copier). Le s/p place #01 si "Slave" ou #00 si "Master" à la position #16 du 2ème secteur en attente en RAM (secteur de boot) (c'est à dire en 3116), ainsi que dans BUF2 à la position #0A, copie le nombre de secteurs/piste (C6AC) plus un à la position #15 du 2ème secteur en RAM (c'est à dire en 3115). Ce s/p a été modifié dans la Version 2.0 GB (3 octets différents).

C576f	AD AE C6	LDA C6AE	nombre de pistes par face et nombre de faces
C579f	8D 09 C2	STA C209	écrit A dans BUF2 à la position #09. C'est ici que se trouve "la" bogue de INIT: le nombre de face n'est correctement mis à jour que si on ne formate pas! En effet, le b7 de C6AE est forcé à zéro par la routine de formatage de C64A à C65F et ne retrouve jamais sa valeur initiale.
C57Cf	29 7F	AND #7F	nombre de pistes par face seul
C57Ef	8D 06 C2	STA C206	écrit A dans BUF2 à la position #06
C581f	AD AC C6	LDA C6AC	nombre de secteurs par piste
C584f	8D 07 C2	STA C207	écrit A dans BUF2 à la position #07
C587f	A2 03	LDX #03	indexe le message "CRLFLFMaster disc (Y/N)**:**"
C589f	20 64 D3	JSR D364	XAFSC affiche X+1ème message externe terminé par "caractère + 128"
C58Cf	20 0F C6	JSR C60F	saisie de touche, retourne avec C = 1 si "Y"
C58Ff	A0 53	LDY #53	"S" pour "Slave" par défaut (C = 0)
C591f	A2 08	LDX #08	8 = nombre de secteurs à copier si "Slave"

```
593f  90 04       BCC C599      si C = zéro, saute les deux instructions suivantes
595f  A0 4D       LDY #4D       "M" pour "Master" (C = 1)
597f  A2 5E       LDX #5E       94 = nombre de secteurs à copier si "Master"
599f  86 F9       STX F9        F9 = nombre de secteurs à copier
59Bf  A9 00       LDA #00       calcule A = #01 si "Slave" ou #00 si "Master"
59Df  2A          ROL           sauve ce résultat à la position #16 (23ème octet)
59Ef  49 01       EOR #01       du 2ème secteur en attente en RAM (secteur de boot),
5A0f  8D 16 31    STA 3116      ainsi que dans BUF2 à la position #0A
5A3f  8D 0A C2    STA C20A      (flag type de disquette)
5A6f  98          TYA           "S" ou "M"
5A7f  20 2A D6    JSR D62A      XAFCAR affiche le caractère ASCII contenu dans A
5AAf  AE AC C6    LDX C6AC      nombre de secteurs par piste
5ADf  E8          INX           plus un
5AEf  8E 15 31    STX 3115      résultat à la position #15 du 2ème secteur en RAM
5B1f  20 06 D2    JSR D206      CBF0/ROM va à la ligne
```

Ce sous-programme a été modifié dans la Version 2.0 GB (3 octets différents): voir à la fin de la banque n°6.

Copie F9 secteurs de la RAM sur la disquette

Le s/p C5B4/C5E6 copie sur la disquette chacun des F9 secteurs en attente en RAM, à partir de l'adresse 3000, à l'aide d'une boucle contrôlant les valeurs de RWBUF, PISTE, SECTEUR et la mise à jour de la bitmap.

```
5B4f  A9 00       LDA #00
5B6f  A0 30       LDY #30       RWBUF = #3000 = début secteurs en attente en RAM
5B8f  8D 03 C0    STA C003
5BBf  8C 04 C0    STY C004
5BEf  A0 00       LDY #00
5C0f  8C 01 C0    STY C001      n° de PISTE active = #00
5C3f  C8          INY
5C4f  78          SEI           interdit les interruptions

5C5f  8C 02 C0    STY C002      n° de SECTEUR actif = #01
5C8f  AD 01 C0    LDA C001      n° de PISTE active
5CBf  20 2D DD    JSR DD2D      XCREAY marque dans la bitmap en BUF2 que le secteur AY est occupé
5CEf  20 A4 DA    JSR DAA4      XSVSEC écrit un secteur selon DRIVE, PISTE, SECTEUR et RWBUF
5D1f  EE 04 C0    INC C004      incrémente HH de RWBUF (page suivante)
5D4f  AC 02 C0    LDY C002      n° de SECTEUR actif
5D7f  CC AC C6    CPY C6AC      teste si n° SECTEUR actif < nombre secteurs/piste
5DAf  90 05       BCC C5E1      si oui, saute les deux instructions suivantes
5DCf  EE 01 C0    INC C001      incrémente le n° de PISTE active et reset le n° de
5DFf  A0 00       LDY #00       SECTEUR actif (pour avoir #01 en début de boucle)

5E1f  C8          INY           secteur suivant
5E2f  C6 F9       DEC F9        décrémente le nombre de secteurs à copier
5E4f  D0 DF       BNE C5C5      et reboucle en tant qu'il en reste à copier
5E6f  58          CLI           fini, autorise les interruptions
```

Termine l'élaboration du secteur de bitmap et le sauve

Le s/p C5E7/C5FB marque dans la bitmap que les secteurs #01, 02, 03, 04, 07, 0A, 0D et #10 de la piste #14 sont occupés (réservés pour le système, la bitmap et le catalogue), puis sauve le secteur de bitmap sur la disquette. Ce s/p a été modifié dans la Version 2.0 GB (2 octets différents).

```
5E7f  A2 07       LDX #07       index pour lire les 8 octets de la table C474
5E9f  86 F9       STX F9        sauve X dans F9
5EBf  A6 F9       LDX F9        récupère X en début de boucle
5EDf  A9 14       LDA #14
5EFf  BC 74 C4    LDY C474,X    les secteurs n°#01, 02, 03, 04, 07, 0A, 0D et #10 de la piste n°#14 sont réservés pour le
Système, la Bitmap et le Catalogue
5F2f  20 2D DD    JSR DD2D      XCREAY marque dans la bitmap en BUF2 que le secteur AY est occupé
5F5f  C6 F9       DEC F9        octet précedent
5F7f  10 F2       BPL C5EB      reboucle tant qu'il en reste
5F9f  20 8A DA    JSR DA8A      XSMAP sauve secteur de bitmap sur la disquette
```

Ce sous-programme a été modifié dans la Version 2.0 GB (2 octets différents): voir à la fin de la banque n°6.

Une autre? (même format)

Le s/p C5FC/C60E affiche "CRLFInit another disc (Y/N):", saisit une touche. Si c'est "Y", reprend en C51F pour formatage identique, sinon retourne. Ce s/p a été modifié dans la Version 2.0 GB (2 octets différents).

```
5FCf  A2 02       LDX #02       indexe le message "CRLFInit another disc (Y/N):"
5FEf  20 64 D3    JSR D364      XAFSC affiche X+1ème message externe terminé par "caractère + 128"
601f  20 0F C6    JSR C60F      saisie de touche, retourne avec C = 1 si "Y"
```

```
C604f  90 06       BCC C60C     saute les deux instructions suivantes si pas "Y"
C606f  20 06 D2    JSR D206     CBF0/ROM va à la ligne
C609f  4C 1F C5    JMP C51F     reprend en C51F pour formatage identique

C60Cf  4C 06 D2    JMP D206     CBF0/ROM va à la ligne et termine
```

Ce sous-programme a été modifié dans la Version 2.0 GB (2 octets différents): voir à la fin de la banque n°6.

Saisie de touche, retourne avec C = 1 si "Y"

Le s/p C60F/C623 attend qu'une touche soit pressée et revient avec le code ASCII corresp (lettre convertie en MAJUSCULE). Si c'est un "ESC", dépile une adresse de retour, sort de la commande INIT et retourne au s/p appelant précédent. Si c'est "Y", retourne avec C = 1, sinon avec C = 0.

```
C60Ff  20 02 D3    JSR D302     JSR EB78/ROM si touche frappée alors N = 1 et A = code ASCII corresp, sinon N = 0
C612f  10 FB       BPL C60F     reprend la saisie tant que pas de touche pressée
C614f  20 A1 D3    JSR D3A1     minMAJ convertit en MAJUSCULE le caractère dans A
C617f  C9 1B       CMP #1B      est-ce "ESC"?
C619f  F0 06       BEQ C621     si oui, termine en C621
C61Bf  C9 59       CMP #59      est-ce "Y"?
C61Df  F0 01       BEQ C620     si oui, termine en C620 avec C = 1
C61Ff  18          CLC          si autre touche, termine en C620 avec C = 0

C620f  60          RTS

C621f  68          PLA          dépile une adresse de retour et retourne
C622f  68          PLA          donc au sous-programme appelant précédent,
C623f  60          RTS          c'est à dire, sort de la commande INIT
```

Saisit une chaîne de A caractères et la copie à partir de la X^{ème} position dans BUF1

Le s/p C624/C649 appelle les s/p ED36 XLINPU, routine de saisie de chaîne et D73E (curseur visible), teste si F4, le mode de sortie de XLINPU est égal à 1 ("ESC"), si oui, dépile une adresse de retour et termine en retournant au s/p appelant précédent. Sinon, copie les F8 caractères de la chaîne saisie dans BUF1 à partir de la position X et retourne.

```
C624f  85 F8       STA F8       longueur de la chaîne à saisir
C626f  85 F2       STA F2       idem
C628f  86 F9       STX F9       position de la chaîne dans BUF1
C62Af  84 F3       STY F3       options E, S, C, J, K pour LINPUT
C62Cf  20 36 ED    JSR ED36     XLINPU (routine de saisie de chaîne). En entrée, C075 contient le caractère à utiliser pour
                                matérialiser la fenêtre. Au retour, F4 contient le mode de sortie, D0 et D1/D2 donnent la longueur et l'adresse de la chaîne
                                dans la zone de stockage des chaînes sous HIMEM.

C62Ff  20 3E D7    JSR D73E     "CURSEUR ON" (curseur visible = vidéo inverse)
C632f  A6 F4       LDX F4       mode de sortie de XLINPU
C634f  CA          DEX          teste si mode de sortie est différent de 1 ("ESC")
C635f  D0 03       BNE C63A     si oui, saute les trois instructions suivantes
C637f  68          PLA          dépile une adresse de retour et retourne
C638f  68          PLA          donc au sous-programme appelant précédent,
C639f  60          RTS          c'est à dire, sort de la commande INIT

C63Af  A0 00       LDY #00      Y = index de lecture dans la chaîne saisie
C63Cf  A6 F9       LDX F9       X = index d'écriture dans BUF1
C63Ef  B1 D1       LDA (D1),Y   lecture caractère dans la chaîne saisie
C640f  9D 00 C1    STA C100,X   écriture dans BUF1
C643f  E8          INX          suivant en écriture
C644f  C8          INY          suivant en lecture
C645f  C4 F8       CPY F8       teste si fini (nombre copié = longueur chaîne)
C647f  D0 F5       BNE C63E     sinon, reboucle en C63E
C649f  60          RTS
```

Formate la ou les faces

Le s/p C64A/C679 :

- copie en C6AF le nombre de faces selon C6AE (#00 pour 1 et #01 pour 2 faces),
- affiche "CRLFLFFormating Side 0 Track 00",
- appelle le s/p D740 (curseur caché),
- rétablit en C6AE le nombre de pistes/face (hélas sans tenir compte du nombre de faces!)
- appelle le s/p C745 qui formate la première face, si une seule face,
- affiche "LFLFCRFormating complete" et retourne.

- Sinon, affiche "<- <- <- <- <- <- <- <- <- <- 1 Track 00",
- appelle le s/p C745 qui formate la 2ème face,

- affiche "LFLFCRFormating complete"
- et retourne.

Ce s/p a été modifié dans la Version 2.0 GB (9 octets différents).

64Af	**0E** AE C6	ASL C6AE	b7 -> C (à zéro si "Simple", à 1 si "Double" face)
64Df	**A9 00**	LDA #00	force A à zéro
64Ff	**2A**	ROL	C -> b0 de A et b7 de A (nul) -> C (important)
650f	8D AF C6	STA C6AF	C6AF = #00 si une face et #01 si deux faces
653f	A9 B0	LDA #B0	AY = adresse C6B0 de la chaîne:
655f	A0 C6	LDY #C6	"CRLFLFFormating Side 0 Track 00"
657f	20 37 D6	JSR D637	AFSTR affiche chaîne terminée par 0 d'adresse AY
65Af	20 40 D7	JSR D740	"CURSEUR OFF" (curseur caché = vidéo normale)
65Df	**4E AE C6**	LSR C6AE	rétablit nombre de pistes/face (sans nombre face)
660f	20 45 C7	JSR C745	formate la première face (car C vaut toujours 0)
663f	AD **AF** C6	LDA C6AF	teste si une seule face
666f	**F0** 0B	BEQ C673	si oui, continue en C673
668f	A9 CD	LDA #CD	sinon, AY = adresse C6CD pour chaîne:
66Af	A0 C6	LDY #C6	"<- <- <- <- <- <- <- <- <- <- 1 Track 00"
66Cf	20 37 D6	JSR D637	AFSTR affiche chaîne terminée par 0 d'adresse AY
66Ff	38	SEC	pour deuxième face
670f	20 45 C7	JSR C745	formate la deuxième face
673f	A9 E2	LDA #E2	AY = adresse C6E2 pour chaîne:
675f	A0 C6	LDY #C6	"LFLFCRFormating complete"
677f	4C 37 D6	JMP D637	AFSTR affiche chaîne d'adresse AY et retourne. Ce sous-programme a été modifié dans la

Version 2.0 GB (9 octets différents): voir à la fin de la banque n°6.

Table de formatage (C67A à C6A3)

67Af	28 **4E** 0C **00** 03 **F6** 01 **FC** 28 **4E** FF	(soit #60 = 96 octets au total)
685f	0C **00** 03 **F5**	(soit 15 octets)
689f	01 **FE** 01 **00** 01 **00** 01 **00** 01 **01** 01 **F7**	(FE pp ff ss 01 CRC, soit 6 octets)
695f	16 **4E** 0C **00** 03 **F5**	(soit 37 octets)
69Bf	01 **FB** 00 **00** 01 **F7**	(soit 258 octets)
6A1f	28 **4E** FF	(soit 40, 30 ou 12 octets selon le nombre de secteurs/piste)

La 1ère partie (C67A/C684) sert à élaborer le début de piste.

La 2ème (C685/C6A3) sert à élaborer une "zone secteur" ("gap" + 256 octets de secteur proprement dit).

En C6A1 se trouve l'index de base pour "gaps" qui est variable et vaut
#28 si 16 ou 17 secteurs/piste,
#1E si 18 secteurs/piste ou #0C si 19 secteurs/piste.
Cet index est ensuite augmenté de #3C (index de base élargi).

Lorsque l'on entre dans la table en C685, le nombre total d'octets écrits dans chaque "zone secteur" dans le tampon est donc lui aussi variable. Il est de 356 (#28 + #3C + #100) si 16 ou 17 secteurs/piste,
346 (#1E + #3C + #100) si 18 secteurs/piste ou
328 (#0C + #3C + #100) si 19 secteurs/piste
et a son importance dans la fiabilité, tant en écriture qu'en lecture, comme nous l'indiquons plus loin.

Variables internes à la banque N°6

6A4f	00 98	adresse pour préparer en RAM une piste complète avec ses "gaps"
6A6f	00 B1	adresse de fin de ce tampon de préparation de piste
6A8f	70	#10 si 18 ou 19 secteurs/piste ou #70 si 16 ou 17 secteurs/piste
6A9f	98	#98 est le HH de l'adresse du tampon de formatage
6AAf	64	index élargi pour "gaps" = index de base + #3C
6ABf	01	HH de cet index élargi (#100 octets pour les data)
6ACf	11	nombre de secteurs par piste (repris dans F6)
6ADf	12	nombre de secteurs par piste + #01
6AEf	00	nombre de pistes/face (repris dans F5) [et nombre de faces]
6AFf	00	nombre de faces: #00 si une face et #01 si deux faces

Autres messages (C6B0 à C6F8) terminés par #00

6B0f 0D 0A 0A 46 6F 72 6D 61 74 69 6E 67 20 53 69 64 65 20 30 20 54 72 61 63 6B 20 30 30 **00**
CRLFLFFormating Side 0 Track 00**BRK**

C6CDf 08 08 08 08 08 08 08 08 08 08 31 20 54 72 61 63 6B 20 30 30 **00**
02 <- <- <- <- <- <- <- <- <- <- 1 Track 00**BRK**

C6E2f 0A 0A 0D 11 46 6F 72 6D 61 74 69 6E 67 20 63 6F 6D 70 6C 65 74 65 **00**
03 LFLFCR.Formating complete**BRK**

Met à jour les n° de piste n° de face et n° de secteur
Le s/p C6F9/C732 met à jour les n° de piste, de face et de secteur de chaque "zone secteur" dans le tampon de formatage à l'aide d'une boucle et du pointeur 0A/0B qui, au début, vise en 9870 (si 16 ou 17 secteurs/piste) ou en 9810 (si 18 ou 19 secteurs/piste) et est ensuite incrémenté de 356, 346 ou 328 pour passer à la zone suivante.

Petite particularité: le n° de secteur est mis à jour à l'aide du s/p 373A. En effet, le premier secteur d'une piste n'est pratiquement jamais le n°1 (voir plus bas). Supposons qu'une piste à formater en 17 secteurs/piste commence au n° 2, par incrémentations successives, le dernier secteur aurait n° 18. Comme cela n'est pas possible, ce n° est ramené à 1 par le s/p C73A.

C6F9f	AD A8 C6	LDA C6A8	#10 si 18 ou 19, #70 si 16 ou 17 secteurs/piste
C6FCf	AC A9 C6	LDY C6A9	#98 HH de l'adresse du tampon de formatage
C6FFf	85 0A	STA 0A	0A/0B = #9810 ou #9870 = pointeur dans ce tampon
C701f	84 0B	STY 0B	(c'est l'adresse du 1er n° de piste)
C703f	A2 00	LDX #00	compteur du nombre de secteurs déjà préparés
C705f	A0 00	LDY #00	index écriture dans chaque zone de préparation
C707f	AD 01 C0	LDA C001	n° de PISTE active
C70Af	29 7F	AND #7F	dont on force à zéro le b7 (flag nombre de faces)
C70Cf	91 0A	STA (0A),Y	écrit le n° de piste #pp au pointeur + Y
C70Ef	C8	INY	position suivante
C70Ff	A5 F8	LDA F8	A = n° de face
C711f	91 0A	STA (0A),Y	écrit le n° de face #ff au pointeur + Y
C713f	C8	INY	position suivante
C714f	A5 F7	LDA F7	A = n° de secteur
C716f	18	CLC	prépare une addition
C717f	69 01	ADC #01	A = n° de secteur + #01
C719f	20 3A C7	JSR C73A	mise à jour éventuelle de F7. Le premier secteur d'une piste n'est pratiquement jamais le n°1 (voir plus bas). Supposons qu'une piste à formater en 17 secteurs par piste commence au n° 2, par incrémentations successives, le dernier secteur aurait n° 18. Comme cela n'est pas possible, ce n° est ramené à 1 par le sous-programme C73A.

C71Cf	91 0A	STA (0A),Y	écrit le n° de secteur #ss au pointeur + Y
C71Ef	18	CLC	prépare une addition
C71Ff	AD AA C6	LDA C6AA	#64 = index élargi pour "gaps"
C722f	65 0A	ADC 0A	mise à jour du pointeur dans le tampon
C724f	85 0A	STA 0A	adresse = adresse + #164
C726f	AD AB C6	LDA C6AB	#01 = HH de #164, augmente le pointeur d'une page
C729f	65 0B	ADC 0B	corresp aux 256 octets de data du secteur
C72Bf	85 0B	STA 0B	(adresse en 0A/0B vise le #pp du secteur suivant)
C72Df	E8	INX	compteur du nombre de secteurs déjà préparés
C72Ef	EC AC C6	CPX C6AC	teste si X atteint le nombre de secteurs par piste
C731f	D0 D2	BNE C705	sinon, reboucle en C705

Calcul du 1er n° de secteur de la piste suivante
Le s/p C733/C739 calcule n° du secteur qui vient d'être préparé + nombre de secteurs/piste - #04.

C733f	A5 F7	LDA F7	n° du secteur qui vient d'être préparé auquel on
C735f	6D AC C6	ADC C6AC	ajoute le nombre de secteurs par piste et on
C738f	E9 04	SBC #04	retranche #04 (les pistes ne commencent pas toutes au secteur n°1, mais selon un ordre plus complexe probablement afin d'avoir une plus grande rapidité d'accès. Les pistes commencent successivement aux secteurs 1, 14, 10, 6, 2, 15, 11, 7, 3, 16, 12, 8, 4, 17, 13, 9, 5, 1 etc... Ainsi, la piste n° 20 d'une disquette commence avec le secteur n°6!)

Mise à jour éventuelle de F7
Pour que #01 =< n° du secteur à écrire =< nombre de secteurs par piste

C73Af	CD AD C6	CMP C6AD	teste si A < nombre de secteurs par piste + #01
C73Df	90 03	BCC C742	si oui (C = 0), saute l'instruction suivante
C73Ff	ED AC C6	SBC C6AC	sinon (C = 1), retranche de A le nombre maximum de secteurs par piste. Exemple: lors du formatage en 17 secteurs par piste, lorsque A atteint 18, on retranche 17 et le n° est ramené à 1.

```
742f 85 F7      STA F7      et replace le résultat dans F7
744f 60         RTS
```

Rappels sur la structure d'une piste

Une piste Sédoric est formée de 16, 17, 18 ou 19 secteurs de 256 octets. Entre ces secteurs de data proprement dits, se trouvent des "gaps" qui contiennent des informations utiles pour le contrôleur de lecteur.

Le début d'une piste (facultatif) commence par une série de 80 fois #4E, 12 fois #00, #C2 #C2 #C2 #FC, puis par une série de 50 fois #4E (selon la norme IBM) ou 40 fois #4E, 12 fois #00, #C2 #C2 #C2 #FC et 40 fois #4E (Sédoric, si 16 ou 17 secteurs/piste, sinon rien), soit une économie de 50 à 146 octets.

Chaque secteur est alors précédé d'un champ d'identification formé de: 12 fois #00, 3 octets #A1 de synchronisation, #FE #pp #ff #ss #tt CRC CRC puis 22 fois #4E. Le champ de data est constitué de 12 fois #00, 3 octets #A1 de synchronisation, le marqueur de début de data #FB, les 512 (IBM) ou 256 (Oric) octets du secteur et enfin CRC CRC. Chaque secteur est suivi de 80 fois #4E (ceci selon la norme IBM et 40, 30 ou 12 fois #4E dans le cas de Sédoric, selon le nombre de secteurs/piste, soit une économie de 40 à 68 octets/secteur). Puis vient le secteur suivant... (NB: CRC = octet de checksum, #pp = n°piste, #ff = n°face, #ss = n°secteur, #tt = taille (#01 pour les 256 octets du Sédoric, #02 pour les 512 octets de l'IBM PC etc...)).

La fin de piste est marquée par un nombre très variable d'octets [#4E] (facultatif). La piste étant circulaire, toutes les valeurs entre la fin de piste et le début de piste sont sans signification.

Selon le nombre de secteurs/piste, la place disponible pour les "gaps" est variable. Toutes ces indications sont théoriques, lorsqu'on lit une piste et ses "gaps" avec un utilitaire spécialisé tel que NIBBLE, on obtient des différences. Le premier des 3 octets de synchronisation, par exemple, est toujours faux, puisque la synchronisation n'a pas encore été obtenue! De plus, la zone située entre la fin des data et les octets de synchronisation de l'en-tête du secteur suivant (soit le "gap" situé entre deux secteurs) contient souvent n'importe quoi. En fait, ni le contrôleur de drive, ni le drive lui-même, ne répondent instantanément. Il s'ensuit des bavures lors des changements d'état de la tête de lecture/écriture. C'est la raison d'être de ces "gaps", qui servent à protéger le secteur suivant. Si l'on voulait augmenter le nombre de secteurs/piste, il faudrait diminuer la taille des "gaps" et donc la fiabilité.

Soit en résumé:

Début de la piste (facultatif): 80 fois #4E, 12 fois #00, #C2 #C2 #C2 #FC et 50 fois #4E (soit 146 octets selon la norme IBM) ou 40 fois #4E, 12 fois #00, #C2 #C2 #C2 #FC et 40 fois #4E (pour Sédoric, soit 96 octets si 16 ou 17 secteurs/piste, sinon rien).

Pour chaque secteur: 12 fois #00, 3 fois #A1, #FE #pp #ff #ss #tt CRC CRC, 22 fois #4E, 12 fois #00, 3 fois #A1, #FB, les 512 octets, CRC CRC, 80 octets #4E (#tt = #02) (soit 141 + 512 = 653 octets selon la norme IBM) ou 12 fois #00, 3 fois #A1, #FE #pp #ff #ss #01 CRC CRC, 22 fois #4E, 12 fois #00, 3 fois #A1, #FB, les 256 octets, CRC CRC et enfin 12, 30 ou 40 octets #4E (selon le nombre de secteurs/piste). Soit environ 256 + (72 à 100) = 328 à 356 octets pour Sédoric.

Fin de la piste (facultatif): un nombre variable d'octets [#4E].

Selon Nibble, une piste IBM compte 146 octets de début de piste + 9 secteurs de 653 octets + 257 octets de fin de piste = 6280 octets. Une piste Sédoric, formatée à 17 secteurs, compte 96 octets de début de piste + 17 secteurs de 358 octets + 98 octets de fin de piste = 6280 octets. Une piste Sédoric, formatée à 19 secteurs, compte 0 octet de début de piste + 19 secteurs de 328 octets + 48 octets de fin de piste = 6280 octets. On comprend mieux le manque de fiabilité du formatage en 19 secteurs/piste dû à la faible largeur des zones de sécurité (12 [#4E] entre chaque secteur et 48 octets entre le dernier et le premier).

Lors de l'élaboration du tampon de formatage Sédoric, les octets #C2 sont remplacés par des octets #F6, les octets #A1 sont remplacés par des octets #F5 et chaque paire de 2 octets [CRC CRC] et remplacée par un octet #F7. Comme on le voit, nombre de variantes sont utilisées, sauf la zone 22 fois #4E, 12 fois #00, 3 fois #A1 qui est strictement obligatoire.

Formate la première face si C = 0 et la deuxième si C = 1

Initialisations
Le s/p C745/C794 initialise divers pointeurs (dont 0A/0B et RWBUF) et variables nécessaires à l'élaboration d'une piste complète avec ses "gaps" dans le tampon de formatage 9800/B100, qui sera envoyé sur la disquette.

```
745f 08         PHP         sauvegarde les indicateurs 6502 dont C
746f 08         PHP         idem une 2ème fois (génial, voir plus loin)
747f AD AC C6   LDA C6AC
74Af 85 F6      STA F6      F6 = nombre de secteurs par piste
74Cf 8D AD C6   STA C6AD
74Ff EE AD C6   INC C6AD    C6AD = nombre de secteurs par piste + #01
```

Selon le nombre de secteurs par piste, la place restante pour les codes placés entre les secteurs (dans les gaps) est variable, on détermine:

```
752f A0 0C      LDY #0C     Y = #0C (soit 12, valeur pour 19 secteurs/piste)
```

```
C754f  C9 13      CMP #13      teste si A >= #13 (en fait si A = 19 valeur maxi)
C756f  B0 08      BCS C760     si oui, continue en C760 (OK pour Y)
C758f  A0 1E      LDY #1E      sinon, Y = #1E (30, valeur pour 18 secteurs/piste)
C75Af  C9 12      CMP #12      teste si A >= #12 (en fait si A = 18)
C75Cf  B0 02      BCS C760     si oui, continue en C760 (OK pour Y)
C75Ef  A0 28      LDY #28      sinon, Y = #28 (soit 40, valeur pour A = 16 ou 17)
C75Ef  A0 2F      LDA #2F      variante dans le cas de Stratoric V1.0

C760f  8C A1 C6   STY C6A1     sauve Y en C6A1 (index de base)
C763f  18         CLC
C764f  98         TYA
C765f  69 3C      ADC #3C      C6AA = Y + #3C (index élargi)
C767f  8D AA C6   STA C6AA
C76Af  AD AE C6   LDA C6AE
C76Df  85 F5      STA F5       F5 = nombre de pistes par face
C76Ff  AD A4 C6   LDA C6A4
C772f  AC A5 C6   LDY C6A5     AY = #9800 (adresse présente en C6A4/C6A5)
C775f  85 0A      STA 0A       0A/0B = #9800 (adresse pour préparer en RAM
C777f  84 0B      STY 0B       une piste complète avec ses "gaps")
C779f  8D 03 C0   STA C003     RWBUF = #9800 (adresse du tampon en RAM
C77Cf  8C 04 C0   STY C004     qui sera envoyé sur la disquette)
C77Ff  28         PLP          récupère les indicateurs 6502 dont C
C780f  A9 00      LDA #00      force à 0 le registre A
C782f  AA         TAX          force X à 0 (index de lecture dans la table C67A)
C783f  A8         TAY          force Y à 0 (index d'écriture dans le tampon 9800)
C784f  2A         ROL          force le b0 de A selon C donc A = C
C785f  85 F8      STA F8       F8 = #00 si 1ère face ou #01 si 2ème face
C787f  28         PLP          récupère les indicateurs 6502 dont C qui passe
C788f  6A         ROR          dans b7 de A dont l'ancien b0 est éliminé
C789f  8D 01 C0   STA C001     donc b7 de PISTE porte C. En clair, si Simple face le n° de la première piste est #00,
                               si Double face, ce n° est #80 (pas mal!)
C78Cf  86 F7      STX F7       F7 = #00 n° du premier secteur
C78Ef  AD AC C6   LDA C6AC     A = nombre de secteurs par piste
C791f  C9 12      CMP #12      teste si A >= #12 (c'est à dire, si vaut 18 ou 19)
C791f  C9 11      CMP #11      variante dans le cas de Stratoric V1.0
C793f  B0 06      BCS C79B     si oui, continue en C79B
```

Elabore un début de piste dans le tampon de formatage

Si formatage en 16 ou 17 secteurs/piste, le s/p C795/C797 appelle le s/p C7E3 qui élabore un en-tête de piste de 96 octets, au début du tampon (de 9800 à 985F), selon les valeurs de la 1ère partie de la table C67A.

```
C795f  20 E3 C7   JSR C7E3     sinon, élabore un en-tête de piste de 96 octets, au début du tampon (de 9800 à 985F), selon
                               les valeurs de la 1ère partie de la table C67A (X = #00). Ceci uniquement lors d'un formatage en 16 ou 17 secteurs par piste
                               (l'en-tête est facultatif).
```

Ajuste le LL du pointeur de mise à jour

Le s/p C798/C79F copie en C6A8 la valeur #70 pour 16 ou 17 secteurs/piste ou la valeur #10 pour 18 ou 19 secteurs/piste.

```
C798f  A9 70      LDA #70      valeur pour 16 ou 17 secteurs par piste
C79Af  2C A9 10   BIT 10A9     et continue en C79D

C79Bf  A9 10      LDA #10      valeur pour 18 ou 19 secteurs par piste

C79Df  8D A8 C6   STA C6A8     sauve en C6A8 la valeur retenue
```

Elabore le reste de la piste dans le tampon de formatage

A l'aide d'une boucle, pour chaque secteur, le s/p C7A0/C7A8 appelle le s/p C7E3 qui élabore une "zone secteur" selon les valeurs de la 2ème partie de la table C67A. Selon le nombre de secteurs/piste (16, 17, 18 ou 19), le nombre total d'octets écrits sera différent (356, 356, 346 ou 328 respectivement). Cette différence porte sur le nombre de "#4E" placés en fin de secteur.

```
C7A0f  A2 0B      LDX #0B      dans les 2 cas, en début de boucle, X = #0B (index pour lecture de la deuxième partie de la
                               table C67A), tandis que Y (index d'écriture dans le tampon évoluera de #00 (ou #60 si 16 ou 17 secteurs par piste) à #1900,
                               lorsqu'il pointera sur la fin du tampon.

C7A2f  20 E3 C7   JSR C7E3     écrit un secteur complet avec 256 octets [#00] encadrés par des octets de synchronisation,
                               n° piste, n° secteur etc), dans le tampon en 9800 + Y, en utilisant les valeurs de la 2ème partie de la table C67A. Selon le
                               nombre de secteurs par piste (16, 17, 18 ou 19), le nombre total d'octets écrits sera différent (356, 356, 346 ou 328
                               respectivement). Cette différence porte sur le nombre de "#4E" placés en fin de secteur.

C7A5f  C6 F6      DEC F6       décrémente le nombre de secteurs par piste
C7A7f  D0 F7      BNE C7A0     reboucle en C7A0 tant que F6 n'est pas nul
```

Elabore une fin de piste dans le tampon de formatage

Le s/p C7A9/C7B8 remplit toute la fin du tampon (jusqu'à l'adresse indiquée en C6A6/C6A7, c'est à dire B100) avec la valeur #4E.

```
7A9f  A9 4E       LDA #4E       A = #4E

7ABf  91 0A       STA (0A),Y    écrit #4E selon l'adresse en 0A/0B + Y
7ADf  C8          INY           indexe la position suivante
7AEf  D0 FB       BNE C7AB      et reboucle en C7AB tant que Y n'est pas nul
7B0f  E6 0B       INC 0B        page suivante
7B2f  A6 0B       LDX 0B        pour test
7B4f  EC A7 C6    CPX C6A7      teste si HH atteint la valeur en C6A7 (#B1)
7B7f  D0 F2       BNE C7AB      sinon, reboucle en C7AB jusqu'à ce que toute la fin du tampon de préparation de piste soit
```
remplie de "#4E".

Formate pour de bon

Le s/p C7B9/C7E2 envoie la commande #08 (positionnement sur piste #00) au s/p XRWTS (CFCD, routine de gestion des lecteurs). A l'aide d'une boucle, appelle le s/p C6F9 qui met à jour les n° de piste, de face et de secteur, envoie la commande #F0 (commande "formater une piste") au s/p XRWTS, affiche en décimal sur 2 digits le n° de la PISTE formatée (COO1 dont le b7 a été forcé à zéro) et ceci pour tous les secteurs de la face.

```
7B9f  A2 08       LDX #08       probablement pour positionnement ou initialisation
7BBf  20 CD CF    JSR CFCD      XRWTS routine de gestion des lecteurs, X = commande

7BEf  20 F9 C6    JSR C6F9      met à jour les n° de piste, de face et de secteur
7C1f  A2 F0       LDX #F0       probablement commande "formater une piste"
7C3f  20 75 DA    JSR DA75      XRWTS X = commande et traite une éventuelle erreur
7C6f  A9 08       LDA #08       A = "flèche gauche" pour reculer de 2 positions
7C8f  20 2A D6    JSR D62A      XAFCAR affiche le caractère ASCII contenu dans A
7CBf  20 2A D6    JSR D62A      idem
7CEf  A2 30       LDX #30       X = "0"
7D0f  8E 4C C0    STX C04C      DEFAFF, code ASCII devant les nombres décimaux
7D3f  AD 01 C0    LDA C001      n° de PISTE formatée à afficher
7D6f  29 7F       AND #7F       élimine le b7 indiquant la face
7D8f  20 4E D7    JSR D74E      affichage en décimal sur 2 digits d'un nombre A
7DBf  EE 01 C0    INC C001      n° de PISTE active suivante à écrire
7DEf  C6 F5       DEC F5        nombre de pistes par face
7E0f  D0 DC       BNE C7BE      et reboucle en C7BE tant qu'il en reste
7E2f  60          RTS
```

Copie une suite d'octets dans le tampon de formatage

A l'aide d'une boucle, le s/p C7E3/C7FF lit une paire d'octets AB dans la table C67A et copie A fois l'octet B dans le tampon de formatage au pointeur 0A/0B + Y et ceci jusqu'à ce que l'octet #FF soit rencontré. Par exemple, lors de l'élaboration d'un début de piste, le 1er nombre d'octets à copier est #28 (40), l'octet suivant #4E sera copié 40 fois à partir du début du tampon (Y = #00), puis l'octet #00 sera copié 12 fois (#0C), l'octet #F6 3 fois, l'octet #FC 1 fois et enfin l'octet #4E 40 fois. En tout, 96 octets (#60) seront mis en place dans le buffer de l'adresse 9800 à l'adresse 985F.

```
7E3f  BD 7A C6    LDA C67A,X    lecture d'un octet dans la table, à la position X.
7E6f  E8          INX           indexe la position suivante dans la table
7E7f  C9 FF       CMP #FF       l'octet lu est-il #FF?
7E9f  F0 13       BEQ C7FE      si oui, simple RTS (seule sortie de ce s/p)
7EBf  85 0C       STA 0C        sauve l'octet lu en 0C (nombre d'octets à copier)
7EDf  BD 7A C6    LDA C67A,X    lecture de l'octet suivant dans la table
7F0f  E8          INX           indexe la position suivante dans la table

7F1f  91 0A       STA (0A),Y    copie l'octet lu dans le buffer, à la position Y
7F3f  C8          INY           indexe la position suivante dans le buffer
7F4f  D0 02       BNE C7F8      saute l'instruction suivante tant que Y ne dépasse pas #FF
                                  (lorsque 256 octets ont été copiés, il faut indexer la page suivante)
7F6f  E6 0B       INC 0B        indexe la page suivante du buffer (incrémente HH)

7F8f  C6 0C       DEC 0C        décrémente le nombre d'octets à copier
7FAf  D0 F5       BNE C7F1      reboucle en C7F1 en tant qu'il en reste à copier
7FCf  F0 E5       BEQ C7E3      reboucle en C7E3 à chaque fois que le nombre voulu d'octets a été copié. Par exemple, le
```
premier nombre d'octets à copier était #28 (40) pour X = #00, l'octet suivant #4E sera copié 40 fois à partir du début du tampon (lorsque Y = #00), puis l'octet #00 sera copié 12 fois (#0C), l'octet #F6 3 fois, l'octet #FC 1 fois et enfin l'octet #4E 40 fois. En tout, 96 octets (#60) seront mis en place dans le buffer de l'adresse 9800 à l'adresse 985F (Y = #60 à la fin).

```
7FEf  60          RTS
```

C7FFf 00 BRK

Version 2.0 GB

La version 2.0 GB du Sédoric est basée sur l'utilisation d'un deuxième secteur de bitmap situé au 2ème secteur de la piste n°20. Ce secteur était déjà réservé, mais non utilisé.

Dans la banque 6, 19 octets ont été modifiés. Les sous-programmes suivants sont touchés:

1) Vérification de la validité du nombre total de secteur

2) Elaboration d'un deuxième secteur de bitmap

3) Correction de la fameuse bogue du flag "Double face".

Teste la validité de AY = nombre total de secteurs
Anciennement de 1919, le nombre maximal de secteurs par disquettes passe à 3840. Toutefois, les lecteurs actuellement sur le marché sont tous limités à 80 pistes par face, ce qui, à raison de 19 secteurs au maximum par piste, nous ammène à 3040 secteurs maxi, soit 760 kilo-octets et ne sera probablement réalisable qu'avec des lecteurs et des disquettes de type HD: bonjour la dépense! Par contre, les lecteurs et disquettes de type DD permettent 720 kilo-octets, ce qui n'est déjà pas si mal! Il faudra alors formater en double face de 80 pistes de 18 secteurs.

C510f	C0 **0F**	CPY #0F	teste si Y < #0F (soit AY < #0F00 donc < 3840)
C512f	90 05	BCC C519	si oui, continue en C519 (avec AY valable)
C514f	D0 9E	BNE C4B4	si Y > #0F (ce qui correspond à AY > #0FFF soit AY > 4095, continue en C4B4 ("ILLEGAL QUANTITY ERROR")
C516f	AA	TAX	si Y = #0F, teste si A > #00, c'est à dire si AY > #0F00 soit 3840 en décimal, ce qui est illégal...
C517f	**D0** 9B	BNE C4B4	"ILLEGAL QUANTITY ERROR" si AY > 3840
C519f	8D 02 C2	STA C202	ecrit le nombre total de secteurs dans BUF2
C51Cf	8C 03 C2	STY C203	aux positions #02/03 (nombre de secteurs libres)

Ce contrôle de validité est complètement inutile. En effet, puisque la validité du nombre de secteurs par piste et du nombre de pistes par face a été vérifiée, le maximum possible ne peut dépasser 19 x 99 x 2 = 3762 secteurs, ce qui est toujours inférieurs aux 3840 secteurs maximum. Il y a donc ici 9 octets récupérables...

Continue l'élaboration du secteur de bitmap
et adapte le 2ème secteur en attente en RAM
La fin de ce sous-programme (C576/C5B3) a été modifiée pour ajouter une sauvegarde de la deuxième bitmap. L'avant-dernière ligne:

C5AEf **8E 15 31** STX 3115 résultat à la position #15 du 2ème secteur en RAM
a été remplacé par: C5AEf 20 38 F6 JSR F638 En F638, dans la version 1.006, se trouvait une petite serie de NOP qui a été remplacée par un micro sous-programme dans la version 2.0 GB:

F638-	**8E 15 31**	STX 3115	c'est la même chose qu'avant, mais on a ajouté
F63B-	**4C 4A FF**	JMP FF4A	cet appel à une nouvelle petite routine en FF4A, placée dans l'ancienne table des vecteurs système:
FF4A-	**A0 03**	LDY #03	pour secteur n°3 de la piste n°20
FF4C-	**4C 8B DC**	JMP DC8B	qui lui-même est situé au coeur de l'ancienne routine XSMAP qui a été modifiée de la manière suivante:
DC8B-	**A9 14**	LDA #14	pour indiquer la piste n°20
DC8D-	**4C 8E DA**	JMP DA8E	qui reprend le cours normal du s/p XSMAP: sauve le deuxième secteur de bitmap au secteur n°3 de la piste n°20 et retourne (en C5B1) pour terminer le s/p d'où l'on était parti (C576/C5B3). Beaucoup de petits bricolages donc, mais c'est bien joué et ça marche! Bravo Ray.

Termine l'élaboration du secteur de bitmap et le sauve
La fin de ce sous-programme (C5E7/C5FB) a été modifiée pour palier à des changements intervenus dans la routine XSMAP. La dernière ligne:

C5F9f 20 8A DA JSR DA8A XSMAP sauve secteur de bitmap sur la disquette
a été remplacée par: C5F9f 20 **89 DC** JSR DC89 routine qui consiste en:

DC89-	A0 02	LDY #02	pour indiquer le secteur n°2
DC8B-	**A9 14**	LDA #14	pour indiquer la piste n°20
DC8D-	**4C 8E DA**	JMP DA8E	qui reprend le cours normal du s/p XSMAP: sauve le premier secteur de bitmap au secteur n°2 de la piste n°20 et retourne (en C5FC) pour terminer le s/p d'où l'on était parti (C5E7/C5FB).

Une autre? (même format)
La fin de ce sous-programme (C5FC/C60B) a été modifiée pour palier aux modifications de bitmap intervenus ci-dessus (les deux bitmaps ne sont pas identiques, la deuxième (qui est d'ailleur sauvée en premier) n'ayant pas les secteurs réservés de la piste n°20. La dernière ligne de ce s/p:

609f 4C 1F C5 JMP C51F reprend en C51F pour formatage identique
à été remplacée par: C609f 4C **DE C4** JMP C4DE La reprise est maintenant effectuée tout au début de l'élaboration des bitmaps.

Formate la ou les faces

Première partie de la correction de la fameuse bogue de INIT:

64Af	**AD** AE C6	LDA C6AE	reprend C6AE sans en modifier le contenu
64Df	**29 7F**	AND #7F	calcule dans A le nombre de pistes par face en forçant simplement le b7 à zéro: c'en est fini de la bogue!
64Ff	**EA**	NOP	de temps en temps c'est agreable de ne rien faire!
650f	8D AF C6	STA C6AF	C6AF = #00 si une face et #01 si deux faces
653f	A9 B0	LDA #B0	AY = adresse C6B0 de la chaîne:
655f	A0 C6	LDY #C6	"CRLFLFFormating Side 0 Track 00"
657f	20 37 D6	JSR D637	AFSTR affiche chaîne terminée par 0 d'adresse AY
65Af	20 40 D7	JSR D740	"CURSEUR OFF" (curseur caché = vidéo normale)
65Df	**EA EA EA**	NOP NOP NOP	...surtout si cela se répète!!!
660f	20 45 C7	JSR C745	formate la première face (car C vaut toujours 0)
663f	AD **AE** C6	LDA C6AE	teste si une seule face (b7 à zéro)
666f	**10** 0B	BPL C673	si oui, continue en C673
668f	A9 CD	LDA #CD	sinon, AY = adresse C6CD pour chaîne:
66Af	A0 C6	LDY #C6	"<- <- <- <- <- <- <- <- <- <- 1 Track 00"
66Cf	20 37 D6	JSR D637	AFSTR affiche chaîne terminée par 0 d'adresse AY
66Ff	38	SEC	pour deuxième face
670f	20 45 C7	JSR C745	formate la deuxième face
673f	A9 E2	LDA #E2	AY = adresse C6E2 pour chaîne:
675f	A0 C6	LDY #C6	"LFLFCRFormating complete"
677f	4C 37 D6	JMP D637	AFSTR affiche chaîne d'adresse AY et retourne

Formate la première face si C = 0 et la deuxième si C = 1
Cette grande routine C745/C7FD a subit une petite modification en C76B qui constitue la deuxième partie (modeste) de la correction de la bogue du flag "double face". La ligne suivante a été modifiée:

76Af AD AE C6 LDA C6AE nombre de pistes/face et nombre de faces
pour devenir tout simplement C76Af AD **AF** C6 LDA C6AF nombre de faces

*** DÉBUT DU NOYAU PERMANENT DE SÉDORIC *** (#C800 À #FFFF)

Table "KEYDEF"

Affectation de codes de fonctions (#00 à #FF) aux codes de touches (#80 à #BF). L'index d'entrée dans la table va de #00 à #3F pour les combinaisons FUNCT+touche et de #40 à #7F pour les combinaisons FUNCT+SHIFT+touche.

```
      FUNCT+touche
C800- 09 AA F6 F1  17 8C AC 07   7 J M K   ■ U Y 8
C808- 90 C9 0F 0A  00 99 DC F4   N T 6 9   , I H L
C810- 05 8B B1 00  00 B4 97 0B   5 R B ;   . O G 0
C818- EB 8D 03 1C  81 E6 C8 84   V F 4 -   ↑ P E /
C820- 00 00 00 00  00 00 00 00   m m c m   g f m s
C828- 00 1C 09 00  BF FE D1 FF   1 e Z m   ← d A r
C830- B6 A7 01 12  C1 1D EA 00   X Q 2 \   ↓ ] S m   FUNCT+] = DIR suivi de CR
C838- 02 E7 94 0E  BC 0D AE 13   3 D C '   → [ W =
      FUNCT+SHIFT+touche
C840- 10 AB A8 BD  00 D9 AD 11   7 J M K   ■ U Y 8
C848- CA C3 00 00  00 92 9E F5   N T 6 9   , I H L
C850- 00 9C B2 00  00 D2 9B 0C   5 R B ;   . O G 0
C858- E9 AF 00 00  BB B9 80 85   V F 4 -   ↑ P E /
C860- 00 00 00 00  00 00 00 00   m m c m   g f m s
C868- 08 00 91 00  00 00 EC FF   1 e Z m   ← d A r
C870- B7 A9 04 06  00 00 F2 00   X Q 2 \   ↓ ] S m   FUNCT+SHIFT+\ = LIST1000- et CR
C878- 09 8A ED 00  C7 0C B0 1C   3 D C '   → [ W =
```

Avec: c CTRL, d DEL, e ESC, f FUNCT, g SHIFT Gauche, m code manquant, r RETURN, s SHIFT DROIT, et ■ SPACE.
Exemples: la touche FUNCT et la touche] permettent d'obtenir directement l'instruction DIR et un RETURN (commande prédéfinie n°#1D, voir ci-dessous), déclanchant alors l'affichage du catalogue à l'écran. De même, à la combinaison FUNCT+SHIFT+\ est affectée le code de fonction #06 corresp à la commande redéfinissable "LIST1000-" suivie de RETURN. Les nombreux #00 de cette table correspondent à "?HEX$(DEEK(#".

Codes de fonctions

1ère serie: 16 commandes redéfinissables avec KEYUSE (USER FUNCTIONS, code 0 à 15)

```
C880- 00   20 20 20 20 3F 48 45 58 24 28 44 45 45 4B 28 A3   ■■■■?HEX$(DEEK(#
C890- 01   20 20 50 41 50 45 52 20 30 3A 49 4E 4B 20 37 8D   ■■PAPER 0:INK 7CR
C8A0- 02   20 44 4F 4B 45 20 23 32 34 45 2C 23 31 30 38 8D   ■DOKE #24E,#108CR
C8B0- 03   44 4F 4B 45 20 23 32 34 35 2C 23 45 45 32 32 8D   DOKE #245,#EE22CR
C8C0- 04   20 20 20 20 20 20 20 20 20 20 52 45 53 45 54 8D   ■■■■■■■■■■RESETCR
C8D0- 05   20 20 44 4F 4B 45 23 32 34 35 2C 23 34 38 34 8D   ■■DOKE#245,#484CR
C8E0- 06   20 20 20 20 20 20 4C 49 53 54 31 30 30 30 2D 8D   ■■■■■■LIST1000-CR
C8F0- 07   20 20 20 20 20 52 45 4E 55 4D 20 31 30 30 30 8D   ■■■■■RENUM 1000CR
C900- 08   20 20 44 4F 4B 45 23 32 46 35 2C 23 34 36 33 8D   ■■DOKE#2F5,#463CR
C910- 09   20 20 20 20 20 20 43 41 4C 4C 23 46 38 44 30 8D   ■■■■■■CALL#F8D0CR
C920- 0A   52 45 4E 55 4D 31 30 30 30 2C 2C 31 30 30 30 8D   RENUM1000,,1000CR
C930- 0B   20 20 20 20 20 20 20 20 4E 55 4D 20 45 4E 44 8D   ■■■■■■■■NUM ENDCR
C940- 0C   20 20 20 20 20 20 20 20 20 20 20 20 53 41 56 45 A2   ■■■■■■■■■■■■SAVE"
C950- 0D   20 20 20 20 20 20 20 20 20 20 20 53 41 56 45 55 A2   ■■■■■■■■■■■SAVEU"
C960- 0E   20 20 20 20 20 20 20 20 20 20 20 20 20 45 58 54 8D   ■■■■■■■■■■■■■EXTCR
C970- 0F   20 44 4F 4B 45 23 32 33 43 2C 23 45 42 37 38 8D   ■DOKE#23C,#EB78CR
```

2ème serie: 16 commandes prédéfinies (code 16 à 31, manuel Sédoric page 102)

```
C980- 10   48 49 52 45 53 8D                                                  HIRESCR
C986- 11   54 45 58 54 8D                                                      TEXTCR
C98B- 12   4C 49 53 54 8D                                                      LISTCR
C990- 13   52 55 4E 8D                                                          RUNCR
C994- 14   4C 50 52 49 4E 54 8D                                              LPRINTCR
C99B- 15   46 4F 52 20 49 3D 31 20 54 4F A0                                FOR I=1 TO■
C9A6- 16   43 55 52 53 45 54 20 31 32 30 2C 31 30 30 2C 31 8D    CURSET 120,100,1CR
C9B7- 17   01 01 01 01 01 81                  CTRL/ACTRL/ACTRL/ACTRL/ACTRL/ACTRL/A
C9BD- 18   45 58 54 8D                                                        EXTCR
C9C1- 19   4E 55 4D 20 45 4E 44 8D                                         NUM ENDCR
C9C8- 1A   53 45 45 4B 8D                                                    SEEKCR
C9CE- 1B   52 45 4E 55 4D 8D                                                RENUMCR
C9D4- 1C   4F 4C 44 8D                                                        OLDCR
```

```
9D8- 1D       44 49 52 8D                                          DIRCR
9DC- 1E       94                                                   CTRL/T
9DD- 1F       E0                                                   CHR$(96)
```

Mots-clés du dos (codes 32 à 127)
(avec adresse d'exécution selon zone #CC27 voir plus loin)

```
C9DE- 50 50 80 00             APPEND        #20=032    FE07 ;#80 = code BASIC "END"
C9E2- 50 50 45 4E 44 00       APPEND        #21=033    FE07
C9E8- 5A 45 52 54 59 00       AZERTY        #22=034    EBDE
C9EE- 43 43 45 4E 54 00       ACCENT        #23=035    EB91

C9F4- 4F 58 00                BOX           #24=036    F0DE
C9F7- 41 43 4B 55 50 00       BACKUP        #25=037    F151
C9FD- 55 49 4C 44 00          BUILD         #26=038    FEE0

CA02- 48 41 4E 47 45 00       CHANGE        #27=039    F148
CA08- 4C 4F 53 45 00          CLOSE         #28=040    FB8D
CA0D- 4F 50 59 00             COPY          #29=041    F157
CA11- 52 45 41 54 45 57 00 CREATEW          #2A=042    DE4D
CA18- 52 45 53 45 43 00       CRESEC        #2B=043    F9BC

CA1E- 45 96 45 00             DELETE        #2C=044    F142 ;#96 = code BASIC "LET"
CA22- 45 4C 45 54 45 00       DELETE        #2D=045    F142
CA28- 45 53 54 52 4F 59 00 DESTROY          #2E=046    E444
CA2F- 45 4C 42 41 4B 00       DELBAK        #2F=047    E437
CA35- 45 4C 00                DEL           #30=048    E446
CA38- 49 52 00                DIR           #31=049    E344
CA3B- 54 52 41 43 4B 00       DTRACK        #32=050    F139
CA41- 4E 55 4D 00             DNUM          #33=051    F12A
CA45- 4E 41 4D 45 00          DNAME         #34=052    F145
CA4A- 4B 45 59 00             DKEY          #35=053    F124
CA4E- 53 59 53 00             DSYS          #36=054    F127
CA52- 54 52 41 43 4B 00       DTRACK        #37=055    F139

CA58- 52 52 97 00             ERRGOTO       #38=056    E999 ;#97 = code BASIC "GOTO"
CA5C- 52 52 47 4F 54 4F 00 ERRGOTO          #39=057    E999
CA63- 52 52 4F 52 00          ERROR         #3A=058    E9B0
CA68- 52 52 D2 00             ERROR         #3B=059    E9B0 ;#D2 = code BASIC "OR"
CA6C- 52 52 00                ERR           #3C=060    E97F
CA6F- 53 41 56 45 00          ESAVE         #3D=061    DDE0
CA74- 58 54 00                EXT           #3E=062    E9ED

CA77- 49 45 4C 44 00          FIELD         #3F=063    FBBF
CA7C- 52 53 45 43 00          FRSEC         #40=064    F99C

CA81- 43 55 52 00             HCUR          #41=065    EBF5

CA85- 4E 49 54 00             INIT          #42=066    F169
CA89- 4E 53 54 52 00          INSTR         #43=067    EC2E
CA8E- 4E 49 53 54 00          INIST         #44=068    F12D

CA93- 55 4D 50 00             JUMP          #45=069    FE12

CA97- 45 59 99 00             KEYIF         #46=070    DA20 ;#99 = code BASIC "IF"
CA9B- 45 59 49 46 00          KEYIF         #47=071    DA20
CAA0- 45 59 55 53 45 00       KEYUSE        #48=072    D9B0
CAA6- 45 59 44 45 46 00       KEYDEF        #49=073    D9FD
CAAC- 45 59 B8 00             KEYDEF        #4A=074    D9FD ;#B8 = code BASIC "DEF"
CAB0- 45 59 53 41 56 45 00 KEYSAVE          #4B=075    DDCD
CAB7- 45 59 00                KEY           #4C=076    E70B

CABA- 49 4E 45 00             LINE          #4D=077    F079
CABE- 53 45 54 00             LSET          #4E=078    FC73
CAC2- 55 53 49 4E 47 00       LUSING        #4F=079    F036
CAC8- 55 E3 47 00             LUSING        #50=080    F036 ;#E3 = code BASIC "SIN"
CACC- 92 00                   LINPUT        #51=081    EC94 ;#92 = code BASIC "INPUT"
CACE- 49 4E 50 55 54 00       LINPUT        #52=082    EC94
CAD4- 4F 41 44 00             LOAD          #53=083    DFF7
CAD8- 44 49 52 00             LDIR          #54=084    E7D0
CADC- 54 59 50 45 00          LTYPE         #55=085    FE95
CAE1- 43 55 52 00             LCUR          #56=086    EBEC

CAE5- 4F 56 45 00             MOVE          #57=087    F136
CAE9- 45 52 47 45 00          MERGE         #58=088    F13C
```

```
AEE- 55 4D 00              NUM           #59=089     EB25

AF1- 55 54 00              OUT           #5A=090     E71F
AF4- 4C 44 00              OLD           #5B=091     E0AF
AF7- 50 45 4E 00           OPEN          #5C=092     FA50

AFB- 55 54 00              PUT           #5D=093     F9CB
AFE- 52 4F 54 00           PROT          #5E=094     E6D0
B02- 52 00                 PR       #5F=095     E7C0
B04- 4D 41 50 00           PMAP          #60=096     F990

B08- 55 49 54 00           QUIT          #61=097     E7F5
B0C- 57 45 52 54 59 00     QWERTY        #62=098     EBE1

B12- 45 53 55 4D 45 00     RESUME        #63=099     E9BB
B18- 53 45 54 00           RSET          #64=100     FC75
B1C- 45 57 49 4E 44 00     REWIND        #65=101     FABB
B22- 45 4E 55 4D 00        RENUM         #66=102     F14E
B27- 45 4E 00              REN           #67=103     E537
B2A- D1 4F 4D 00           RANDOM        #68=104     E796 ;#D1 = code BASIC "AND"
B2E- 41 4E 44 4F 4D 00     RANDOM        #69=105     E796
B34- 45 53 54 4F 52 45 00 RESTORE        #6A=106     E7D9
B3B- 45 53 45 54 00        RESET         #6B=107     E7B8

B40- 57 41 50 00           SWAP          #6C=108     EA3B
B44- 45 45 4B 00           SEEK          #6D=109     F154
B48- 54 52 55 4E 00        STRUN         #6E=110     E853
B4D- 54 98 00              STRUN         #6F=111     E853 ;#98 = code BASIC "RUN"
B50- 59 53 54 45 4D 00     SYSTEM        #70=112     E702
B56- 54 41 54 55 53 00     STATUS        #71=113     E62E
B5C- 41 56 45 55 00        SAVEU         #72=114     DD4D
B61- 41 56 45 4D 00        SAVEM         #73=115     DD4A
B66- 41 56 45 4F 00        SAVEO         #74=116     DD53
B6B- 41 56 45 00           SAVE          #75=117     DD50
B6F- 45 41 52 43 48 00     SEARCH        #76=118     E5FC
B75- 59 53 00              SYS           #77=119     F15A
B78- 4D 41 50 00           SMAP          #78=120     F996

B7C- 4B 45 4E 00           TKEN          #79=121     E89D
B80- 41 4B 45 00           TAKE          #7A=122     F8DF
B84- 59 50 45 00           TYPE          #7B=123     FE98
B88- 52 41 43 4B 00        TRACK         #7C=124     F130

B8D- 53 45 52 00           USER          #7D=125     EA7F
B91- 4E 54 4B 45 4E 00     UNTKEN        #7E=126     E8E1
B97- E3 47 00              USING         000         EE99 ;#E3 = code BASIC "SIN"
B9A- 53 49 4E 47 00        USING         000         EE99
B9F- 4E 50 52 4F 54 00     UNPROT        000         E6D3

BA5- 55 53 45 52 00        VUSER         000         F121

BAA- 49 44 54 48 00        WIDTH         000         E740
BAF- 49 4E 44 4F 57 00     WINDOW        000         F210

BB5- 9A 00                 RESTORE       000         E7D9 ;#9A = code BASIC "RESTORE"

BB7- 5D 00                 ]             000         EC04 ;#5D = code ASCII "]"

BB9- FF 00                 255           000         E83E
```

NB: Les mots-clés qui comportent un token BASIC sont codés de deux manières différentes. Effectivement, si l'utilisateur a choisi de taper tous les mots du Sédoric en minuscules, pour les distinguer des mots du BASIC, l'encodage ne sera pas réalisé par la ROM en ECB9 et chaque lettre du mot sera alors significative. On peut d'ailleurs ici faire la remarque suivante: il est conseillé à l'utilisateur de taper son texte en majuscules, car de la sorte les mots sont raccourcis et l'analyse syntaxique ultérieure par RAMOV et donc l'exécution en sont accélérées.

Sous-table selon la 1ère lettre du mot-clé Sédoric

	reg A	1ère lettre	adresse (hexa)	n°ordre (décimal)	nombre (décimal)
CBBB- DE C9 00 04	00	A	C9DE	00	04
CBBF- F4 C9 04 03	01	B	C9F4	04	03
CBC3- 02 CA 07 05	02	C	CA02	07	05
CBC7- 1E CA 0C 0C	03	D	CA1E	12	12
CBCB- 58 CA 18 07	04	E	CA58	24	07
CBCF- 77 CA 1F 02	05	F	CA77	31	02
CBD3- CC CC 21 00	06	G	CCCC	33	00
CBD7- 81 CA 21 01	07	H	CA81	33	01
CBDB- 85 CA 22 03	08	I	CA85	34	03
CBDF- 93 CA 25 01	09	J	CA93	37	01
CBE3- 97 CA 26 07	0A	K	CA97	38	07
CBE7- BA CA 2D 0A	0B	L	CABA	45	10
CBEB- E5 CA 37 02	0C	M	CAE5	55	02
CBEF- EE CA 39 01	0D	N	CAEE	57	01
CBF3- F1 CA 3A 03	0E	O	CAF1	58	03
CBF7- FB CA 3D 04	0F	P	CAFB	61	04
CBFB- 08 CB 41 02	10	Q	CB08	65	02
CBFF- 12 CB 43 09	11	R	CB12	67	09
CC03- 40 CB 4C 0D	12	S	CB40	76	13
CC07- 7C CB 59 04	13	T	CB7C	89	04
CC0B- 8D CB 5D 05	14	U	CB8D	93	05
CC0F- A5 CB 62 01	15	V	CBA5	98	01
CC13- AA CB 63 02	16	W	CBAA	99	02
CC17- CC CC 65 00	17	X	CCCC	101	00
CC1B- CC CC 65 00	18	Y	CCCC	101	00
CC1F- CC CC 65 00	19	Z	CCCC	101	00
CC23- B5 CB 65 03	1A	autre CBB5	101	03	

Table des adresses d'exécution des mots-clés Sédoric (ADR-1)
(regroupées par initiale des mots-clés, sous la forme LL puis HH)

```
CC27- A         06FE/06FE/DDEB/90EB
CC2F- B         DDF0/50F1/DFFE

CC35- C         47F1/8CFB/56F1/4CDE/BBF9
CC3F- D         41F1/41F1/43E4/36E4/45E4/43E3/38F1/29F1/44F1/23F1/26F1/38F1

CC57- E         98E9/98E9/AFE9/AFE9/7EE9/DFDD/ECE9
CC65- F         BEFB/9BF9

CC69- H         F4EB

CC6B- I         68F1/2DEC/2CF1
CC71- J         11FE

CC73- K         1FDA/1FDA/AFD9/FCD9/FCD9/CCDD/0AE7
CC81- L         78F0/72FC/35F0/35F0/93EC/93EC/F6DF/CFE7/94FE/EBEB

CC95- M         35F1/3BF1
CC99- N         24EB

CC9B- O         1EE7/AEE0/4FFA
CCA1- P         CAF9/CFE6/BFE7/8FF9

CCA9- Q         F4E7/E0EB
CCAD- R         BAE9/74FC/BAFA/4DF1/36E5/95E7/95E7/D8E7/B7E7

CCBF- S         3AEA/53F1/52E8/52E8/01E7/2DE6/4CDD/49DD/52DD/4FDD/FBE5/59F1/95F9
CCD9- T         9CE8/DEF8/97FE/2FF1

CCE1- U         7EEA/E0E8/98EE/98EE/D2E6
CCEB- V         20F1

CCED- W         3FE7/0FF2

CCF1- autre     D8E7/03EC/3DED
```

Table NOM et EXTENSION par défaut

CF7-	43 4F 4D	COM extension courante
CFA-	42 41 4B	BAK extension pour SAVEU
CFD-	43 4F 4D	COM extension par défaut
D00-	3F 3F 3F 3F 3F 3F 3F 3F 3F	????????? Joker * ou NFA omis
D09-	42 41 4B	BAK (ne semble pas utilisée)

Table de constantes diverses

D0C-	28 50 35 5D	valeurs par défaut pour la commande WIDTH
D10-	00 00 01 01 FA BF 23 34 36 37 FF	utilisé par la commande STRUN (CALL#467 #FF)
D1B-	7B 0E FA 35 10	utilisé par les commandes LINE et BOX (0,174532925)
D20-	81 C9 0F DA A2	utilisé par la commande LINE et BOX (-1,57079633)
D25-	C6 C9 88 02 88 02	utilisé par la commande OPEN (initialisation de **FI**)
D2B-	4F 46 46	OFF
D2E-	53 45 54	SET
D31-	C7 81 C2 82 45 D3 66 A5 C8 A3 8F D2 42 B5 98 E0	non identifié

Table de conversion QWERTY / AZERTY

D41-	B1 BE AE AA 82 93	codes des caractères ; M Z A W Q
D47-	AE AA B1 BE 93 82	codes des caractères M ; W Q Z A

Table de convertion ACCENT OFF / ACCENT SET

D4D-	40 10 08 1C 02 1E 22 1E 00	code #40	@ -> à
D56-	5C 00 00 1E 20 20 20 1E 04	code #5C	\ -> ç
D5F-	7B 04 08 1C 22 3E 20 1E 00	code #7B	{ -> é
D68-	7C 10 08 22 22 22 26 1A 00	code #7C	| -> ù
D71-	7D 10 08 1C 22 3E 20 1E 00	code #7D	} -> è
D7A-	7E 1C 22 1C 22 3E 20 1E 00	code #7E	~ -> ê

Table de constantes diverses

D83-	41 58 59 50 B8	A X Y P et DEF pour la commande USER
D88-	0A 64 E8 10	LL des valeurs 10, 100, 1000 et 10000
D8C-	00 00 03 27	HH des valeurs 10, 100, 1000 et 10000
D90-	84 A4 C4 E4	table de codes selon drive actif pour la routine XRWTS

Variables réservées par le système

(Le numéro d'ordre de #00 à #24 est indiqué à gauche après l'adresse)

D94-	00	45 4E	EN	Error Number (mise à jour après une erreur)
D96-	02	45 4C	EL	Error Line (mise à jour après une erreur)
D98-	04	49 4E	IN	mise à jour par INstr (position de la sous-chaîne dans la chaîne)
D9A-	06	4F 4D	OM	Output Mode (mode de sortie de LINPUT)
D9C-	08	53 4B	SK	SeeK (nombre d'occurences de la chaîne trouvé par SEEK)
D9E-	0A	46 54	FT	File Type (type fichier chargé) (mis à jour par LOAD)
DA0-	0C	45 4F	EO	variable non identifiée
DA2-	0E	52 41	RA	affiche Registre A du microprocesseur (commande USER)
DA4-	10	52 58	RX	affiche Registre X du microprocesseur (commande USER)
DA6-	12	52 59	RY	affiche Registre Y du microprocesseur (commande USER)
DA8-	14	52 50	RP	affiche Registre d'état du microproc. (commande USER)
DAA-	16	45 46	EF	Existing File (si le fichier existe EF = 1 , sinon EF = 0)
DAC-	18	53 54	ST	STart adress (adresse de début du fichier) (LOAD)
DAE-	1A	45 44	ED	EnD adress (adresse de fin du fichier) (LOAD)
DB0-	1C	45 58	EX	EXecution adress (adresse d'exécution du fichier) (LOAD)
DB2-	1E	43 58	CX	Curseur X (abscisse du curseur TEXT ou HIRES) (HCUR et LCUR)
DB4-	20	43 59	CY	Curseur Y (ordonnée du curseur TEXT ou HIRES) (HCUR et LCUR)
DB6-	22	46 50	FP	Free Piste (n° de piste du secteur libéré par PRESEC)
DB8-	24	46 53	FS	Free Secteur (n° du secteur libéré par PRESEC)
DBA-	53 43 4A 4B 45			table des paramètres de LINPUT: S, C, J, K et E

NB: Il manque AN, angle courant pour les instructions graphiques. Mais comme le montre l'exemple de la page 67 du manuel, l'utilisateur doit lui-même créer la variable AN et lui donner une valeur en degrès. AN ne semble donc pas être comptée parmi les variables système, bien que les commandes LINE et BOX l'utilisent implicitement (voir le code en F054).

Messages Sédoric (zone CDBF)
(Le numéro d'ordre X est indiqué à gauche sous l'adresse)

CDBF- 46 49 4C 45 20 4E 4F 54 20 46 4F 55 4E **C4**
01 FILE NOT FOUN**D**

CDCD- 44 52 49 56 45 20 4E 4F 54 20 49 4E 20 4C 49 4E **C5**
02 DRIVE NOT IN LIN**E**

CDDE- 49 4E 56 41 4C 49 44 20 46 49 4C 45 20 4E 41 4D **C5**
03 INVALID FILE NAM**E**

CDEE- 44 49 53 4B 20 49 2F **CF**
04 DISK I/**O**

CDF7- 57 52 49 54 45 20 50 52 4F 54 45 43 54 45 **C4**
05 WRITE PROTECTE**D**

CE06- 57 49 4C 44 43 41 52 44 28 53 29 20 4E 4F 54 20 41 4C 4C 4F 57 45 **C4**
06 WILDCARD(S) NOT ALLOWE**D**

CE1D- 46 49 4C 45 20 41 4C 52 45 41 44 59 20 45 58 49 53 54 **D3**
07 FILE ALREADY EXIST**S**

CE30- 44 49 53 4B 20 46 55 4C **CC**
08 DISK FUL**L**

CE39- 49 4C 4C 45 47 41 4C 20 51 55 41 4E 54 49 54 **D9**
09 ILLEGAL QUANTIT**Y**

CE49- 53 59 4E 54 41 **D8**
0A SYNTA**X**

CE4F- 55 4E 4B 4E 4F 57 27 4E 20 46 4F 52 4D 41 **D4**
0B UNKNOW'N FORMA**T**

CE5E- 54 59 50 45 20 4D 49 53 4D 41 54 43 **C8**
0C TYPE MISMATC**H**

CE6B- 46 49 4C 45 20 54 59 50 45 20 4D 49 53 4D 41 54 43 **C8**
0D FILE TYPE MISMATC**H**

CE7D- 46 49 4C 45 20 4E 4F 54 20 4F 50 45 **CE**
0E FILE NOT OPE**N**

CE8A- 46 49 4C 45 20 41 4C 52 45 41 44 59 20 4F 50 45 **CE**
0F FILE ALREADY OPE**N**

CE9B- 45 4E 44 20 4F 46 20 46 49 4C **C5**
10 END OF FIL**E**

CEA6- 42 41 44 20 52 45 43 4F 52 44 20 4E 55 4D 42 45 **D2**
11 BAD RECORD NUMBE**R**

CEB7- 46 49 45 4C 44 20 4F 56 45 52 46 4C 4F **D7**
12 FIELD OVERFLO**W**

CEC5- 53 54 52 49 4E 47 20 54 4F 4F 20 4C 4F 4E **C7**
13 STRING TOO LON**G**

CED4- 55 4E 4B 4E 4F 57 27 4E 20 46 49 45 4C 44 20 4E 41 4D **C5**
14 UNKNOW'N FIELD NAM**E**

Messages Sédoric (zone CEE7)
(Le numéro d'ordre X est indiqué à gauche sous l'adresse)

EE7- 0A 0D 54 52 41 43 4B **BA**
LFCRTRACK**:**

EEF- 20 53 45 43 54 4F 52 **BA**
■SECTOR**:**

EF7- 20 57 52 49 54 45 20 46 41 55 4C 54 **A0**
■WRITE FAULT■

F04- 20 52 45 41 44 20 46 41 55 4C 54 **A0**
■READ FAULT■

F10- 0A 0D 20 42 52 45 41 4B 20 4F 4E 20 42 59 54 45 20 **A3**
LFCRBREAK ON BYTE **#**

F22- 0D 0A 44 72 69 76 65 **A0**
CRLFDrive■

F2A- 20 28 4D 61 73 74 65 72 29 **A0**
■(Master)■

F34- 20 73 65 63 74 6F 72 73 20 66 72 65 65 20 **A8**
■sectors free (

F43- 20 46 69 6C 65 73 **A0**
■Files■

F4A- 20 49 53 20 50 52 4F 54 45 43 54 45 **C4**
■IS PROTECTE**D**

F57- 20 28 59 29 65 73 20 6F 72 20 28 4E 29 6F **BA**
■(Y)es or (N)o**:**

F66- 20 44 45 4C 45 54 45 44 0D **8A**
■DELETED CR **LF**

F70- 49 4E 53 45 52 54 20 4D 41 53 54 45 52 **A0**
INSERT MASTER■

F7E- 41 4E 44 20 50 52 45 53 53 20 27 52 45 54 55 52 4E **A7**
AND PRESS 'RETURN'

F90- 20 41 4C 52 45 41 44 59 20 45 58 49 53 54 53 0A **8D**
■ALREADY EXISTSLF**CR**

FA1- 20 2D 2D 3E **A0**
■-->■

FA6- 55 53 45 52 **A0**
USER■

FAB- 20 28 53 6C 61 76 65 20 29 **A0**
■(Slave)■

FB5- 20 28 54 79 70 65 **BD**
■(Type**=**

FBC- 29 **A0**
)■

FBE- 20 44 49 53 43 20 49 4E 20 44 52 49 56 45 **A0**
■DISC IN DRIVE■

Rappel: CR = Carriage Return (retour chariot), place le curseur en début de ligne
LF = Line Feed, le curseur descend d'une ligne vers le bas

XRWTS ROUTINE DE GESTION DES LECTEURS

En entrée, X contient le n° de commande. En sortie, Z = 1 si pas d'erreur, sinon Z = 0 (notamment, si disquette protégée en écriture V = 1). DRIVE (C000), PISTE (C001), SECTEUR (C002) et RWBUF (C003/C004) doivent être à jour.

Variables et registres utilisés

0310- reçoit X, code de commande: #18 (positionne tête), #88 (lecture), #A8 (écriture), #C0 (activation drive selon C00B)
0311- piste sous la tête
0312-
0313- registre DATA pour lecture/écriture disquette
0314- reçoit 04FB = code DRIVE et FACE
0315-
0316-
0317-
0318- Ready
04FB- code DRIVE et FACE
C005- X = code de commande (voir annexe G pour de plus amples détails)
-Restore (#08) positionne la tête sur la piste #00
-Seek (#18) positionne la tête et met à jour le "Track Register"
-Read Sector (#80 ou #88) lit un secteur sans tester le n° de face
-Write Sector (#A0 ou #A8) écrit un secteur sans tester le n° de face
-Read Address (#C0) le FD1793 lit le prochain champ ID (n° de piste, n° de face, n° de secteur, taille du secteur, CRC1 et CRC2), vérifie la validité
-Read Track (#E0) tous les octets de gaps, en-têtes et data sont lus
-Write Track (#F0) formate une piste
C006- 1 ou 2
C007- 8
C008- 7
C017- "bilan" de l'I/O: #00 si pas d'erreur, b6=1 si protection écriture
C060- RWBUF lors de l'activation du DRIVE visé

En raison du manque d'informations disponibles sur les variables utilisées dans la page 3, le désassemblage de la routine XRWTS est parfois imprécis, veuillez nous en excuser.

Début de la routine XRWTS

CFCD-	08	PHP	sauvegarde les indicateurs du 6502
CFCE-	AD 0E 03	LDA 030E	sauvegarde le contenu de 030E (VIAIER,
CFD1-	48	PHA	registre d'autorisation d'interruption)
CFD2-	98	TYA	sauvegarde Y
CFD3-	48	PHA	
CFD4-	A9 40	LDA #40	masque 0100 0000, force b6 de VIAIER à 1 et
CFD6-	8D 0E 03	STA 030E	b7 à 0: interdit interruption T1
CFD9-	20 E9 CF	JSR CFE9	routine XRWTS proprement dite
CFDC-	68	PLA	
CFDD-	A8	TAY	récupère la valeur Y d'origine
CFDE-	68	PLA	
CFDF-	8D 0E 03	STA 030E	récupère la valeur 030E (VIAIER) de d'origine
CFE2-	28	PLP	récupère les indicateurs 6502
CFE3-	A9 FF	LDA #FF	masque 1111 1111 pour test C017, c'est
CFE5-	2C 17 C0	BIT C017	à dire positionne Z, N et V selon "bilan"
CFE8-	60	RTS	et retourne

Entrée principale de la routine XRWTS proprement dite

CFE9-	A0 02	LDY #02	C006 est mis à #02 (variable locale XRWTS)

Entrée secondaire pour rebouclage

CFEB-	8C 06 C0	STY C006	mise à jour variable locale avec Y courant
CFEE-	A0 08	LDY #08	dans tous les cas, C007
CFF0-	8C 07 C0	STY C007	(variable locale XRWTS) est mis à #08

Point d'entrée réel: exécution de la commande X

CFF3-	48	PHA	sauvegarde de la valeur de A sur pile
CFF4-	8E 05 C0	STX C005	et sauve X en C005 (code de commande à exécuter)
CFF7-	AC 00 C0	LDY C000	n° du DRIVE cible (de 0 à 3)
CFFA-	B9 90 CD	LDA CD90,Y	lit valeur corresp dans table CD90 soit 1**00**0 0100 pour A, 1**01**0 0100 pour B, 1**10**0 0100 pour C et 1**11**0 0100 pour D, c'est à dire #84, #A4, #C4 et #E4 respectivement. On remarque que les b6/b5 portent le

numéro de drive (valeurs 0, 1, 2 et 3). b4 à 0 pour première face. Les autres bits sont à 0, sauf b7 et b2.

FFD- 2C 01 C0 BIT C001 teste b7 du n° de PISTE (à 1 si deuxième face)
000- 10 02 BPL D004 saute ligne suivante si première face visée
002- 09 10 ORA #10 force à 1 le b4 de A (valeur lue dans la table, qui devient donc: 1001 0100 pour A, 1011 0100 pour B, 1101 0100 pour C et 1111 0100 pour D, lorsque la deuxième face est visée)

004- 8D FB 04 STA 04FB code DRIVE et FACE
007- CC 0B C0 CPY C00B le DRIVE demandé est-il le drive actif?
00A- F0 0A BEQ D016 si oui, on continue en D016
00C- 8C 0B C0 STY C00B sinon, C00B est mis à jour pour activation
00F- 20 EA D0 JSR D0EA rend actif le drive indiqué en C00B
012- 90 02 BCC D016 si C = 0, continue en D016
014- 68 PLA sinon, récupère A et retourne
015- 60 RTS

Suite

016- AD 03 C0 LDA C003
019- AC 04 C0 LDY C004
01C- 85 F3 STA F3 F3/F4 mis à jour avec RWBUF
01E- 84 F4 STY F4
020- 78 SEI interdit les interruptions
021- A9 20 LDA #20 0010 0000 masque pour ET logique (positionne Z)
023- 2C 05 C0 BIT C005 teste b5, b6 et b7 du code de commande
026- 10 29 BPL D051 si b7=0, continue en D051 (b7 prime sur b6)
028- 50 02 BVC D02C si b6=0, saute l'instruction suivante (b6 prime sur b5)
02A- F0 25 BEQ D051 si b5=0, continue en D051

02C- AD 01 C0 LDA C001 sinon, compare PISTE demandée
02F- CD 0C C0 CMP C00C et C00C, piste active
032- F0 06 BEQ D03A continue en D03A si identiques
034- 48 PHA sinon, empile PISTE
035- 8A TXA
036- 09 04 ORA #04 force à 1 le b2 de X (code de commande)
038- AA TAX
039- 68 PLA récupère PISTE

03A- 29 7F AND #7F force à 0 le b7 de PISTE (à 1 si deuxième face)
03C- CD 11 03 CMP 0311 la PISTE demandée est-elle sous la tête?
03F- F0 10 BEQ D051 si oui, continue en D051
041- 8A TXA sinon, sauve le code de commande en A
042- A2 18 LDX #18 exécute la commande n° #18, c'est à dire
044- 20 F3 CF JSR CFF3 positionne la tête selon PISTE
047- 8D 05 C0 STA C005 remet le code de commande précédent dans C005
04A- AA TAX et dans X
04B- AD 13 03 LDA 0313 piste atteinte -> piste sous la tête
04E- 8D 11 03 STA 0311 (recopie le contenu de 0313 dans 0311)

PISTE est en place sous la tête

051- AD 01 C0 LDA C001 PISTE visée -> piste active
054- 8D 0C C0 STA C00C (mise à jour de C00C avec C001)
057- 29 7F AND #7F dont on force le b7 à 0 (flag 1ère/2ème face)
059- 8D 13 03 STA 0313 avant de l'écrire en 0313
05C- AD 02 C0 LDA C002 SECTEUR -> 0312
05F- 8D 12 03 STA 0312 (mise à jour de 0312 avec C002)
062- A0 00 LDY #00 Y = #00 préset pour délai
064- 8A TXA A reçoit le code de commande
065- 30 03 BMI D06A délai si X positif, sinon continue en D06A

Delai

067- 88 DEY
068- D0 FD BNE D067 pause: reboucle tant que Y ne revient pas à 0

06A- AD FB 04 LDA 04FB code DRIVE et FACE
06D- 09 01 ORA #01 masque 0000 0001 force b0 à 1
06F- 8D 14 03 STA 0314 et copie ce flag mis à jour en 0314
072- 8E 10 03 STX 0310 ainsi que X (code de commande) en 0310
075- 8A TXA teste les 4 bits forts de X:
076- 29 F0 AND #F0 1111 0000 force à 0 les b0 à b3

```
D078-  C9 E0      CMP #E0     le résultat est-il égal à 1110 0000?
D07A-  58         CLI         autorise les interruptions
D07B-  F0 04      BEQ D081    si oui, continue en D081
D07D-  29 20      AND #20     sinon, teste b5 du résultat (à 1 si écriture)
D07F-  D0 12      BNE D093    si écriture, continue en D093
```

Lecture sur disquette

```
D081-  AD 18 03   LDA 0318    si lecture, teste b7 de 0318 (drive prêt?)
D084-  30 FB      BMI D081    reboucle tant qu'il ne passe pas à 0
D086-  AD 13 03   LDA 0313    recopie le contenu de 0313 (octet/disquette)
D089-  91 F3      STA (F3),Y  à l'adresse indiquée en F3/F4 +Y (buffer)
D08B-  C8         INY         incrémente l'index et reboucle tant que Y ne
D08C-  D0 F3      BNE D081    retourne pas à 0 (c'est à dire après 256 octets)
D08E-  E6 F4      INC F4      incrémente HH de l'adresse d'écriture (page suivante)
D090-  4C 81 D0   JMP D081    et reboucle en D081 dans tous les cas. La sortie de ce s/p se fait sur ordre du contrôleur
(interruption), à une adresse spécifiée par ailleurs
```

Ecriture sur disquette

```
D093-  AD 18 03   LDA 0318    teste b7 de 0318 (drive prêt?)
D096-  30 FB      BMI D093    reboucle tant qu'il ne passe pas à 0
D098-  B1 F3      LDA (F3),Y  puis lit à l'adresse indiquée en F3/F4 +Y
D09A-  8D 13 03   STA 0313    et recopie en 0313 (c'est à dire sur disquette)
D09D-  C8         INY         incrémente l'index et reboucle tant que Y ne
D09E-  D0 F3      BNE D093    retourne pas à 0 (c'est à dire après 256 octets)
D0A0-  E6 F4      INC F4      incrémente HH de l'adresse de lecture (secteur suivant)
D0A2-  4C 93 D0   JMP D093    et reboucle en D093 dans tous les cas. La sortie de ce s/p se fait sur ordre du contrôleur
(interruption), à une adresse spécifiée par ailleurs
```

S/P vectorisé en #FFFE (IRQ)

```
D0A5-  2C 14 03   BIT 0314    teste b7 de 0314 (flag DRIVE et FACE)
D0A8-  10 03      BPL D0AD    saute l'instruction suivante si b7 à 0
D0AA-  4C F5 04   JMP 04F5    si b7 = 1, continue en 04F5 (IRQRAM)

D0AD-  68         PLA         si b7 = 0, dépile 3 octets inutiles
D0AE-  68         PLA
D0AF-  68         PLA
D0B0-  AD FB 04   LDA 04FB    recopie flag 04FB (lecture/écriture)
D0B3-  8D 14 03   STA 0314    en 0314 (flag DRIVE et FACE)
D0B6-  18         CLC         C = 0
D0B7-  AD 10 03   LDA 0310    lit le contenu de 0310 (code de commande)
D0BA-  29 5C      AND #5C     0101 1100 met à 0 les bits 0, 1, 5 et 7
D0BC-  A8         TAY         et sauve le résultat en Y
D0BD-  AE 05 C0   LDX C005    teste C005 (code de commande)
D0C0-  30 02      BMI D0C4    si b7 à 1, saute l'instruction suivante
D0C2-  A0 00      LDY #00     si b7 à 0, met à 0 tous les bits de Y

D0C4-  8C 17 C0   STY C017    et sauve Y en C017 ("bilan" de l'I/O)
D0C7-  29 40      AND #40     masque 0100 0000, teste si b6 est à 1, si oui,
D0C9-  D0 0F      BNE D0DA    disquette protégée en écriture, continue en D0DA
D0CB-  98         TYA         reprend le "bilan" précédent dans A
D0CC-  29 10      AND #10     et teste si b4 est à 0
D0CE-  F0 0D      BEQ D0DD    si oui, continue en D0DD
D0D0-  CE 06 C0   DEC C006    décrémente C006 (flag égal à 1 ou 2)
D0D3-  F0 05      BEQ D0DA    et branche en D0DA si devient nul
D0D5-  20 EA D0   JSR D0EA    rend actif le drive indiqué en C00B
D0D8-  90 0D      BCC D0E7    si C = 0, continue en D0E7

D0DA-  38         SEC         C = 1

D0DB-  68         PLA         récupère A et retourne
D0DC-  60         RTS

D0DD-  98         TYA         reprend le résultat précédent dans A
D0DE-  29 0C      AND #0C     0000 1100 met à 0 les bits sauf b2 et b3
D0E0-  F0 F9      BEQ D0DB    continue en D0DB si tous deux étaient nuls
D0E2-  CE 07 C0   DEC C007    sinon décrémente C007
D0E5-  F0 F3      BEQ D0DA    branche en D0DA si devient nul
```

```
0E7- 4C F7 CF   JMP CFF7      et reboucle en CFF7
     Rend actif le drive indiqué en C00B

0EA- 8A         TXA
0EB- 48         PHA
0EC- AD 03 C0   LDA C003      sauvegarde X et RWBUF
0EF- 48         PHA
0F0- AD 04 C0   LDA C004
0F3- 48         PHA
0F4- A9 60      LDA #60
0F6- A0 C0      LDY #C0       RWBUF mis à jour avec l'adresse C060
0F8- 8D 03 C0   STA C003
0FB- 8C 04 C0   STY C004
0FE- AD 06 C0   LDA C006      sauve dans A la variable locale valant 1 ou 2
101- A2 C0      LDX #C0       code de commande pour activation drive C00B
102- A0 01      LDY #01       nouvelle valeur pour C006
104- 20 EB CF   JSR CFEB      exécute commande X
108- 8D 06 C0   STA C006      régénère l'ancienne valeur de C006
10B- 68         PLA
10C- 8D 04 C0   STA C004
10F- 68         PLA           récupère RWBUF initial
110- 8D 03 C0   STA C003
113- B0 06      BCS D11B      si C = 1, continue en D11B
115- AD 12 03   LDA 0312      sinon, recopie le contenue de 0312
118- 8D 11 03   STA 0311      en 0311 (piste sous la tête)

11B- 68         PLA
11C- AA         TAX           récupère X et le copie en C005 (code commande)
11D- 8E 05 C0   STX C005
120- 60         RTS
```

S/P vectorisé en FFFA/FFFB (NMI)

```
121- AD FB 04   LDA 04FB      code DRIVE et FACE
124- 8D 14 03   STA 0314      recopié en 0314
127- AD 10 03   LDA 0310      teste si b0 de 0310 est à 0
12A- 4A         LSR           en le poussant dans C
12B- 90 05      BCC D132      si c'est le cas, continue en D132
12D- A9 D0      LDA #D0       si b0 de 0310 était à 1,
12F- 8D 10 03   STA 0310      remplace la valeur de 0310 par #D0

132- 38         SEC           met C à 1
133- 4C F8 04   JMP 04F8      et continue en 04F8 (NMIRAM)
```

S/P "BREAK ON BYTE"

En entrée X/F2 contiennent l'adresse de l'instruction suivant le "BREAK ON BYTE". En sortie, réinitialisation de la pile et retour au Ready après affichage du message et de l'adresse du "BREAK ON BYTE".

```
136- 86 F3      STX F3        sauve valeur de X dans F3 (LL de l'adresse suivante)
138- A2 04      LDX #04       indexe "BREAK ON BYTE"
13A- 20 6C D3   JSR D36C      affiche X+1ème message de zone CEE7 terminé par "caractère + 128"
13D- 38         SEC           prépare une soustraction
13E- A6 F3      LDX F3        récupère dans X le LL de l'adresse suivante
140- A5 F2      LDA F2        récupère dans A le HH de l'adresse suivante
142- E9 02      SBC #02       calcule l'adresse du BREAK (A = LL - #02)
144- B0 01      BCS D147      saute l'instruction suivante si pas de retenue
146- CA         DEX           sinon décrémente aussi HH de l'adresse suivante
147- 48         PHA           sauve LL, le résultat de la soustraction
148- 8A         TXA           HH passe dans A pour être affiché en premier
149- 20 13 D6   JSR D613      XAFHEX affiche en hexadécimal le contenu de A
14C- 68         PLA           récupère LL dans A pour l'afficher en second
14D- 20 13 D6   JSR D613      XAFHEX affiche en hexadécimal le contenu de A
150- 58         CLI           autorise les interruptions
151- A2 FF      LDX #FF       réinitialise la pile (transfère #FF dans S)
153- 9A         TXS           et continue en D154 (retourne au Ready)
```

SÉRIE D'APPELS À DES SOUS-PROGRAMMES EN ROM

Dans chacun de ces sous-programmes, on a un appel au s/p D5D8 en RAMOV, qui lira l'adresse de la routine à appeler en ROM. Cette adresse se trouve juste après le JSR D5D8 selon la version de ROM utilisée: les deux premiers octets constituent l'adresse LLHH de la version Oric-1, les deux suivants celle de la version Atmos. Le s/p D5D8 ajustera son RTS pour revenir sur l'octet suivant l'adresse de la routine Atmos.

Retourne au Ready après affichage d'un message d'erreur

D154-	20 D8 D5	JSR D5D8	XROM exécute à partir de la RAM une routine ROM
D157-	AD C4 A0 C4		adresse ROM 1.0 adresse ROM 1.1
D15B-	60	RTS	

Décale un bloc memoire vers le haut

En CE/CF adresse du 1er octet du bas, en C9/CA adresse dernier octet du haut, en C7/C8 et AY adresse cible vers 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 en A0/A1 (haut des tableaux)

D15C-	20 D8 D5	JSR D5D8	XROM exécute à partir de la RAM une routine ROM
D15F-	F8 C3 F4 C3		adresse ROM 1.0 adresse ROM 1.1
D163-	60	RTS	

Vérifie que l'adresse AY est en dessous des chaînes
"OUT OF MEMORY ERROR" si AY trop haut, zone C7/CF n'est pas affectée, AY conservé

D164-	20 D8 D5	JSR D5D8	XROM exécute à partir de la RAM une routine ROM
D167-	48 C4 44 C4		adresse ROM 1.0 adresse ROM 1.1
D16B-	60	RTS	

Affiche "OUT OF MEMORY"
Puis réinitialise la pile, affiche " ERROR" et retourne au "Ready"

D16C-	A2 4D	LDX #4D	message "OUT OF MEMORY"
D16E-	2C A9 A3	BIT A3A9	continue en D171

Affiche le message "DISP TYPE MISMATCH"
Puis réinitialise la pile, affiche " ERROR" et retourne au "Ready"

D16F-	A9 A3	LDA #A3	message "DISP TYPE MISMATCH" (bogue: LDX)
D171-	20 D8 D5	JSR D5D8	XROM exécute à partir de la RAM une routine ROM
D174-	85 C4 7E C4		adresse ROM 1.0 adresse ROM 1.1 (affiche le message)

Réinitialise la pile, affiche " ERROR" et retourne au "Ready"

D178-	20 D8 D5	JSR D5D8	XROM exécute à partir de la RAM une routine ROM
D17B-	A3 C4 96 C4		adresse ROM 1.0 adresse ROM 1.1
D17F-	60	RTS	

Retourne au Ready

D180-	20 D8 D5	JSR D5D8	XROM exécute à partir de la RAM une routine ROM
D183-	B5 C4 A8 C4		adresse ROM 1.0 adresse ROM 1.1
D187-	60	RTS	

Restaure les liens des lignes à partir du début

D188-	A5 9A	LDA 9A	Prendre début du Basic
D18A-	A4 9B	LDY 9B	comme pointeur de travail

Restaure les liens des lignes à partir de l'adresse AY

D18C-	20 D8 D5	JSR D5D8	XROM exécute à partir de la RAM une routine ROM
D18F-	73 C5 63 C5		adresse ROM 1.0 adresse ROM 1.1
D193-	60	RTS	

Encode les mots-clés

D194-	20 D8 D5	JSR D5D8	XROM exécute à partir de la RAM une routine ROM
D197-	0A C6 FA C5	adresse ROM 1.0 adresse ROM 1.1	
D19B-	60	RTS	

Recherche une ligne BASIC de n° 33/34 à partir du début

Si trouve, retourne avec C = 1 et adresse en CE/CF (visant le 1er octet de lien)

19C- 20 D8 D5 JSR D5D8 XROM exécute à partir de la RAM une routine ROM
19F- DE C6 B3 C6 adresse ROM 1.0 adresse ROM 1.1
1A3- 60 RTS

Recherche une ligne BASIC à partir de la ligne courante

Si trouve, retourne avec C = 1 et adresse en CE/CF (visant le 1er octet de lien)

1A4- 20 D8 D5 JSR D5D8 XROM exécute à partir de la RAM une routine ROM
1A7- EE C6 C3 C6 adresse ROM 1.0 adresse ROM 1.1
1AB- 60 RTS

Place TXTPTR au début du programme BASIC

1AC- 20 D8 D5 JSR D5D8 XROM exécute à partir de la RAM une routine ROM
1AF- 65 C7 3A C7 adresse ROM 1.0 adresse ROM 1.1
1B3- 60 RTS

Exécute la commande "LIST" simplifiée

1B4- 20 D8 D5 JSR D5D8 XROM exécute à partir de la RAM une routine ROM
1B7- 99 C7 6C C7 adresse ROM 1.0 adresse ROM 1.1
1BB- 60 RTS

ROM 1.0: Simple RTS, ROM 1.1: Met l'imprimante en service

1BC- 20 D8 D5 JSR D5D8 XROM exécute à partir de la RAM une routine ROM
1BF- 40 C8 16 C8 adresse ROM 1.0 adresse ROM 1.1
1C3- 60 RTS

Met l'imprimante hors service

1C4- 20 D8 D5 JSR D5D8 XROM exécute à partir de la RAM une routine ROM
1C7- 3D C8 2F C8 adresse ROM 1.0 adresse ROM 1.1
1CB- 60 RTS

Exécute la commande "Restore" du BASIC

1CC- 20 D8 D5 JSR D5D8 XROM exécute à partir de la RAM une routine ROM
1CF- 1F C9 52 C9 adresse ROM 1.0 adresse ROM 1.1
1D3- 60 RTS

"UNDEF'D STATEMENT ERROR" (GOSUB)

1D4- 20 D8 D5 JSR D5D8 XROM exécute à partir de la RAM une routine ROM
1D7- F1 C9 23 CA adresse ROM 1.0 adresse ROM 1.1
1DB- 60 RTS

Calcule le déplacement à l'instruction suivante, met à jour TXTPTR en ajoutant Y

1DC- 20 D8 D5 JSR D5D8 XROM exécute à partir de la RAM une routine ROM
1DF- 1C CA 4E CA adresse ROM 1.0 adresse ROM 1.1
1E3- 20 D8 D5 JSR D5D8 XROM exécute à partir de la RAM une routine ROM
1E6- 0D CA 3F CA adresse ROM 1.0 adresse ROM 1.1
1EA- 60 RTS

Exécute la commande "IF"

1EB- 20 D8 D5 JSR D5D8 XROM exécute à partir de la RAM une routine ROM
1EE- 41 CA 73 CA adresse ROM 1.0 adresse ROM 1.1
1F2- 60 RTS

XCRGOT + évalue le numéro de ligne à TXTPTR (résultat en 33/34)

1F3- 20 9E D3 JSR D39E XCRGOT relit le caractère à TXTPTR et le convertit en MAJUSCULE
1F6- 20 D8 D5 JSR D5D8 XROM exécute à partir de la RAM une routine ROM
1F9- 98 CA E2 CA adresse ROM 1.0 adresse ROM 1.1
1FD- 60 RTS

Affecte un nombre à une variable

1FE- 20 D8 D5 JSR D5D8 XROM exécute à partir de la RAM une routine ROM
201- EF CA 39 CB adresse ROM 1.0 adresse ROM 1.1
205- 60 RTS

Va à la ligne

206- 20 D8 D5 JSR D5D8 XROM exécute à partir de la RAM une routine ROM
209- 9F CB F0 CB adresse ROM 1.0 adresse ROM 1.1
20D- 60 RTS

Affiche le caractère présent dans A

D20E- 20 D8 D5 JSR D5D8 XROM exécute à partir de la RAM une routine ROM
D211- 12 CC D9 CC adresse ROM 1.0 adresse ROM 1.1
D215- 60 RTS

Evalue une expression numérique à TXTPTR
Retourne avec la valeur numérique dans ACC1

D216- 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

Vérifie si l'expression évaluée à TXTPTR est bien numérique

D219- 18 CLC une variable numérique est demandée
D21A- 24 38 BIT 38 et saute le SEC suivant (continue en D21C)

Vérifie si l'expression évaluée à TXTPTR est bien alphanumérique

D21B- 38 SEC une variable alphanumérique est demandée

Vérifie si expression évaluée à TXTPTR est bien conforme
(numérique si C = 0, alphanumérique si C = 1)
Retourne avec la valeur numérique dans ACC1 ou l'adresse de chaîne dans D3/D4

D21C- 20 D8 D5 JSR D5D8 XROM exécute à partir de la RAM une routine ROM
D21F- 7D CE 09 CF adresse ROM 1.0 adresse ROM 1.1 vérifie si variable OK
D223- 60 RTS

Evalue une expression numérique ou alphanumérique 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

D224- 20 D8 D5 JSR D5D8 XROM exécute à partir de la RAM une routine ROM
D227- 8B CE 17 CF adresse ROM 1.0 adresse ROM 1.1
D22B- 60 RTS

Demande une virgule à TXTPTR et lit le caractère suivant
avec conversion éventuelle de minuscule en MAJUSCULE
Cette lecture ne sert souvent qu'a placer TXTPTR sur le caractère qui suit la virgule

D22C- A9 2C LDA #2C code de "," suite en D22E

Demande à TXTPTR un octet identique à A et lit le caractère suivant
avec conversion éventuelle de minuscule en MAJUSCULE
Cette routine ne sert souvent qu'a placer TXTPTR sur le caractère qui suit l'octet exigé. Elle est à l'origine de pratiquement tous les problèmes d'incompatibilité des minuscules dans la syntaxe des commandes Sédoric (bogue). En effet l'octet exigé correspond souvent à un token BASIC ou à une lettre majuscule.

D22E- 20 D8 D5 JSR D5D8 XROM exécute à partir de la RAM une routine ROM
D231- DB CF 67 D0 adresse ROM 1.0 adresse ROM 1.1
D235- 4C A1 D3 JMP D3A1 convertit en MAJUSCULE et RTS

Place dans AY et D3/D4 "l'adresse" de la variable à TXTPTR
Décode le nom de la première variable à TXTPTR. 2B contient un code permettant d'exclure certains types (remis à zéro par CLEAR). Si #00 tous les types sont autorisés, si #40 demande le nom d'un tableau (pour STORE ou RECALL par exemple), si #80 interdit les tableaux et les entiers (pour FOR NEXT par exemple). 27 sert de flag "consultation/déclaration" pour les tableaux. Au retour AY, B6/B7 et D3/D4 pointent sur les data (longueur/adresse dans le cas d'une chaîne) de cette variable dans la zone des variables BASIC

D238- 20 D8 D5 JSR D5D8 XROM exécute à partir de la RAM une routine ROM
D23B- FC D0 88 D1 adresse ROM 1.0 adresse ROM 1.1
D23E- 85 D3 STA D3 et passe le résultat dans D3/D4
D241- 84 D4 STY D4
D243- 60 RTS

Cherche l'adresse de la valeur d'une variable
dont les 2 caractères significatifs sont indiqués 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) (rappel la valeur d'une chaîne est remplacée par sa longueur et son adresse).

D244- 20 D8 D5 JSR D5D8 XROM exécute à partir de la RAM une routine ROM
D247- 58 D1 E8 D1 adresse ROM 1.0 adresse ROM 1.1
D24A- 60 RTS

Transfère le nombre de ACC1 en #D4-#D3 (non signé)

24C- 20 D8 D5 JSR D5D8 XROM exécute à partir de la RAM une routine ROM
24F- 17 D2 A9 D2 adresse ROM 1.0 adresse ROM 1.1
253- 60 RTS

Transfère le nombre de AY dans ACC1 (signé)

254- 20 D8 D5 JSR D5D8 XROM exécute à partir de la RAM une routine ROM
257- ED D3 99 D4 adresse ROM 1.0 adresse ROM 1.1
25B- 60 RTS

Interdit le mode direct

25C- 20 D8 D5 JSR D5D8 XROM exécute à partir de la RAM une routine ROM
25F- 19 D4 D2 D4 adresse ROM 1.0 adresse ROM 1.1
263- 60 RTS

Réserve une place en mémoire pour une chaîne de longueur A
Sauvegarde la longueur en D0 et l'adresse en D1/D2

264- 20 D8 D5 JSR D5D8 XROM exécute à partir de la RAM une routine ROM
267- F0 D4 AB D5 adresse ROM 1.0 adresse ROM 1.1
26A- 60 RTS

"STRING TOO LONG ERROR"

26C- 20 D8 D5 JSR D5D8 XROM exécute à partir de la RAM une routine ROM
26F- C7 D6 82 D7 adresse ROM 1.0 adresse ROM 1.1
273- 60 RTS

Vérifie s'il y a une chaîne à TXTPTR
Retourne longueur dans A et adresse dans XY et dans 91/92

274- 20 1B D2 JSR D21B SEC et JSR CF09/ROM (vérifie si expression évaluée à TXTPTR est bien alphanumérique, retourne adresse chaîne dans D3/D4)

277- 20 D8 D5 JSR D5D8 XROM exécute à partir de la RAM une routine ROM
27A- 15 D7 D0 D7 adresse ROM 1.0 adresse ROM 1.1 (longueur -> A avec N et
27E- 60 RTS Z selon cette longueur, adresse -> XY et 91/92)

Evalue un nombre entier à TXTPTR et le retourne dans X

27F- 20 16 D2 JSR D216 JSR CF17/ROM et CF09/ROM (évalue une expression numérique à TXTPTR, retourne avec cette valeur numérique dans ACC1)

Prend un entier dans ACC1 et le retourne dans X

282- 20 D8 D5 JSR D5D8 XROM exécute à partir de la RAM une routine ROM
285- 10 D8 CB D8 adresse ROM 1.0 adresse ROM 1.1 (prend entier dans X)
289- 60 RTS

Convertit le nombre présent dans ACC1 en entier signé dans YA, D3/D4 et 33/34
En sortie Z et N sont positionnés selon LL, c'est à dire Y

28A- 20 D8 D5 JSR D5D8 XROM exécute à partir de la RAM une routine ROM
28D- 6B D8 26 D9 adresse ROM 1.0 adresse ROM 1.1
291- 60 RTS

Prend 2 coordonnées xy à TXTPTR et les retourne dans #2F8(x) et X(y)

292- 20 D8 D5 JSR D5D8 XROM exécute à partir de la RAM une routine ROM
295- 96 D9 22 DA adresse ROM 1.0 adresse ROM 1.1
299- 60 RTS

Effectue AY - ACC1 -> ACC1 (soustraction)

29A- 20 D8 D5 JSR D5D8 XROM exécute à partir de la RAM une routine ROM
29D- 80 DA 0B DB adresse ROM 1.0 adresse ROM 1.1
2A1- 60 RTS

Additionne le contenu de ACC1 (floating point accumulator)
et la valeur pointée par AY et replace le résultat dans ACC1

2A2- 20 D8 D5 JSR D5D8 XROM exécute à partir de la RAM une routine ROM
2A5- 97 DA 22 DB adresse ROM 1.0 adresse ROM 1.1
2A9- 60 RTS

Multiplie le contenu de ACC1 (floating point accumulator)
par la valeur pointée par AY et replace le résultat dans ACC1

D2AA- 20 D8 D5 JSR D5D8 XROM exécute à partir de la RAM une routine ROM
D2AD- B7 DC ED DC adresse ROM 1.0 adresse ROM 1.1
D2B1- 60 RTS

Effectue AY / ACC1 -> ACC1 (division)
D2B2- 20 D8 D5 JSR D5D8 XROM exécute à partir de la RAM une routine ROM
D2B5- E0 DD E4 DD adresse ROM 1.0 adresse ROM 1.1
D2B9- 60 RTS

Transfère dans ACC1 (floating point accumulator) la valeur pointée par AY
D2BA- 20 D8 D5 JSR D5D8 XROM exécute à partir de la RAM une routine ROM
D2BD- 73 DE 7B DE adresse ROM 1.0 adresse ROM 1.1
D2C1- 60 RTS

Recopie les 5 octets de ACC1 de l'adresse XY à l'adresse XY + 4
D2C2- 20 D8 D5 JSR D5D8 XROM exécute à partir de la RAM une routine ROM
D2C5- A5 DE AD DE adresse ROM 1.0 adresse ROM 1.1
D2C9- 60 RTS

Transfère un nombre non signé YA dans ACC1
D2CA- 20 D8 D5 JSR D5D8 XROM exécute à partir de la RAM une routine ROM
D2CD- D5 D8 40 DF adresse ROM 1.0 adresse ROM 1.1
D2D1- 60 RTS

Convertit ACC1 en chaîne décimale d'adresse AY
La chaîne AY décimale est située à partir de #0100 (qui contient le signe "-" ou un espace) et est terminée par #00

D2D2- 20 D8 D5 JSR D5D8 XROM exécute à partir de la RAM une routine ROM
D2D5- D1 E0 D5 E0 adresse ROM 1.0 adresse ROM 1.1
D2D9- 60 RTS

Effectue un changement de signe de ACC1
D2DA- 20 D8 D5 JSR D5D8 XROM exécute à partir de la RAM une routine ROM
D2DD- 6D E2 71 E2 adresse ROM 1.0 adresse ROM 1.1
D2E1- 60 RTS

Génère un nombre entre 0 et 1 (en FA)
D2E2- 20 D8 D5 JSR D5D8 XROM exécute à partir de la RAM une routine ROM
D2E5- 79 E3 7D E3 adresse ROM 1.0 adresse ROM 1.1
D2E9- 60 RTS

Effectue la fonction ACC1 = COS(ACC1)
D2EA- 20 D8 D5 JSR D5D8 XROM exécute à partir de la RAM une routine ROM
D2ED- 87 E3 8B E3 adresse ROM 1.0 adresse ROM 1.1
D2F1- 60 RTS

Effectue la fonction ACC1 = SIN(ACC1)
D2F2- 20 D8 D5 JSR D5D8 XROM exécute à partir de la RAM une routine ROM
D2F5- 8E E3 92 E3 adresse ROM 1.0 adresse ROM 1.1
D2F9- 60 RTS

Evalue un nombre non signé à TXTPTR (sur 2 octets)

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)

D2FA- 20 D8 D5 JSR D5D8 XROM exécute à partir de la RAM une routine ROM
D2FD- 9D E7 53 E8 adresse ROM 1.0 adresse ROM 1.1
D301- 60 RTS

Saisit une touche: si touche frappée alors N = 1 et A = code ASCII sinon N = 0
D302- 20 D8 D5 JSR D5D8 XROM exécute à partir de la RAM une routine ROM
D305- 05 E9 78 EB adresse ROM 1.0 adresse ROM 1.1
D309- 60 RTS

Autorise IRQ (gestion clavier et curseur)

```
30A- 20 D8 D5    JSR D5D8      XROM exécute à partir de la RAM une routine ROM
30D- C7 EC E0 ED adresse ROM 1.0 adresse ROM 1.1
311- 60          RTS
```

Effectue la commande "DRAW"

```
312- 20 D8 D5    JSR D5D8      XROM exécute à partir de la RAM une routine ROM
315- 79 F0 10 F1               adresse ROM 1.0 adresse ROM 1.1
319- 60          RTS
```

Trouve le code ASCII de la touche pressée

En entrée, 0208 contient le code de la touche, 0209 le code de la touche SHIFT ou CTRL et 020C le masque minuscule/MAJUSCULE. En sortie A contient le code ASCII avec b7 à 1. Si le b7 de A est à 0, pas de touche pressée.

```
31A- 20 D8 D5    JSR D5D8      XROM exécute à partir de la RAM une routine ROM
31D- 94 F4 EF F4               adresse ROM 1.0 adresse ROM 1.1
321- 60          RTS
```

Appelle la routine d' E/S du PSG 8912

Met X dans le registre A du PSG 8912 (Programmable Sound Generator).
Il y a 14 registres: n°1 à 13 pour le son et n°14 pour le clavier.

```
322- 20 D8 D5    JSR D5D8      XROM exécute à partir de la RAM une routine ROM
325- 35 F5 90 F5               adresse ROM 1.0 adresse ROM 1.1
329- 60          RTS
```

Eteint/allume le curseur

Si le curseur était visible (b0 de 026A à 1) et si A = #01 le curseur sera mis en vidéo inverse sinon le caractère sous le curseur sera en video normale

```
32A- 20 D8 D5    JSR D5D8      XROM exécute à partir de la RAM une routine ROM
32D- CB F7 01 F8               adresse ROM 1.0 adresse ROM 1.1
331- 60          RTS
```

Régénère le jeu de caractères normaux (descend de la ROM dans la RAM)

```
332- 20 D8 D5    JSR D5D8      XROM exécute à partir de la RAM une routine ROM
335- 3E F9 82 F9               adresse ROM 1.0 adresse ROM 1.1
339- 60          RTS
```

Incrémente TXTPTR et lit un caractère (CHRGET)

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.

```
33A- 20 D8 D5    JSR D5D8      XROM exécute à partir de la RAM une routine ROM
33D- E2 00 E2 00               adresse ROM 1.0 adresse ROM 1.1
341- 60          RTS
```

Lit le caractère à TXTPTR (CHRGOT)

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.

```
342- 20 D8 D5    JSR D5D8      XROM exécute à partir de la RAM une routine ROM
345- E8 00 E8 00               adresse ROM 1.0 adresse ROM 1.1
349- 60          RTS
```

Copie NOM et EXT de la table CCF7 dans BUFNOM

```
34A- A0 09       LDY #09       Y = #09 copiera 3 octets
34C- 2C A0 00    BIT 00A0      continue en D34F
34D- A0 00       LDY #00       Y = #00 copiera 12 octets
34F- BD F7 CC    LDA CCF7,X    lecture dans table CCF7 selon X à l'entrée
352- 99 29 C0    STA C029,Y    écriture dans BUFNOM selon Y (ci-dessus)
355- C8          INY           Y suivant (de 0 à 12 ou de 9 à 12)
356- E8          INX           X suivant
357- C0 0C       CPY #0C
359- D0 F4       BNE D34F      et reboucle tant que Y est < 12
35B- 60          RTS           retourne avec Z = 1 (égal)
```

NB: BUFNOM (de C028 à C034) comporte 13 cases: dnnnnnnnnneee où d est le n° du drive, n est le nom en 9 caractères et e est l'extension en 3 caractères. Cette routine permet de lire nnnnnnnnneee ou eee dans la table CCF7 et de l'ecrire à la bonne place dans BUFNOM.

Affichage des messages

Affiche X+1ème message d'erreur externe terminé par un "caractère + 128"

D35C-	AD 0D C0	LDA C00D	initialise AY avec adresse -1 messages
D35F-	AC 0E C0	LDY C00E	d'erreur externe contenue dans EXTER
D362-	D0 12	BNE D376	et continue en D376

XAFSC affiche le X+1ème message externe terminé par un "caractère + 128"

D364-	AD 0F C0	LDA C00F	initialise AY avec adresse - 1 des messages externes
D367-	AC 10 C0	LDY C010	contenue dans EXTMS (adresse - 1 du 1er message: "LFCRTRACK:")
D36A-	D0 0A	BNE D376	et continue en D376

Affiche le X+1ème message situé dans la zone CEE7 et terminé par un "caractère + 128"

D36C-	A9 E6	LDA #E6	initialise AY avec CEE6
D36E-	A0 CE	LDY #CE	(adresse -1 du 1er message)
D370-	D0 04	BNE D376	et continue en D376

Affiche le X+1ème message situé dans la zone CDBF et terminé par un "caractère + 128"

D372-	A9 BE	LDA #BE	initialise AY avec CDBE
D374-	A0 CD	LDY #CD	(adresse - 1 du 1er message) et continue en D376

Entrée réelle affichage (X+1)ème message de zone AY+1 terminé par un caractère + 128

D376-	85 18	STA 18	dépose en 18/19 l'adresse - 1
D378-	84 19	STY 19	du début des messages (adresse - 1 du 1er)
D37A-	A0 00	LDY #00	index de lecture
D37C-	CA	DEX	n° d'ordre du message à afficher
D37D-	30 0C	BMI D38B	si trouvé branche en D38B pour affichage
D37F-	E6 18	INC 18	mise à jour adresse du caractère à lire
D381-	D0 02	BNE D385	(avec repport de retenue si besoin)
D383-	E6 19	INC 19	
D385-	B1 18	LDA (18),Y	lecture caractère
D387-	10 F6	BPL D37F	si positif reboucle pour lire caractère suivant
D389-	30 F1	BMI D37C	si c'est dernier caractère reboucle message suivant
D38B-	C8	INY	mise à jour index de lecture
D38C-	B1 18	LDA (18),Y	lecture caractère
D38E-	08	PHP	sauvegarde du drapeau N
D38F-	29 7F	AND #7F	mise à zéro éventuelle du bit de signe
D391-	20 2A D6	JSR D62A	XAFCAR affiche le caractère ASCII contenu dans A
D394-	28	PLP	récupère le drapeau N
D395-	10 F4	BPL D38B	si pas dernier caractère reboucle caractère suivant
D397-	60	RTS	sinon c'est fini

XCRGET saisit dans A le caractère suivant et le convertit en MAJUSCULE

D398-	20 3A D3	JSR D33A	JSR 00E2/ROM incrémente TXTPTR, lit un caractère (CHRGET), les espaces sont sautés, 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
D39B-	4C A1 D3	JMP D3A1	conversion minuscule -> MAJUSCULE

XCRGOT saisit dans A le caractère courant et le convertit en MAJUSCULE

D39E-	20 42 D3	JSR D342	JSR 00E8/ROM (CHRGOT) lit le caractère à TXTPTR 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

minMAJ convertit (si nécessaire) en MAJUSCULE le caractère dans A

D3A1-	08	PHP	sauve P, c à d les indic init par 00E2 ou 00E8
D3A2-	C9 61	CMP #61	compare à "a"
D3A4-	90 06	BCC D3AC	inutile de continuer si lettre < "a"
D3A6-	C9 7B	CMP #7B	compare à "é" (lettre qui suit "z")
D3A8-	B0 02	BCS D3AC	inutile de continuer si lettre >= "é" (les lettres accentuées ne sont pas converties)
D3AA-	29 DF	AND #DF	si "a" <= lettre <= "z" effectue un ET logique avec masque 1101 1111, c'est à dire conserve tous les bits sauf le b5 qui est remis à zéro.

Exemple: "b" = #62 = 0110 0010
ET 1101 1111
= 0100 0010 = #42 = "B"

D3AC-	28	PLP	récupère P
D3AD-	60	RTS	et retourne

ENTRÉE SÉDORIC
RECHERCHE L'ADRESSE D'EXÉCUTION D'UN MOT-CLÉ SÉDORIC

Résumé:
- Lecture du caractère présent à TXTPTR (sans incrémentation de TXTPTR)
- Conversion éventuelle en majuscule si la lettre est comprise entre "a" et "z"
- Conversion en valeur de #00 (A) à #19 (Z) ou #1A (autre caractère)
- Calcul de l'index des coordonnées dans la sous-table selon la 1ère lettre mot-clé
- Lecture de l'adresse 1er mot-clé de même initiale dans la table principale (18/19)
- Lecture du nombre de mots-clés ayant la même initiale (mis en F2),
- Lecture du numéro d'ordre du mot-clé Sédoric (mis en X)
- Examen correspondance avec l'un des mots-clés ayant la même initiale si trouve: exécution en D417
  sinon: continue en D428

```
3AE- A2 00      LDX #00
3B0- 8E FD 04   STX 04FD      RAZ de ERROR (numéro de l'erreur Sédoric)
3B3- BA         TSX           sauve pointeur de pile en SAUVES
3B4- 8E 23 C0   STX C023      pour sortie éventuelle par erreur
3B7- A5 E9      LDA E9
3B9- A4 EA      LDY EA
3BB- 8D 1F C0   STA C01F      sauvegarde de TXTPTR en C01F/C020 (SVTPTR)
3BE- 8C 20 C0   STY C020
3C1- 20 9E D3   JSR D39E      XCRGOT relit le caractère à TXTPTR et le convertit en MAJUSCULE
3C4- E9 41      SBC #41       teste si #40<A<#5C (A contient-il une lettre)
3C6- A0 1A      LDY #1A       #1A = valeur finale si le caractère dans A n'est pas une lettre
3C8- 90 08      BCC D3D2      A<#41 ce n'est pas une lettre, donc A = #1A par défaut
     NB: BCC D3D2 teste C résultant de SBC#41 (C n'est pas affecté par le LDY #1A)
3CA- C9 1A      CMP #1A       on compare en réalité à #41 + #1A = #5B
3CC- B0 04      BCS D3D2      A>=#5B pas une lettre, donc A = #1A par défaut
3CE- A8         TAY           #00 <= Y <= #19 pour un caractère de A à Z
3CF- 20 3A D3   JSR D33A      JSR 00E2/ROM incrémente TXTPTR, lit un caractère (CHRGET), les espaces sont sautés,
     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
3D2- 98         TYA           A de #00 (A) à #19 (Z) ou #1A (autre caractère)
3D3- 0A         ASL           calcule index Y en multipliant A par 4
3D4- 0A         ASL           car on va indexer avec Y une sous-table en CBBB
3D5- A8         TAY           constituée de groupes de 4 octets
3D6- B9 BB CB   LDA CBBB,Y    lit 2 premiers octets (adresse dans table principale
3D9- 85 18      STA 18        (C9DE-CBBA) où commencent mots-clés de même
3DB- B9 BC CB   LDA CBBC,Y    initiale) et place ces 2 octets en 18/19
3DE- 85 19      STA 19        (travail encodage/décodage mots clés)
3E0- B9 BE CB   LDA CBBE,Y    lit le nombre de mots-clés de même initiale
3E3- 85 F2      STA F2        et le place en F2 (TRAV0)
3E5- BE BD CB   LDX CBBD,Y    lit n° d'ordre mot-clé Sédoric, le garde en X
3E8- C6 F2      DEC F2        décrémente le nombre de mots-clés de même initiale
3EA- 30 3C      BMI D428      si négatif: branche en D428 ("non trouvé")
3EC- A0 FF      LDY #FF       pour chaque mot-clé examiné consulte la table
3EE- C8         INY           principale à l'adresse où commence la liste des
3EF- B1 18      LDA (18),Y    mots-clés de même initiale (privés de la 1ère
3F1- F0 24      BEQ D417      lettre, déjà connue), lit le mot-clé courant si #00 (= marqueur de fin de mot) est atteint sans
     trouver de différence, le bon mot-clé est trouvé, branche en D417 pour "exécution"
3F3- 85 F3      STA F3        chaque lettre, placée en F3, sera comparée à
3F5- B1 E9      LDA (E9),Y    la lettre corresp à TXTPTR et chargée en A
3F7- C9 61      CMP #61       mise en MAJUSCULE: compare à "a"
3F9- 90 06      BCC D401      inutile de continuer si lettre < "a"
3FB- C9 7B      CMP #7B       compare à "é" (lettre qui suit "z")
3FD- B0 02      BCS D401      inutile de continuer si lettre >= "é"
3FF- 29 DF      AND #DF       si "a" <= lettre <= "z" ET logique avec masque 1101 1111, c'est à dire remise à zéro du b5
     (conversion minuscule -> MAJUSCULE)
401- C5 F3      CMP F3        le caractère pointé par TXTPTR est-il égal à celui de la table?
403- F0 E9      BEQ D3EE      si oui, on reboucle pour examiner le suivant
405- C8         INY           sinon, cherche la fin de ce mauvais mot-clé
406- B1 18      LDA (18),Y    dans la table (fin marquée par #00), ce qui évite de vérifier le mot-clé dans son entier
408- D0 FB      BNE D405      lorsque fin mot-clé trouvée,
40A- E8         INX           augmente X (n° d'ordre mot-clé Sédoric)
40B- 38         SEC           prépare addition de Y à 18/19 pour
40C- 98         TYA           mise à jour de l'adresse du mot-clé suivant,
40D- 65 18      ADC 18        Y = A = nombre de lettres du mot-clé
40F- 85 18      STA 18        testé infructueusement (Super!)
411- 90 D5      BCC D3E8      LL en 18 et report de
413- E6 19      INC 19        l'éventuelle retenue sur HH en 19
415- B0 D1      BCS D3E8      reboucle pour explorer ce nouveau mot-clé
```

S/P "exécution"

D417- 8A TXA multiplie n° d'ordre mot-clé Sédoric par 2
D418- 0A ASL pour obtenir l'index d'une table d'adresses
D419- AA TAX (deux octets par adresse)
D41A- BD 28 CC LDA CC28,X lit et empile l'adresse d'exécution de
D41D- 48 PHA la commande Sédoric (en fait adresse - 1
D41E- BD 27 CC LDA CC27,X car sera appelée par un RTS, qui tout bêtement
D421- 48 PHA incrémente adresse de retour avant de l'utiliser)
D422- 20 E3 D1 JSR D1E3 CA3F/ROM met à jour TXTPTR en ajoutant Y, TXTPTR pointe maintenant sur le caractère qui suit le mot-clé Sédoric (#00 ou ":" ou paramètres de la commande)
D425- 4C 9E D3 JMP D39E XCRGOT relit le caractère à TXTPTR et le convertit en MAJUSCULE et surtout **RTS** à l'adresse empilée juste avant

S/P "non trouvé"

D428- AD 1F C0 LDA C01F entrée s/p "non trouvé"
D42B- AC 20 C0 LDY C020 restaure la valeur originale de
D42E- 85 E9 STA E9 TXTPTR en vue d'une autre stratégie:
D430- 84 EA STY EA recherche d'un nom de fichier
D432- 20 9E D3 JSR D39E XCRGOT relit le caractère à TXTPTR et le convertit en MAJUSCULE
D435- F0 12 BEQ D449 RTS si A = #00 (ce s/p sert probablement à d'autres fins)
Ce RTS retourne au s/p appelant (0400) et continue en ECB9 sur ROM
D437- A0 FF LDY #FF initialise pour que Y = 0 en début de boucle
D439- C8 INY la boucle ci-contre relit le buffer d'entrée
D43A- B1 E9 LDA (E9),Y jusqu'à ":" ou #00 ou ">"
D43C- F0 0C BEQ D44A continue en D44A lorsqu'il trouve #00
D43E- C9 3A CMP #3A est-ce ":" ?
D440- F0 08 BEQ D44A continue en D44A lorsqu'il trouve ":"
D442- C9 D3 CMP #D3 si pas ">" reboucle
D444- D0 F3 BNE D439 Sauf présence de ">" finit par sortir en D44A
D446- 4C BA F5 JMP F5BA si trouve ">" continue en F5BA
D449- 60 RTS RTS utilisé en D435 (BEQ D449)

S/P "sortie vers LOAD"

D44A- A9 00 LDA #00 initialise A = #00 pour commande LOAD
D44C- 4C F9 DF JMP DFF9 (caractérise passage par s/p "non trouvé")

Exemple: décodage d'une commande sans paramètre: OLD

D3C1- La lettre "O" ou "o" est saisie (conversion éventuelle en majuscule) et vaut #4F
D3C4- De #4F, retire #41, il reste Y = #0E qui donne l'index Y = 4 x Y = #38
D3D6- 4 octets de la sous-table sont lus à partir de CBBB + 38 (soit CBF3). 18/19 reçoit l'adresse de début des mots-clés commençant par "O" dans la table principale soit CAF1 X reçoit #3A (n° d'ordre du 1er mot-clé en "O" c'est à dire OUT). F2 reçoit #03 (nombre de mots-clés commençant par "O")
D3E8- F2 est décrémenté (F2 = 2, il y a encore des mots-clés à examiner)
D3EC- Y = #FF pour passer à #00 à ligne suivante (qui fait partie d'une boucle)
D3EE- Y sert à saisir les lettres à partir de l'adr écrite en 18/19 (CAF1). C'est le "U" de OUT qui vaut #55 (donc pas nul), continue en...
D3F3- compare ce "U" avec la lettre présente à TXTPTR et qui est un "L", car TXTPTR à été incrémenté en D3CF (avec conversion éventuelle en majuscule)
D401- Ces lettres ne sont pas identiques, continue en...
DA05- où la fin (#00) du mot-clé en cours (OUT) est recherchée dans le tableau principal. Quand Y pointe sur le #00 final il vaut 3 (car 3 octets ont été lus: "U", "T" et #00)
D40A- Le n° d'ordre X est incrémenté et passe à #3B (= #38 + #03)
D40B- L'adresse en 18/19 (début du prochain mot-clé commençant par "O" dans la table principale) est mise à jour en ajoutant la valeur de Y (index ayant servi à trouver le #00 de fin du mot-clé précédent et valant 3) et pointe maintenant sur CAF4.
D3E8- F2 est décrémenté et passe à 1 (pas négatif), Y est remis à #00
D3EE- Y sert à saisir la lettre présente à l'adresse écrite en 18/19 (CAF4). C'est le "L" de OLD qui vaut #4C (pas nul), on continue en...
D3F3- en comparant ce "U" avec la lettre présente à TXTPTR et qui est toujours un "L", car TXTPTR n'a pas été augmenté
D401- Les lettres sont égales, on reboucle en D3EE pour tester la suite...
D3EE- Y = Y + 1 pour pointer sur la lettre suivante du mot-clé en cours (OLD) dans tableau principal. Cette lettre, un "D" (toujours pas nulle), est comparée à la lettre suivante à TXTPTR + Y qui est aussi un "D". On reboucle en D3EE pour tester la suite...
D3EE- Y = Y + 1 pour pointer sur la lettre suivante du mot-clé en cours (toujours OLD) dans le tableau principal. Ce n'est pas une lettre, mais #00 marquant la fin du mot-clé Sédoric, continue en D417 ("exécution")
D417- Le n° d'ordre X qui vaut #3B est multiplié par 2 et passe à #76
DA1A- La table des adresses d'exécution (de CC28 à CCF7) est lue à la position X = #76 et l'adresse qui s'y trouve (en CC9D/CC9E) est lue et empilée. L'adresse trouvée E0AE est l'adresse d'exécution -1 de la commande "OLD"
D422- TXTPTR est mis à jour en ajoutant Y = 3 (nombre de caractères) (Y pointait sur le #00 placé après le OLD du tableau principal des mots-clés)
D425- Finalement "OLD" est appelé par le RTS terminant la routine XCRGET
NB1: Si pas de mot-clé commençant par une initiale donnée, le programme n'est pas dirigé vers l'adresse CCCC, mais, continue

en D3EA vers "non trouvé"

B2: Lorsque le caractère lu n'est pas une lettre de A à Z, le programme est dirigé vers l'adresse CBB5 (fin de la table principale).

ANALYSE D'UN NOM DE FICHIER

Flags utilisés:

C indique si le nom_de_fichier requis peut être ambigu (0) ou non (1)
N à 0 si entrée provient du s/p "non trouvé" de l'interprèteur (en D428)
F2 est la limite de fin de la zone nom (9) ou extension (12) dans BUFNOM
F3 est la longueur du nom_de_fichier (le nombre d'octet restant à lire)
F4 porte le flag N et contient donc #00 si entrée via s/p "non trouvé"
F5 recueille l'octet analysé (si b7 à 1 token BASIC en cours d'analyse)
F6 indique si une extension a été trouvée (b7 à 1 si oui)

Quelques considérations générales:

Un **nom_de_fichier non ambigu (NF)** ne peut ni être omis, ni comporter de jocker, il est formé de [d-]n[.e] d étant une lettre de A à D (drive par défaut si absente), n étant le nom_de_fichier proprement dit (de 1 à 9 caractères), e étant l'extension (de 0 à 3 caractères). Seuls des lettres ou des chiffres peuvent être utilisés. Si le premier caractère de l'extension est un espace (pas de 1er caractère), l'extension par défaut sera utilisée.

Il en est de même pour un **nom_de_fichier ambigu (NFA)**, mais les caractères autorisés comptent en plus * et ?, en outre un NFA peut être omis (exemple DIR) ou se réduire à la seule lettre indiquant le drive à utiliser (exemple DIR A).

Certains groupes de caractères, corresp à un mot-clé BASIC, peuvent avoir été codés et ont été remplacés par le token corresp (octet dont le b7 est à 1). Il faudra donc les décoder, c'est à dire remplacer ce token par les caractères du mot-clé d'origine.

Lorsque l'entrée s'est effectuée via le s/p "non trouvé" de l'interprèteur (b7 de F4 à 1), le NF (chargement direct) ou l'indication de changement de drive (d-) ne comportent pas de "", de même, lorsqu'un NFA est requis (C = 0) et que ce NFA est omis ou reduit à une lettre (de A à D), il n'y a pas de "". La fin du NF ou du NFA est marquée par un espace ou par la virgule qui précède un paramètre ou par le token TO (COPY TO B est valable) ou par la marque de fin d'instruction (zéro ou ":"). L'analyse comporte donc la détection de certains arguments qui peuvent être mêlés aux NF ou aux NFA: ,A EN ,AUTO ,C ,E EN ,J ,L ,N ,T EN ,V et le token TO. Rappel: la virgule n'est pas un caractère autorisé entre les "" d'un nom de fichier.

XNF saisit à TXTPTR un "nom_de_fichier_non_ambigu" et l'écrit dans BUFNOM
Cette entrée est utilisée par les commandes COPYM, CREATEW, ESAVE, KEYSAVE, MERGE, OPEN, SAVE, SAVEO, SAVEM, SAVEU, STATUS et WINDOW.

D44F-	38	SEC	entrée avec C = 1 si "nom de fichier" non-ambigu
D450-	24 18	BIT 18	continue en D452

XNFA saisit à TXTPTR ["]nom_de_fichier_ambigu["] et l'écrit dans BUFNOM
Entrée utilisée par COPY, COPYO, COPYM, DEL, DIR, PROT, REN, SEARCH et UNPROT

D451-	18	CLC	entrée avec C = 0 si "NFA" ou NFA omis ou réduit
D452-	A9 80	LDA #80	A = #80 (flag N = 1, entrée en D44F ou en D451)

Entrée secondaire pour chargement direct ou avec LOAD

a) venant du s/p "non-trouvé" de l'interpréteur Sédoric (D428) (A = #00 et N = 0) (nom_de_fichier ou drive-) avec TXTPTR pointant sur le 1er caractère de cette commande (NF sans "", ni *, ni ?) .

b) venant de la commande LOAD "nom_de_fichier" (A = #80) avec TXTPTR pointant sur le caractère qui suit le D de LOAD, ("NF" sans *, ni ?).

D454-	08	PHP	sauve indicateurs dont C et N
D455-	85 F4	STA F4	le b7 de F4 est maintenant porteur du flag N
D457-	46 F5	LSR F5	mise à 0 du b7 de F5 (à 1 si token en cours)

Réinitialisation de BUFNOM

D459-	AD 09 C0	LDA C009	lit le n° du lecteur par défaut et le place au
D45C-	8D 28 C0	STA C028	préfixe de BUFNOM indiquant le drive à utiliser

BUFNOM comporte 16 octets: **nnnnnnnnneeepstt** (de C029 à C038) où:
n est le **n**om (9 caractères),
e est l'**e**xtension (3 caractères),
ps (2 octets) sont les coordonnées **p**iste/**s**ecteur du descripteur principal et

tt (2 octets) représentent la **t**aille **t**otale du fichier (+ indic. éventuelle PROT)

45F- A2 0B LDX #0B X = 11 soit index pour 12 copies (de 11 à 0)
461- A9 20 LDA #20 code ASCII de l'espace
463- 85 F3 STA F3 #20 -> F3 (pour éviter erreur n° 3 prématurée)
465- 9D 29 C0 STA C029,X
468- CA DEX rempli d'espaces nnnnnnnnneee de BUFNOM
469- 10 FA BPL D465
46B- 28 PLP lorsque X = #FF, récupère indicateurs dont C et N
46C- 10 62 BPL D4D0 si N = 0 branche en D4D0, c'est à dire pour toute entrée venant du s/p "non trouvé" de l'interprèteur Sédoric: chargement direct (NF sans "") ou changement de drive actif (lettre de A à D suivie d'un "-")
46E- B0 3B BCS D4AB branche si "nom_de_fichier_non_ambigu"

Analyse si NFA omis ou réduit à 1 seule lettre de A à D
(seuls cas sans "", lorsque le flag b7 de F4 est à 1)

Rappel: un ["]nom_de_fichier_ambigu["] peut comporter des jokers (* et ?), il peut être omis (DIR) ou réduit à un simple nom de lecteur (DIR A).

1) NFA omis

470- 20 9E D3 JSR D39E XCRGOT relit le caractère à TXTPTR et le convertit en MAJUSCULE
473- D0 0C BNE D481 branche en D481 si ce caractère n'est pas un 0 ou ":" de fin de commande. Si c'est un 0 on est en présence d'un NFA omis (par exemple DIR)
475- A9 0C LDA #0C l'analyse du NFA proprement dit est terminée
477- 85 F2 STA F2 F2 = 12 (pointeur de fin de zone nom + ext)
479- 20 B5 D5 JSR D5B5 rempli de "?" nnnnnnnnneee de BUFNOM

NB: X n'est pas nul en entrée mais vaut #FF (sortie de la boucle D465/D469) donc bogue car le 1er "?" est écrit en C128 au lieu de C029! De plus au retour Z = 0 car le dernier DEX entraine X = #0B (non nul).

47C- F0 03 BEQ D481 branche en D481 si retourne avec Z = 1 (bogue?)
47E- 4C 03 D5 JMP D503 sinon valide drive et termine (jump forcé)
481- C9 2C CMP #2C est-ce une ","? (NFA omis suivi de paramètre)
483- F0 F0 BEQ D475 si oui reboucle en D475 (rempli de "?" et sort)
485- C9 C3 CMP #C3 est-ce le token TO? (COPY omis TO ...)
487- F0 EC BEQ D475 si oui reboucle en D475 (rempli de "?" et sort)

2) NFA réduit à une lettre (drive)

489- 38 SEC prépare soustraction (rappel C = 0 au débuut s/p)
48A- E9 41 SBC #41 convertit 1ère lettre en n° de drive (A = 0 etc.)
48C- A8 TAY copie ce n° de drive potentiel en Y
48D- C9 04 CMP #04 teste si 0 à 3 (A à D)
48F- B0 1A BCS D4AB A >= 4, ce n'est pas une indication de drive
491- 20 98 D3 JSR D398 s'il s'agit d'une lettre de A à D, XCRGET saisit dans A le caractère suivant à TXTPTR, les espaces étant sautés, Z = 1 si fin d'instruction ("0" ou ":"), C = 0 si chiffre (#30 à #39), Y et X inchangés
494- F0 08 BEQ D49E branche si fin d'instruction (et avec C = 1)
496- C9 C3 CMP #C3 est-ce le token TO? (par ex COPY B TO ...)
498- F0 04 BEQ D49E si oui, continue en D49E avec C = 1
49A- C9 2C CMP #2C est-ce une virgule? (cf COPY ... TO B ,C)
49C- D0 05 BNE D4A3 non, branche en D4A3 (oui, continue avec C = 1)
49E- 8C 28 C0 STY C028 n° de drive -> C028 et rebouclage forcé en D475
4A1- B0 D2 BCS D475 (rempli nnnnnnnnneee de "?" et fin analyse NFA)

Non, ce n'était pas un NFA omis ou réduit
(C'est donc un "NFA")
4A3- A5 E9 LDA E9
4A5- D0 02 BNE D4A9 décremente TXTPTR pour annuler le JSR XCRGET
4A7- C6 EA DEC EA (pointe à nouveau sur le 1er caractère du NFA)
4A9- C6 E9 DEC E9

Evalue la chaîne "NF" ou "NFA"
(place sa longueur dans F3 et son adresse dans TXTPTR)

4AB- 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. Cherche une variable alphanumérique ou une chaîne obligatoirement encadrée de "":
4AE- 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

```
D4B1-  85 F3     STA F3     longueur de la chaîne -> F3
D4B3-  A8        TAY        longueur de la chaîne -> Y
D4B4-  88        DEY        teste si au moins 1 caractère
D4B5-  30 7B     BMI D532   la chaîne était vide: "INVALID FILE NAME ERROR"
```

si chaîne pas vide, ajuste la longueur à celle du 1er morceau sans espace:

```
D4B7-  B1 91     LDA (91),Y lit caractères du nom de fichier par la fin
D4B9-  C9 20     CMP #20    est-ce un espace?
D4BB-  D0 02     BNE D4BF   sinon, saute l'instruction suivante
D4BD-  C6 F3     DEC F3     si oui, réduit longueur car pas significatif
D4BF-  88        DEY        vise caractère précédent
D4C0-  10 F5     BPL D4B7   et le lit tant que index pas négatif
D4C2-  A5 E9     LDA E9
D4C4-  48        PHA
D4C5-  A5 EA     LDA EA     sauve TXTPTR actuel sur la pile
D4C7-  48        PHA
D4C8-  A5 91     LDA 91
D4CA-  85 E9     STA E9     et le remplace par l'adresse de la chaîne
D4CC-  A5 92     LDA 92     proprement dite (nom_de_fichier sans "")
D4CE-  85 EA     STA EA
```

Cherche une éventuelle indication de drive

```
D4D0-  20 9E D3  JSR D39E   XCRGOT relit le caractère à TXTPTR et le convertit en MAJUSCULE. Cherche d'abord à
```

déterminer si le nom commence par A- B- C- ou D-

NB: Si lettre, XCRGOT retourne avec C = 1, c'est ce qu'il faut pour la soustraction

```
D4D3-  E9 41     SBC #41    convertit 1ère lettre en n° (A = 0, B = 1 etc...)
D4D5-  AA        TAX        sauve ce n° de drive potentiel dans X
D4D6-  C9 04     CMP #04    teste si 0 à 3 (A à D)
D4D8-  B0 2F     BCS D509   non, ce n'est pas une indication de drive
D4DA-  A0 01     LDY #01    oui, c'est peut-être une indication de drive
D4DC-  B1 E9     LDA (E9),Y indexe le caractère suivant et le lit
D4DE-  C9 2D     CMP #2D    est-ce un "-"?
D4E0-  F0 04     BEQ D4E6   oui, continue en D4E6 (c'était bien un drive)
D4E2-  C9 CD     CMP #CD    est-ce un "-" (token BASIC) (codage possible)
D4E4-  D0 23     BNE D509   non, c'était la 1ère lettre d'un nom de fichier
```

Gère l'indication de drive trouvée (X = n° de drive)

```
D4E6-  8E 28 C0  STX C028   X -> préfixe de BUFNOM (écrase n° par défaut)
D4E9-  C6 F3     DEC F3     réduit la longueur de 2 caractères
D4EB-  C6 F3     DEC F3     (lettre de A à D et "-")
D4ED-  F0 4E     BEQ D53D   si reste rien, "INVALID FILE NAME ERROR"
```

NB: Si entrée par s/p "non trouvé", F3 n'a pas été ajusté avec la longueur réelle de la chaîne, mais mis à #20 (en D463) pour éviter cette erreur n°3

```
D4EF-  20 98 D3  JSR D398   2 fois XCRGET saisit dans A le caractère suivant
D4F2-  20 98 D3  JSR D398   (avance au 3ème caractère pour passer lecteur et "-")
D4F5-  D0 12     BNE D509   continue en D509, si ce n'est pas une "fin d'instruction", sinon, on a peut-être drive-
```

(changement de drive actif)

```
D4F7-  24 F4     BIT F4     teste b7 de F4 (à 0 si entrée par "non trouvé")
D4F9-  30 37     BMI D532   si N = 1, chaîne vide: "INVALID FILE NAME ERROR"
D4FB-  68        PLA        si N = 0, (changement de drive actif par défaut)
D4FC-  68        PLA        retire 2 octets de la pile (TXTPTR inutile)
D4FD-  20 BD D7  JSR D7BD   valide le drive demandé s'il est autorisé
D500-  8E 09 C0  STX C009   et le prend comme lecteur "par défaut" (DRVDEF)
```

Sortie générale du s/p "Analyse d'un nom de fichier"

```
D503-  20 BD D7  JSR D7BD   valide le drive demandé (répétition = bogue?)
D506-  4C 9E D3  JMP D39E   XCRGOT relit le caractère à TXTPTR et le convertit en MAJUSCULE
```

Analyse du NF ou NFA proprement dit

NB: Certaines parties du nom_de_fichier ont pu être codées et sont condensées sous la forme d'un token BASIC. Il faut "décompacter", c'est à dire remplacer tout token par le mot-clé original corresp.

```
D509-  A2 00     LDX #00    X = #00 (X vise lettre analysée, ici lettre n°0)
D50B-  A9 09     LDA #09
D50D-  85 F2     STA F2     F2 = 9 (pointeur de fin de zone nom)
D50F-  46 F6     LSR F6     0 -> b7 de F6 (extension pas encore trouvée)
D511-  20 9E D3  JSR D39E   XCRGOT relit le caractère à TXTPTR et le convertit en MAJUSCULE
```

514- 24 F6 BIT F6 teste b7 de F6 (à 0 par défaut si l'extension n'est pas trouvée)
516- 30 12 BMI D52A si b7 = 1 (ext déjà trouvée), continue en D52A
518- C9 2E CMP #2E le caractère lu est-il un "." (cherche extension)
51A- D0 0E BNE D52A sinon, continue en D52A
51C- 66 F6 ROR F6 si oui (extension trouvée), C = 1 -> b7 de F6
51E- E0 0A CPX #0A X pointe-t-il plus loin que le caractère n°9 ?
520- B0 1B BCS D53D si oui, "INVALID FILE NAME ERROR" X = 9 position maxi du "."
522- A9 0C LDA #0C sinon, OK, mais il faut mettre à jour F2 qui
524- 85 F2 STA F2 doit viser la fin de zone extension (F2 = 12)
526- A2 08 LDX #08 X = #08 pour pointer sur dernier caractère du nom
528- D0 15 BNE D53F branchement forcé en D53F

Recherche d'une éventuelle virgule précédant un paramètre
(et marquant la fin du NF en l'absence de "")

52A- C9 2C CMP #2C le caractère lu est-il une ","? (début de paramètre)
52C- D0 06 BNE D534 sinon, continue en D534 (recherche token etc)
52E- 24 F4 BIT F4 si "," teste b7 de F4 (0 si s/p "" non trouvé)
530- 10 27 BPL D559 si 0 OK, branche en D559 (car implique sans "")
532- 30 78 BMI D5AC sinon "INVALID FILE NAME ERROR" (car implique des "" et les virgules sont interdites à l'intérieur d'un "nom_de_fichier")
534- 20 67 D5 JSR D567 remplace tout token par mot-clé corresp, remplace "*" par des "?" et vérifie si les caractères sont autorisés (A-Z, 0-9)
537- 9D 29 C0 STA C029,X écrit le caractère A dans BUFNOM selon pointeur X
53A- 98 TYA récupère le dernier caractère lu Y dans A
53B- E4 F2 CPX F2 F2 = pointeur de fin de zone nom ou extension
53D- B0 6D BCS D5AC si X >=F2 "INVALID FILE NAME ERROR" (nom trop long)
53F- C6 F3 DEC F3 décrémente le nombre de caractères à analyser
541- F0 10 BEQ D553 s'il ne reste plus rien, vérifie ext et termine
543- E8 INX sinon augmente position du pointeur dans BUFNOM
544- 24 F5 BIT F5 teste b7 de F5 (à 1 si un token est en cours)
546- 30 CC BMI D514 si b7=1 reboucle pour analyse "2e" caractère lu
548- 20 98 D3 JSR D398 si b7=0 XCRGET saisit dans A le caractère suivant
54B- D0 C7 BNE D514 si pas fin instruct, reboucle analyse caractère lu
54D- 24 F4 BIT F4 si fin instruction teste F4 (b7=0 "" non trouvé)
54F- 10 08 BPL D559 s'il est nul, vérifie extension et termine
551- 30 59 BMI D5AC sinon "INVALID FILE NAME ERROR" (car implique des "" et une marque de fin d'instruction est interdite dans un "nom_de_fichier")

Teste le 1er caractère de l'extension
(Si pas valide remplace les 3 caractères par l'extension par défaut et termine)

553- 68 PLA
554- 85 EA STA EA
556- 68 PLA restaure TXTPTR
557- 85 E9 STA E9
559- AD 32 C0 LDA C032 lit le 1er caractère de l'extension dans BUFNOM
55C- C9 20 CMP #20 est-ce un espace?
55E- D0 A3 BNE D503 sinon, c'est fini, sortie générale
560- A2 00 LDX #00 si oui, X = #00 (indexe extension par défaut)
562- 20 4A D3 JSR D34A lit "COM" dans table CCF7, le copie dans BUFNOM
565- F0 9C BEQ D503 revient avec Z = 1: c'est fini, sortie générale

Token, *, ? lettres de A à Z, chiffres de 0 à 9
emplace tout token par le mot-clé corresp, remplace tout "*" par des "?" et vérifie si tous les caractères sont autorisés (?, A-Z et 0-9)

567- 24 F5 BIT F5 teste b7 de F5 (à 1 si token en cours)
569- 30 24 BMI D58F 1 = il y a encore des caractères du mot-clé à lire
56B- A8 TAY caractère lu A -> Y et b7 de A -> N (à 1 si token)
56C- 10 43 BPL D5B1 si N = 0, vérifie que caractère seulement *, ?, lettre ou chiffre et remplace * par ? jusqu'à fin de zone nom ou nom + ext

Recherche du mot-clé BASIC

56E- 85 F5 STA F5 sauve A (qui contient donc un token) dans F5
570- 29 7F AND #7F force à zéro le b7 de A qui contient maintenant le n° d'ordre du mot-clé BASIC (par exemple token #80, END a le n°#00)
572- 85 24 STA 24 A -> 24 (compteur pour trouver le bon mot-clé)
574- A9 E9 LDA #E9 table C0E9 en ROM contient la liste des mots-
576- A0 C0 LDY #C0 clés, chacun se terminant par un ASCII + #80
578- 85 16 STA 16 (NB: table commence par un ASCII + #80)

D57A- 84 17 STY 17 place adresse C0E9 en 16/17 pour lecture en ROM
D57C- A0 00 LDY #00 index à valeur fixe, c'est l'adresse en 16/17 qui augmente
D57E- C6 24 DEC 24 mot-clé suivant
D580- 30 0D BMI D58F branchera lorsque le contenu de 24 deviendra negatif. C'est la seule sortie de ce s/p, avec 16/17 pointant sur le caractère situé juste avant le début du mot-clé cherché dans la table des mots-clés.
D582- E6 16 INC 16
D584- D0 02 BNE D588 incrémente 16/17 (vise le caractère suivant
D586- E6 17 INC 17 dans la table des mots-clés en ROM)
D588- 20 53 04 JSR 0453 lecture à l'adresse pointée en 16/17 plus index Y
D58B- 10 F5 BPL D582 reboucle en D582 si pas nég (cherche le dernier caractère)
D58D- 30 EF BMI D57E reboucle en D57E si négatif (dernier caractère trouvé, décrémente contenu de 24 (n° d'ordre mot-clé) et reprend lecture

Token trouvé, copie le mot-clé dans BUFNOM

Ce s/p lit les caractères 2 par 2 (pour réduire les accès à la ROM). Mais seul le 1er des 2 est aussitôt testé pour savoir si c'est le dernier caractère du mot-clé (b7 à 1). Si c'est le cas le 2ème des 2 n'est pas exploité (c'est la 1ère lettre du token suivant), car le flag b7 de F5 est immédiatement baissé (pas de token en cours). Si le 1er des 2 n'est pas le dernier, le 2ème caractère (qui est toujours dans Y) est alors exploité, car le b7 de F5 est resté à 1 (token en cours).

D58F- A0 00 LDY #00 index Y = 0 pour lecture "1er" caractère
D591- E6 16 INC 16
D593- D0 02 BNE D597 incrémente 16/17 (vise 1er caractère du mot-clé)
D595- E6 17 INC 17
D597- 20 53 04 JSR 0453 lecture à l'adresse pointée en 16/17 plus index Y
D59A- 48 PHA empile le caractère lu
D59B- A0 01 LDY #01 pour lecture octet suivant ("2ème" caractère)
D59D- 20 53 04 JSR 0453 lecture à l'adresse pointée en 16/17 plus index Y
D5A0- A8 TAY sauve le caractère lu dans Y
D5A1- 68 PLA récupére le premier caractère lu (met N à jour)
D5A2- 08 PHP sauvegarde les indicateurs (dont N)
D5A3- 29 7F AND #7F force à zéro le b7 de A (au cas où dernier caractère)
D5A5- 28 PLP récupère les indicateurs (dont N)
D5A6- 10 19 BPL D5C1 si pas le dernier caractère, incrémente F3 = longueur du nom_de_fichier (car au lieu de l'octet du token, on a un ou plusieurs caractères) et vérifie le caractère (seulement "?", lettre ou chiffre)
D5A8- 46 F5 LSR F5 si dernier caract, force N et le b7 de F5 à 0
D5AA- 10 17 BPL D5C3 vérifie que le caractère est seulement "?", lettre ou chiffre

"INVALID FILE NAME ERROR"

D5AC- A2 02 LDX #02 pour "INVALID FILE NAME ERROR"
D5AE- 4C 7E D6 JMP D67E incrémente X et traite l'erreur n° X

Remplace "*" par "?" jusqu'à la fin de la zone nom ou extension

D5B1- C9 2A CMP #2A A est-il une "*"?
D5B3- D0 0E BNE D5C3 sinon, continue en D5C3
D5B5- A9 3F LDA #3F si oui, A = "?"
D5B7- 9D 29 C0 STA C029,X qui est mis dans BUFNOM à la position courante
D5BA- E8 INX vise la position suivante dans BUFNOM et
D5BB- E4 F2 CPX F2 reboucle en D5B7 tant que X < fin de zone
D5BD- D0 F8 BNE D5B7 c'est à dire rempli de "?" la zone corresp à *
D5BF- CA DEX lorsque X = F2, X = X - 1 (X vise dernier caractère de
D5C0- 60 RTS zone nom ou extension) et retourne avec A = "?"
NB: Un "*" écrase tous les caractères qui pourraient suivre dans la zone

Vérifie que le caractère est valide (seulement "?", lettre ou chiffre)

D5C1- E6 F3 INC F3 F3 = longueur du nom_de_fichier
D5C3- C9 3F CMP #3F A est-il un "?"?
D5C5- F0 10 BEQ D5D7 si oui, branche sur un simple RTS
D5C7- C9 30 CMP #30 contient-il un caractère dont code < à celui de "0"?
D5C9- 90 E1 BCC D5AC si oui, "INVALID FILE NAME ERROR"
D5CB- C9 3A CMP #3A contient-il un caractère dont code < à celui de ":"?
D5CD- 90 08 BCC D5D7 si oui, OK c'est un chiffre branche sur RTS
D5CF- C9 41 CMP #41 contient-il un caractère dont code < à celui de "A"?
D5D1- 90 D9 BCC D5AC si oui, "INVALID FILE NAME ERROR"
D5D3- C9 5B CMP #5B contient-il un caractère dont code >= à celui de "Z"?
D5D5- B0 D5 BCS D5AC si oui, "INVALID FILE NAME ERROR"
D5D7- 60 RTS retourne avec A = caractère valide

AUTRES ROUTINES SÉDORIC D'USAGE GÉNÉRAL

XROM Exécute à partir de la RAM une routine ROM

e JSR XROM doit être suivi respectivement et impérativement de l'adresse de la routine V1.0 puis de l'adresse de la routine V1.1

```
5D8- 85 0C     STA 0C
5DA- 84 0D     STY 0D      sauve AY en 0C/0D
5DC- 08        PHP
5DD- 68        PLA
5DE- 85 27     STA 27      sauve P en 27
5E0- 18        CLC
5E1- 68        PLA         prend l'adresse pour RTS sur la pile
5E2- 85 0E     STA 0E      (adresse pointant après le JSR XROM)
5E4- 69 04     ADC #04
5E6- A8        TAY         y ajoute 4 pour sauter les DATA
5E7- 68        PLA         et empile la nouvelle adresse de retour
5E8- 85 0F     STA 0F
5EA- 69 00     ADC #00     l'adresse d'origine (celle des DATA)
5EC- 48        PHA         est gardée en 0E/0F
5ED- 98        TYA
5EE- 48        PHA
5EF- A0 01     LDY #01
5F1- AD 24 C0  LDA C024    si ROM V1.0 alors A = #00 et Y = #01 (octets DATA 1 et 2)
5F4- 10 02     BPL D5F8    si ROM V1.1 alors A = #80 et Y = #03 (octets DATA 3 et 4)
5F6- A0 03     LDY #03
5F8- B1 0E     LDA (0E),Y
5FA- 8D F0 04  STA 04F0    lit adresse s/p ROM à exécuter
5FD- C8        INY         et la place en 04F0/04F1 (EXEVEC)
5FE- B1 0E     LDA (0E),Y
600- 8D F1 04  STA 04F1
603- A4 0D     LDY 0D
605- A5 27     LDA 27
607- 48        PHA         restaure P, A et Y et exécute s/p ROM
608- A5 0C     LDA 0C      dont le RTS utilisera l'adresse empilée
60A- 28        PLP         pointant sur la suite du programme appelant
60B- 4C 71 04  JMP 0471
```

Convertit n° lecteur en lettre et l'affiche

```
60E- 18        CLC         Retenue mise à zéro pour addition
60F- 69 41     ADC #41     A = A + #41 (convertit n° lecteur en lettre)
611- 50 17     BVC D62A    continue en XAFCAR (affiche le caractère ASCII)
```

XAFHEX Affiche en hexadécimal le contenu de A

```
613- 48        PHA         sauve A sur la pile
614- 4A        LSR         A est de la forme xy (de 00 à FF)
615- 4A        LSR         élimine les 4 bits de poids faible
616- 4A        LSR         reste 0x
617- 4A        LSR
618- 20 1E D6  JSR D61E    convert en caractère 0 à 9 ou A à F et affiche
61B- 68        PLA         récupère A
61C- 29 0F     AND #0F     élimine les 4 bits de poids fort, reste 0y
61E- C9 0A     CMP #0A     est-ce un nombre de 0 à 9?
620- 90 02     BCC D624    si oui (C = 0), branche en D624
622- 69 06     ADC #06     sinon (C = 1), A = A + 6 + C (#0A devient #11 etc)
624- 18        CLC         prépare addition suivante
625- 69 30     ADC #30     A = A + #30 (#0A du début devient #41 = "A")
627- 2C A9 20  BIT 20A9    continue en D62A (affiche le caractère ASCII)
628- A9 20     LDA #20     code ASCII de l'espace
```

XAFCAR affiche comme un caractère ASCII le contenu de A

```
62A- C9 0D     CMP #0D     le caractère est-il un CR?
62C- D0 06     BNE D634    sinon, continue en D20E (affiche le caractère présent dans A)
62E- A9 00     LDA #00     si oui, remet à zéro la position de
630- 85 30     STA 30      curseur sur la ligne (écran ou imprimante)
632- A9 0D     LDA #0D     puis reprend A = CR et
634- 4C 0E D2  JMP D20E    continue en D20E (affiche le caractère présent dans A)
```

XAFSTR affiche chaîne terminée par 0 d'adresse AY

```
637- 85 91     STA 91      place l'adresse de la
```

```
D639-  84 92      STY 92        chaîne en 91/92
D63B-  A0 00      LDY #00       met à zéro index de lecture Y
D63D-  B1 91      LDA (91),Y    lit caractère de la chaîne
D63F-  F0 06      BEQ D647      si nul, fini simple RTS
D641-  20 2A D6   JSR D62A      XAFCAR affiche le caractère ASCII contenu en A
D644-  C8         INY           incrémente l'index de lecture
D645-  D0 F6      BNE D63D      et reboucle tant que Y ne revient pas à zéro
D647-  60         RTS           (longueur maxi de chaîne 256 caractères)
```

Affiche " DISC IN DRIVE X AND PRESS 'RETURN'" et get "ESC" ou "RETURN"

```
D648-  A2 14      LDX #14       indexe " DISC IN DRIVE "
D64A-  20 6C D3   JSR D36C      affiche X+1ème message de zone CEE7 terminé par "caractère + 128"
D64D-  AD 00 C0   LDA C000      lecteur actif
D650-  20 0E D6   JSR D60E      convertit n° lecteur en lettre et l'affiche
D653-  20 06 D2   JSR D206      CBF0/ROM va à la ligne
D656-  A2 0D      LDX #0D       indexe "AND PRESS 'RETURN'"
D658-  20 6C D3   JSR D36C      affiche X+1ème message de zone CEE7 terminé par "caractère + 128"
D65B-  58         CLI           autoriser les interruptions
D65C-  20 69 D6   JSR D669      demande un 'ESC' (C = 1) ou un 'RETURN' (C = 0)
D65F-  78         SEI           interdire les interruptions
D660-  08         PHP           sauve les drapeaux 6502
D661-  20 06 D2   JSR D206      CBF0/ROM va à la ligne
D664-  20 06 D2   JSR D206      CBF0/ROM va à la ligne
D667-  28         PLP           restaure les drapeaux
D668-  60         RTS
```

Demande un "ESC" (C = 1) ou un "RETURN" (C = 0)

```
D669-  20 02 D3   JSR D302      JSR EB78/ROM si touche frappée alors N = 1 et A = code ASCII corresp, sinon N = 0
D66C-  C9 1B      CMP #1B       "ESC"?
D66E-  F0 05      BEQ D675      oui: on arrête avec C = 1
D670-  C9 0D      CMP #0D       "RETURN"?
D672-  D0 F5      BNE D669      non reboucle jusqu'à ESC ou RETURN
D674-  18         CLC           C = 0
D675-  60         RTS           retourne avec C = 1 si ESC et 0 si RETURN
```

Idem mais élimine l'adresse de retour si "ESC"

```
D676-  20 69 D6   JSR D669      demande un "ESC" (C = 1) ou un "RETURN" (C = 0)
D679-  90 FA      BCC D675
D67B-  68         PLA
D67C-  68         PLA
D67D-  60         RTS
```

Initialise n° erreur et continue à ERRVEC

(incrémente X et traite erreur n° X)

```
D67E-  E8         INX           qui devient le n° de l'erreur
D67F-  8E FD 04   STX 04FD      variable ERROR (n° de l'erreur)
D682-  6C 1D C0   JMP (C01D)    ERRVEC adresse de traitement des erreurs, ce peut être par exemple la routine D685
                                ci-après
```

Routine de traitement des erreurs

```
D685-  8A         TXA           n° de l'erreur
D686-  20 DE D7   JSR D7DE      place dans la variable EN le n° de l'erreur A
D689-  A5 A8      LDA A8
D68B-  A4 A9      LDY A9        AY = n° de la ligne BASIC courante
D68D-  C0 FF      CPY #FF       teste si A9 = #FF (mode direct)
D68F-  D0 01      BNE D692      si pas mode direct, saute l'instruction suivante
D691-  98         TYA           si mode direct, AY = #FFFF (65535)
D692-  8D FE 04   STA 04FE
D695-  8C FF 04   STY 04FF      04FE/04FF = n° de la ligne de l'erreur
D698-  20 F2 D7   JSR D7F2      place le n° de ligne de l'erreur dans la variable EL
D69B-  20 C4 D1   JSR D1C4      JSR C82F/ROM mettre l'imprimante hors service
D69E-  58         CLI           autorise les interruptions
D69F-  2C 18 C0   BIT C018      teste si le b7 de C018 est à zéro
D6A2-  10 25      BPL D6C9      si oui, continue en D6C9 (erreur non gérée)
D6A4-  AE 23 C0   LDX C023      sinon, X = SAUVES (pointeur de pile)
D6A7-  9A         TXS           que l'on remet en place
D6A8-  AD 1B C0   LDA C01B
D6AB-  AC 1C C0   LDY C01C      n° de la ligne BASIC où il faut reprendre
D6AE-  85 A8      STA A8
D6B0-  84 A9      STY A9        remet en place le n° de la ligne BASIC courante
D6B2-  AD 19 C0   LDA C019
D6B5-  AC 1A C0   LDY C01A      valeur de TXTPTR où il faut reprendre
```

```
6B8-  85 E9      STA E9
6BA-  84 EA      STY EA      remet en place TXTPTR
6BC-  AD 1F C0   LDA C01F
6BF-  AC 20 C0   LDY C020    valeur du pointeur de tampon clavier
6C2-  8D 21 C0   STA C021
6C5-  8C 22 C0   STY C022    remet en place pointeur de tampon clavier
6C8-  60         RTS
```

Affiche l'erreur, réinitialise la pile et retourne au "Ready"

```
6C9-  20 0A D3   JSR D30A    JSR EDE0/ROM autorise IRQ (gestion clavier/curseur)
6CC-  AE FD 04   LDX 04FD    variable ERROR (n° de l'erreur)
6CF-  E0 04      CPX #04     teste si c'est l'erreur n°#04
6D1-  D0 33      BNE D706    sinon, continue directement en D706
6D3-  A2 00      LDX #00     si oui, indexe "LFCRTRACK:"
6D5-  20 6C D3   JSR D36C    affiche X+1ème message de zone CEE7 terminé par "caractère + 128"
6D8-  AD 01 C0   LDA C001    PISTE
6DB-  20 13 D6   JSR D613    XAFHEX affiche en hexadécimal le contenu de A
6DE-  AD 05 C0   LDA C005    type d'erreur (b5 à 0 = WRITE, à 1 = READ FAULT)
6E1-  29 F0      AND #F0     1111 0000 force à 0 les 4 bits faibles
6E3-  49 F0      EOR #F0     1111 0000 inverse les 4 bits forts
6E5-  F0 14      BEQ D6FB    continue en D6FB si le résultat est nul
6E7-  A2 01      LDX #01     sinon, indexe "■SECTOR:"
6E9-  20 6C D3   JSR D36C    affiche X+1ème message de zone CEE7 terminé par "caractère + 128"
6EC-  AD 02 C0   LDA C002    SECTEUR
6EF-  20 13 D6   JSR D613    XAFHEX affiche en hexadécimal le contenu de A
6F2-  A2 03      LDX #03     pour message n°4 "■READ FAULT■"
6F4-  AD 05 C0   LDA C005    type d'erreur (b5 à 0 = WRITE, à 1 = READ FAULT)
6F7-  29 20      AND #20     0010 0000 force à zéro tous les bits sauf b5
6F9-  F0 02      BEQ D6FD    continue en D6FD si le résultat est nul
6FB-  A2 02      LDX #02     indexe "■WRITE FAULT■"
6FD-  20 6C D3   JSR D36C    affiche X+1ème message de zone CEE7 terminé par "caractère + 128"
700-  AD 17 C0   LDA C017    n° de l'I/O error
703-  20 13 D6   JSR D613    XAFHEX affiche en hexadécimal le contenu de A
706-  AE FD 04   LDX 04FD    reprend la variable ERROR (n° de l'erreur)
709-  CA         DEX         et la décrémente
70A-  20 06 D2   JSR D206    CBF0/ROM va à la ligne
70D-  A9 3F      LDA #3F     A = "?"
70F-  20 2A D6   JSR D62A    XAFCAR affiche le caractère ASCII contenu dans A
712-  E0 1A      CPX #1A     teste si X >= #1A
714-  B0 05      BCS D71B    si oui, saute les deux instructions suivantes
716-  20 72 D3   JSR D372    affiche X+1ème message de zone CDBF terminé par "caractère + 128"
719-  30 20      BMI D73B    suite forcée en D73B (b7 dernier caractère à 1)
71B-  E0 31      CPX #31     teste si X < #31 (les erreurs utilisateurs peuvent avoir un n° compris entre #32 (50) et #FF
      (255))
71D-  90 15      BCC D734    (il y a ici une bogue, #32 aurait été mieux!) si oui, continue en D734
71F-  A2 10      LDX #10     sinon, indexe "USER " (même pour l'erreur n°49!)
721-  20 6C D3   JSR D36C    affiche X+1ème message de zone CEE7 terminé par "caractère + 128"
724-  AD FD 04   LDA 04FD    reprend la variable ERROR (n° de l'erreur)
727-  A0 00      LDY #00
729-  8C 4C C0   STY C04C    DEFAFF code ASCII devant les nombres décimaux
72C-  A2 01      LDX #01     pour afficher sur 3 digits n° erreur utilisateur
72E-  20 58 D7   JSR D758    affichage en décimal d'un nombre AY
731-  4C 3B D7   JMP D73B    suite forcée en D73B
734-  8A         TXA
735-  E9 19      SBC #19     calcule X = X - #19
737-  AA         TAX
738-  20 5C D3   JSR D35C    affiche le X+1ème message d'erreur externe terminé par un C+128 (adresse zone en
      C00D/C00E)
73B-  4C 78 D1   JMP D178    C496/ROM réinitialise la pile, affiche " ERROR" et va au "Ready"
```

Curseur visible / curseur caché

```
73E-  38         SEC         C = 1 ira dans b0 de 026A (pour CURSEUR ON)
73F-  24 18      BIT 18      et continue en D741
740-  18         CLC         C = 0 ira dans b0 de 026A (pour CURSEUR OFF)
741-  08         PHP         sauvegarde les indicateurs dont C
742-  4E 6A 02   LSR 026A    éjecte le b0 en décalant tous les bits à droite
745-  28         PLP         récupère les indicateurs dont C
746-  2E 6A 02   ROL 026A    récup C dans b0 en décal tous les bits à gauche
749-  A9 01      LDA #01     effectue 0000 0001 ET registre 026A
74B-  4C 2A D3   JMP D32A    s/p ROM F801 "Eteindre/allumer curseur" (c'est à dire vidéo normale ou inverse) Si le
```

curseur était visible (b0 de 026A à 1) et si A = #01, le curseur sera mis en vidéo inverse sinon vidéo normale

Affichage en décimal sur 2 digits d'un nombre A de #00 à #63 (99)

D74E-	A2 00	LDX #00	X = #00 (pour soustractions successives de 10)
D750-	A0 00	LDY #00	Y = #00 (HH du nombre mis à zéro)
D752-	2C A2 03	BIT 03A2	continue en D758 avec X = #00 et Y = #00

Affichage en décimal sur 5 digits d'un nombre AY de #0000 à #FFFF (65535)

D753-	A2 03	LDX #03	X = #03 (pour soustractions successives de 10000, 1000, 100 et 10)
D755-	2C A2 02	BIT 02A2	continue en D758 avec X = #03 et AY = nombre

Affichage en décimal sur 4 digits d'un nombre AY de #0000 à #270F (9999)

D756-	A2 02	LDX #02	X = #02 (pour soustractions successives de 1000, 100 et 10) continue avec AY = nombre

Entrée secondaire: affichage en décimal d'un nombre AY selon X

Nombre de 0 à 99 pour X = 0, 999 pour X = 1, 9999 pour X = 2 et 65535 pour X = 3. La table CD88/CD8B contient les LL et la table CD8C/CD8F contient les HH des "tranches décimales" 10, 100, 1000 et 10000. Pour convertir en décimal, on va diviser par 10000, 1000, 100 et 10. Ces divisions se feront pour soustractions successives de chaque "tranche décimale" avec comptage dans F2 du nombre de soustractions effectuées et qui donne la valeur du digit corresp. DEFAFF contient le caractère à afficher dans les digits libres devant le nombre décimal (afin que la chaîne de caractères ait toujours la même longueur) (en général un espace). Lors de l'affichage du directory DEFAFF contient "*" et X = #02. On obtient par exemple, les affichages suivants: ****0, *219 ou 1326. Si DEFAFF contient #00, il n'y a pas de caractère par défaut. Cet affichage se fera tant qu'un digit significatif n'a pas été trouvé, ce moment étant signalé par le drapeau C073 (mis à zéro tout au début et qui devient différent de zéro dès qu'un digit significatif est trouvé).

D758-	85 F3	STA F3	sauve A dans F3 (LL du nombre)
D75A-	84 F4	STY F4	et Y dans F4 (HH du nombre)
D75C-	A9 00	LDA #00	mise à zéro de C073
D75E-	8D 73 C0	STA C073	(flag 1ère soustraction du 1er tour)
D761-	A9 FF	LDA #FF	prépare F2 pour #00 en début de boucle
D763-	85 F2	STA F2	(compteur nombre de soustractions effectuées)
D765-	E6 F2	INC F2	part de 0, mais comptera une de trop, donc le compte est bon
D767-	38	SEC	prépare soustraction
D768-	A5 F3	LDA F3	récupère F3 (LL du nombre) et le
D76A-	A8	TAY	sauve dans Y (valeur précédant la soustraction)
D76B-	FD 88 CD	SBC CD88,X	en soustrait LL de tranche décimale (la + forte
D76E-	85 F3	STA F3	au début) et remet F3 en place
D770-	A5 F4	LDA F4	récupère F4 (HH du nombre) et le sauve
D772-	48	PHA	sur la pile (valeur précédant la soustraction)
D773-	FD 8C CD	SBC CD8C,X	en soustrait HH de tranche décimale (la + forte
D776-	85 F4	STA F4	au début) et remet F4 en place
D778-	68	PLA	YA contient valeur de F3/F4 avant soustraction
D779-	B0 EA	BCS D765	branche s'il faut encore retirer
D77B-	84 F3	STY F3	sinon (trop retiré), remet en place AY
D77D-	85 F4	STA F4	la dernière valeur valable de F3/F4
D77F-	A5 F2	LDA F2	A = nombre de soustractions effectuées
D781-	F0 05	BEQ D788	si F2 nul (1ère soustraction), branche en D788
D783-	8D 73 C0	STA C073	si F2 pas nul, copie la valeur de F2 en C073
D786-	D0 09	BNE D791	qui devient <> 0 et continue en D791
D788-	AC 73 C0	LDY C073	récup valeur flag C073 pour test du tour précédent
D78B-	D0 04	BNE D791	si pas nulle, continue en D791
D78D-	AD 4C C0	LDA C04C	si nulle, pas encore trouvé de digit significatif
D790-	2C 09 30	BIT 3009	A = valeur de DEFAFF et continue en D793
D791-	09 30	ORA #30	(0011 0000) force à 1 b4 et b5 de A (convertit un nombre de 0 à 9 en code ASCII de #30 à #39)
D793-	20 2A D6	JSR D62A	XAFCAR affiche le caractère ASCII contenu dans A
D796-	CA	DEX	pour soustract tranche décimale suivante (+ faible)
D797-	10 C8	BPL D761	et reboucle tant que X n'est pas < 0
D799-	A5 F3	LDA F3	A = reste de la ou des divisions
D79B-	4C 24 D6	JMP D624	affiche dernier digit et retourne

Recherche "?" dans BUFNOM revient avec C = 1 si pas trouvé

D79E-	38	SEC	entrée avec C = 1 (jokers interdits)
D79F-	24 18	BIT 18	et continue en D7A1
D7A0-	18	CLC	entrée avec C = 0 (jokers autorisés)
D7A1-	66 F2	ROR F2	place C dans b7 de F2
D7A3-	A2 0B	LDX #0B	X = 11 (vise le dernier caractère de BUFNOM)
D7A5-	BD 29 C0	LDA C029,X	lit caractère pointé dans BUFNOM
D7A8-	C9 3F	CMP #3F	est-ce un "?"?
D7AA-	F0 05	BEQ D7B1	si oui, branche en D7B1
D7AC-	CA	DEX	caractère précédent
D7AD-	10 F6	BPL D7A5	reboucle tant que X n'est pas négatif
D7AF-	38	SEC	et retourne avec C = 1 (pas trouvé de joker)

```
7B0-  60          RTS          ou avec C = 0 (joker trouvé, mais autorisé)
7B1-  26 F2       ROL F2       récupère C = 0 ou 1 selon entrée
7B3-  90 FB       BCC D7B0     simple RTS si C = 0 (jokers autorisés)
7B5-  A2 05       LDX #05      pour "WILDCARD(S) NOT ALLOWED ERROR"
7B7-  2C A2 01    BIT 01A2     et traite cette erreur en D67E
7B8-  A2 01       LDX #01      pour "DRIVE NOT IN LINE ERROR"
7BA-  4C 7E D6    JMP D67E     et traite cette erreur en D67E
```

Vérifie si drive demandé est "on line" et le valide "actif"

```
7BD-  AC 28 C0    LDY C028     1er octet de BUFNOM = n° du lecteur demandé
7C0-  8C 00 C0    STY C000     on le met dans DRIVE (n° du lecteur actif)
7C3-  B9 39 C0    LDA C039,Y   vérifie dans TABDRV que ce lecteur est "online"
7C6-  F0 F0       BEQ D7B8     si #00, "DRIVE NOT IN LINE ERROR"
7C8-  60          RTS          si lecteur est "online" retourne
```

Recherche et mise à jour des variables système

En entrée X indexe dans la table des variables système CD94/CDBF, 2 lettres qui représentent les 2 caractères significatifs de la variable

```
7C9-  A2 0E       LDX #0E      X = #0E et continue en D7EF pour RA
7CB-  2C A2 10    BIT 10A2
7CC-  A2 10       LDX #10      X = #10 et continue en D7EF pour RX
7CE-  2C A2 12    BIT 12A2
7CF-  A2 12       LDX #12      X = #12 et continue en D7EF pour RY
7D1-  2C A2 14    BIT 14A2
7D2-  A2 14       LDX #14      X = #14 et continue en D7EF pour RP
7D4-  2C A2 16    BIT 16A2
7D5-  A2 16       LDX #16      X = #16 et continue en D7EF pour EF
7D7-  2C A2 06    BIT 06A2
```

Place dans la variable OM le n° du mode de sortie

```
7D8-  A2 06       LDX #06      X = #06 et continue en D7EF pour OM
7DA-  2C A2 04    BIT 04A2
7DB-  A2 04       LDX #04      X = #04 et continue en D7EF pour IN
7DD-  2C A2 00    BIT 00A2
```

Place dans la variable EN le n° de l'erreur A

```
7DE-  A2 00       LDX #00      X = #00 et continue en D7EF pour EN
7E0-  2C A2 0A    BIT 0AA2
7E1-  A2 0A       LDX #0A      X = #0A et continue en D7EF pour FT
```

Les trois autres variables de STATUS: ST, ED et EX étant en fait indexées par Y = #18, #1A et #1C (voir ci-dessous)

```
7E3-  2C A2 1E    BIT 1EA2
7E4-  A2 1E       LDX #1E      X = #1E et continue en D7EF pour CX
7E6-  2C A2 20    BIT 20A2
7E7-  A2 20       LDX #20      X = #20 et continue en D7EF pour CY
7E9-  2C A2 22    BIT 22A2
7EA-  A2 22       LDX #22      X = #22 et continue en D7EF pour FP
7EC-  2C A2 24    BIT 24A2
7ED-  A2 24       LDX #24      X = #24 et continue en D7EF pour FS
                               Les variables EL, SK, ST, ED, EX, EO sont indexées par Y:
7EF-  A0 00       LDY #00      Y = #00 et continue en D803 pour toutes ces dernières
7F1-  2C A2 02    BIT 02A2
```

Place dans la variable EL le n° de ligne de l'erreur AY

```
7F2-  A0 02       LDY #02      Y = #02 et continue en D803 pour EL
7F4-  2C A2 08    BIT 08A2
7F5-  A0 08       LDY #08      Y = #08 et continue en D803 pour SK
7F7-  2C A2 18    BIT 18A2
7F8-  A0 18       LDY #18      Y = #18 et continue en D803 pour ST
7FA-  2C A2 1A    BIT 1AA2
7FB-  A0 1A       LDY #1A      Y = #1A et continue en D803 pour ED
7FD-  2C A2 1C    BIT 1CA2
7FE-  A0 1C       LDY #1C      Y = #1C et continue en D803 pour EX
800-  2C A2 0C    BIT 0CA2
801-  A0 0C       LDY #0C      Y = #0C et continue en D803 pour EO (bogue: semble inutilisée)
803-  85 F2       STA F2       sauve A en F2 (LL de valeur de la variable)
805-  BD 94 CD    LDA CD94,X   lit deux caractères
808-  85 B4       STA B4       dans la table des variables
80A-  BD 95 CD    LDA CD95,X   et les place en B4/B5
80D-  85 B5       STA B5       (caractères significatifs d'une variable)
80F-  98          TYA          Y précédem chargé -> A, c'est à dire #00 (HH de la variable) ou index pour 1 variable
```

secondaire (dans ce cas, A qui contient cet index sera copié en D1 (poids fort de ACC1), puis remis à zéro par le s/p ROM DF40 ci-après

D810- A4 F2 LDY F2 récupère dans Y la valeur qu'avait A en entrée, c'est à dire LL de la valeur de la variable en entrée
D812- 20 CA D2 JSR D2CA JSR DF40/ROM transfère un nombre non signé YA dans ACC1 (floating point accumulator)
D815- 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)
D818- AA TAX l'adresse où mettre la valeur est maintenant en XY
D819- 4C C2 D2 JMP D2C2 JSR DEAD/ROM recopie les 5 octets de ACC1 de l'adresse XY à l'adresse XY + 4. A l'issue de ce s/p, X (qui est inchangé) semble pointer dans le tableau CD94/CDBF sur la variable suivante, indexée à l'entrée par Y!

PRENDRE UN CARACTÈRE AU CLAVIER
(REMPLACE EB78 EN ROM)

Afin de pouvoir traiter les touches de fonctions (voir le manuel Sédoric page 54, 55 et 102), la gestion des touches a été modifiée. Sous Sédoric, le vecteur JMP EB78 (en 023B/023D) est remplacé par JMP 045B. En 045B, Sédoric passe sur RAMOV, exécute la routine **D845** (incluant l'ancien EB78/ROM) et revient sur ROM.

Que fait donc le s/p **EB78/ROM**? Il examine le tampon de touche 02DF. Lorsqu'une touche est pressée, le b7 de 02DF passe à 1, tandis que les b6 à b0 contiennent le code ASCII corresp. Le s/p EB78 met ce code ASCII dans A, force N à 1 et remet 02DF à 0. Lorsqu'aucune touche n'a été pressée, le s/p EB78 retourne avec N = 0. X et Y ne sont pas touchés.

Que fait de plus le s/p **D845/RAMOV**? Il examine si une touche de fonction a été pressée (combinaison FUNCT+touche ou FUNCT+SHIFT+touche). Si ce n'est pas le cas, il termine comme ci-dessus, avec en plus le b7 de C049 à 0 (pas de touche de fonction en cours). Si une touche de fonction a été pressée, il recherche la chaîne de caractères qui correspond au code de fonction associé à la combinaison de touches pressées. Puis termine avec le premier caractère dans A, N à 1, et b7 de C049 à 1, ce qui signifie "il y a d'autres caractères à saisir". Lors du prochain appel au s/p D845, le programme examine l'état du b7 de C049, le trouvant à 1, il ira directement lire le caractère suivant et ainsi de suite jusqu'au dernier. Lorsqu'au cours d'un appel au s/p D845, le programme arrive au dernier caractère de la chaîne du code de commande qui était en cours de traitement, il remet le b7 de C049 à 0 et retourne avec le dernier caractère dans A et avec N à 1.

Ce s/p comporte deux entrées, qui se font un peu plus loin en D843 (pour LINPUT) et en D845 (tous les autres cas).

Lecture d'un octet en ROM ou en RAMOV selon adresse en 16/17

81C- E6 16 INC 16
81E- D0 02 BNE D822 incrémente l'adresse de lecture en 16/17
820- E6 17 INC 17
822- A0 00 LDY #00 index Y = 0 pour lecture à l'adresse en 16/17
824- 2C 48 C0 BIT C048 teste b6 de C048 (si 0, c'est une commande Sédoric donc lecture en RAMOV; si 1, commande BASIC donc lecture en ROM)
827- 50 03 BVC D82C si RAMOV visée, saute la ligne suivante
829- 4C 53 04 JMP 0453 si ROM visée, passe par page 4 (bascule sur ROM, effectue un LDA (16),Y puis bascule sur RAMOV et RTS au point d'appel du s/p D81C)

82C- B1 16 LDA (16),Y A = contenu cellule dont l'adresse est en 16/17
82E- F0 3F BEQ D86F branche si A = 0 (fin d'un mot-clé Sédoric)
830- 10 3F BPL D871 simple RTS si le b7 de l'octet lu A est à 0
832- 2C 48 C0 BIT C048 si le b7 de l'octet lu A est à 1, cela marque soit la fin d'une commande redéfinissable ou prédéfinie, soit un token BASIC incorporé dans un mot-clé Sédoric, car les commandes BASIC ont déjà été déviées en D829. Pour le savoir, on teste le b7 de C048 (à 0 si c'est une commande redéfinissable ou prédéfinie écrite en toutes lettres, sans token ou à 1 si c'est un token BASIC incorporé dans un mot-clé Sédoric).
835- 10 38 BPL D86F si commande redéfinie ou prédéfinie, continue en D86F
837- 29 7F AND #7F si token BASIC incorporé, force à 0 le b7 de A afin de "retomber" sur un nombre compris entre 0 et 127 et par conséquent N = 0
839- 60 RTS puis retourne

Teste si touche corresp à n° de colonne et de ligne a été pressée

83A- 8D 00 03 STA 0300 place le n° de ligne en 0300 (port B)
83D- A9 08 LDA #08 0000 1000 = masque pour tester b3 (réponse)
83F- 2D 00 03 AND 0300 donne A = 0000 x000 avec x = 1 si touche corresp
842- 60 RTS pressée, soit A = #08 et Z = 0

Entrée appelée par LINPUT

843- 38 SEC C = 1, entrée utilisée uniquement par LINPUT
844- 24 18 BIT 18 (F2 = longueur spécifiée) et continue en D846

Entrée générale principale

845- 18 CLC C = 0, entrée utilisée dans tous les autres cas: appels dérivés de la ROM (vecteur 045B), ainsi que WINDOW, TYPE et BUILD.

846- 6E 4A C0 ROR C04A flag point d'entrée: le b7 de C04A reçoit C et passe donc à 1 si LINPUT, sinon passe à 0, notamment pour les appels de la ROM
849- 20 02 D3 JSR D302 JSR EB78/ROM si touche frappée alors N = 1 et A = code ASCII corresp, sinon N = 0
84C- 08 PHP sauve P et notamment N, flag de touche pressée

D84D- 8D 46 C0 STA C046 sauve A = code ASCII corresp à la touche
D850- 8E 47 C0 STX C047 sauve X d'origine (nombre de caractères à l'entrée)
D853- 2C 49 C0 BIT C049 teste b7 (à 1 si code de fontion en cours). Il est bien temps: 4 instructions inutiles ont déjà été effectuées! Pourquoi ne pas avoir testé en premier si une chaîne associée à une touche de fonction est en cours de saisie? Il fallait placer cette instruction en D849, à la place du JSR D302...
D856- 10 1A BPL D872 si nul, branche en D872 (cours normal du programme)

Un code de fonction est en cours de saisie

D858- 2C 4A C0 BIT C04A teste le b7 du flag C04A (flag point d'entrée)
D85B- 30 07 BMI D864 continue en D864 si s/p appelé depuis LINPUT

Vérification spéciale pour les appels de la ROM
Les s/p de la ROM utilisent un tampon clavier de 78 caractères au maximum. L'utilisation des touches de fonctions pourrait conduire à saturer ce tampon puisque qu'un simple appui correspond à plusieurs caractères (16 au maximum). Les appels provenant de WINDOW, TYPE et BUILD subissent aussi cette vérification, mais sans conséquence, X étant géré autrement.

D85D- E0 4E CPX #4E teste si 78 caractères entrés (maxi pour ROM)
D85F- 90 03 BCC D864 sinon, le buffer n'est pas encore plein, saute l'instruction suivante
D861- 4E 49 C0 LSR C049 si oui, le tampon est plein, force à 0 le b7 de C049 (flag "pas de code de fonction en cours") afin d'interrompre tout rajout de caractères. Cette solution est assez cavalière et doit générer des "SYNTAX ERROR"!

Saisie de la suite de la chaîne associée au code de fonction

D864- 20 1C D8 JSR D81C incrémente l'adresse en 16/17 et lit un octet dans la chaîne
D867- 10 03 BPL D86C si N = 0, saute ligne suivante (fin commande non atteinte)
D869- 4E 49 C0 LSR C049 si N = 1, fin de commande, force à 0 le b7 de C049 (flag "pas de code de fontion en cours")
D86C- 29 7F AND #7F force à 0 le b7 de A (octet lu à adresse en 16/17)
D86E- 28 PLP récupère P mais, simultanément, remet à jour N:

Sortie après mise à 1 de N

D86F- 24 E2 BIT E2 Il n'existe pas de mode immédiat pour l'instruction BIT, d'où le truc suivant: E2 contient toujours la valeur #E6 (1110 0110), ce qui entraîne N = 1 (flag "une touche a été pressée").
D871- 60 RTS et retourne

Pas de code de fonction en cours

D872- 28 PLP récupère P
D873- 10 FC BPL D871 Sortie si N = 0 (pas de touche pressée)
D875- A9 00 LDA #00 RAZ de C04B et de C048 qui sont utilisés:
D877- 8D 4B C0 STA C04B - pour sauver 1ère lettre d'un mot-clé Sédoric
D87A- 8D 48 C0 STA C048 - comme flag "type de code de commande". Son b7 est à 0 s'il s'agit d'une commande redéfinissable (code #00 à #0F) ou prédéfinie (code #10 à #1F). Sinon, il s'agit soit d'une commande Sédoric (b6 à 0, code #20 à #7F), soit d'une commande BASIC (b6 à 1, code de #80 à #FD).

Gestion des touches de fonction (touche + FUNCT ou touche + FUNCT + SHIFT)

Teste si la touche FUNCT a été pressée

D87D- A9 0E LDA #0E = 14 = n° de registre du PSG 8912 (Programmable Sound Generator) (A = 1 à 13 pour son, et 14 pour le clavier = port A)
D87F- A2 EF LDX #EF = 1110 1111 où le b4 est à 0, pour activer la colonne 4, c'est à dire celle des touches CTRL, FUNCT, SHIFT G et DROIT.
D881- 20 22 D3 JSR D322 JSR F590/ROM met X dans le registre A du PSG
D884- A9 15 LDA #15 pour placer 0001 0101 sur le port B. Dans cette valeur, la signification de chaque bit est la suivante:

b0 à b2 = n° de ligne ici 5,
b3 = 0 passera à 1 si la touche corresp est pressée,
b4 = 1 est le signal STROBE de l'imprimante (1 = inactif),
b5 = 0 non utilisé normalement (cette ligne a été récupérée pour gérer les "cartouches PB5")
b6 = 0 relai magnéto inactif (ouvert),
b7 = 0 sortie k7 (1 serait mieux, "L'Oric à Nu" page 338).

La touche testée, dont le code est écrit en 0209, répond à la forme binaire **V0CCCLLL** (voir "L'Oric à Nu" page 339) dans lequel **CCC** est le N° de colonne ici 4 = 100, **LLL** le n° de ligne ici 5 = 101 et **V** = 1 si la touche est pressée. Le bit b6 est toujours à 0. Ici on a 1010 0101 soit #A5 qui est le code de la touche FUNCT (voir le manuel Sédoric page 104).

D886- 20 3A D8 JSR D83A teste si la touche FUNCT a été pressée

889- D0 38 BNE D8C3 si FUNCT a été pressée, branche en D8C3 car b3 est passé à 1

Touche ordinaire: teste si AZERTY ou QWERTY

88B- AD 46 C0 LDA C046 sinon, pas besoin de gérer touche de fonction,
88E- AE 47 C0 LDX C047 récupère A = code ASCII et X = nombre de caractères
891- 2C 3D C0 BIT C03D force V et N selon MODCLA (b6=ACCENT b7=AZERTY)
894- 10 D9 BPL D86F si QWERTY (mode par défaut), termine en D86F
896- AD 08 02 LDA 0208 si AZERTY il faut intervertir certaines lettres
899- A2 05 LDX #05 compare le code de touche saisi avec ceux de la
89B- DD 41 CD CMP CD41,X table CD41 (6 codes pour ; M Z A W et Q)
89E- F0 0C BEQ D8AC code trouvé, branche en D8AC pour conversion
8A0- CA DEX code pas trouvé, décrémente index X et examine
8A1- 10 F8 BPL D89B le précédent, tant qu'il en reste (X >= 0)
8A3- AD 46 C0 LDA C046 rien trouvé, récupère A = code ASCII de touche
8A6- AE 47 C0 LDX C047 récupère X = nombre de caractères dans buffer
8A9- 4C 6F D8 JMP D86F sort en D86F: ce n'était alors pas une lettre "critique"

Conversion QWERTY / AZERTY

8AC- AD 08 02 LDA 0208 empile le code de la touche pressée (en 0208
8AF- 48 PHA se trouve le code des touches "normales")
8B0- BD 47 CD LDA CD47,X lit le code de touche corresp en AZERTY
8B3- 8D 08 02 STA 0208 et le met à la place du code touche pressée
8B6- 20 1A D3 JSR D31A JSR F4EF/ROM "Trouver le code ASCII"
8B9- AA TAX X = code ASCII corresp au code de remplacement
8BA- 68 PLA récupère l'ancien code de touche
8BB- 8D 08 02 STA 0208 et le remet en place
8BE- 8A TXA A = code ASCII corresp au code de remplacement
8BF- 29 7F AND #7F masque 0111 111, force à 0 le b7 de A
8C1- 10 E3 BPL D8A6 rebouclage forcé en D8A6 (récupère X et termine)

Teste si une touche SHIFT a aussi été pressée

8C3- A9 17 LDA #17 pour placer 0001 0111 sur le port B (dans cette valeur: b0 à b2 = n° de ligne correspond à la ligne 7 du clavier). D'autre part, la colonne 4 est toujours activée. Le code de la touche testée vaut donc V0CCCLLL = 1010 0111 = #A7 qui est le code de la touche SHIFT DROIT.
8C5- 20 3A D8 JSR D83A teste si la touche SHIFT DROIT a été pressée
8C8- D0 07 BNE D8D1 oui, branche en D8D1; non, teste l'autre shift
8CA- A9 14 LDA #14 0001 0100 correspond à ligne 4 de la colonne 4 soit code = 1010 0100 = #A4 qui est le code de la touche SHIFT GAUCHE
8CC- 20 3A D8 JSR D83A teste si la touche SHIFT GAUCHE a été pressée
8CF- F0 02 BEQ D8D3 sinon, saute la ligne qui suit (A = 0000 0000)
8D1- A9 40 LDA #40 si oui, A = 0100 0000 (FUNCT+SHIFT, met b6 à 1)
8D3- 0D 08 02 ORA 0208 effectue A OU code de touche (10CCCLLL)
NB: Les codes **V0CCCLLL** vont donc de #80 (1000 0000) à #BF (1011 1111) si FUNCT+touche et de #C0 (1100 0000) à #FF (1111 1111) si FUNCT+SHIFT+touche
8D6- 29 7F AND #7F masque 0111 1111, force à 0 le b7. A vaut donc de #00 (0000 0000) à #3F (0011 1111) si FUNCT+touche et de #40 (0100 0000) à #7F (0111 1111) si FUNCT+SHIFT+touche.

Rappel des codes de touche (manuel Sédoric page 104):

Codes pour		Index de lecture pour	
touche seule	0 1 2 3 4 5 6 7 8 9 A B C D E F	touche + FUNCT	touche+ FUNCT+SHIFT
#80 à #8F #90 à #9F #A0 à #AF #B0 à #BF	7 J M K ■ U Y 8 N T 6 9 , I H L 5 R B ; . O G 0 V F 4 - ↑ P E / m m c m g f m s 1 e Z m ← d A r X Q 2 \ ↓] S m 3 D C ' → [W =	#00 à #0F #10 à #1F #20 à #2F #30 à #3F	#40 à #4F #50 à #5F #60 à #6F #70 à #7F
avec: c CTRL, d DEL, e ESC, f FUNCT, g SHIFT Gauche, m code manquant, r RETURN, s SHIFT DROIT, et ■ SPACE. Exemple: #80 correspond à la touche "7" et #BF à la touche "=". Certains codes (exemple #A0) ne correspondent à aucune touche.			

Rappel de la table "KEYDEF":

C800- 09 AA F6 F1 17 8C AC 07 C808- 90 C9 0F 0A 00 99 DC F4 C810- 05 8B B1 00 00 B4 97 0B C818- EB 8D 03 1C 81 E6 C8 84 C820- 00 00 00 00 00 00 00 00 C828- 00 1C 09 00 BF FE D1 FF C830- B6 A7 01 12 C1 1D EA 00 C838- 02 E7 94 0E BC 0D AE 13 C840- 10 AB A8 BD 00 D9 AD 11 C848- CA C3 00 00 00 92 9E F5 C850- 00 9C B2 00 00 D2 9B 0C C858- E9 AF 00 00 BB B9 80 85 C860- 00 00 00 00 00 00 00 00 C868- 08 00 91 00 00 00 EC FF C870- B7 A9 04 06 00 00 F2 00 C878- 09 8A ED 00 C7 0C B0 1C	7 J M K ■ U Y 8 N T 6 9 , I H L 5 R B ; . O G 0 V F 4 - ↑ P E / m m c m g f m s 1 e Z m ← d A r X Q 2 \ ↓] S m 3 D C ' → [W = 7 J M K ■ U Y 8 N T 6 9 , I H L 5 R B ; . O G 0 V F 4 - ↑ P E / m m c m g f m s 1 e Z m ← d A r X Q 2 \ ↓] S m 3 D C ' → [W =

Avec: c CTRL, d DEL, e ESC, f FUNCT, g SHIFT Gauche, m code manquant, r RETURN, s SHIFT DROIT, et ■ SPACE. Exemples: la touche FUNCT et la touche] permettent d'obtenir directement l'instruction DIR et un RETURN (commande prédéfinie n°#1D, voir ci-dessous), déclanchant alors l'affichage du catalogue à l'écran. De même, à la combinaison FUNCT+SHIFT+\ est affectée le code de fonction #06 corresp à la commande redéfinissable "LIST1000-" suivie de RETURN. Les nombreux #00 de cette table correspondent à "?HEX$(DEEK(#".

Remarques fondamentales

La lecture de cette table montre qu'il y régne la plus grande anarchie. Certains codes y existent en plusieurs exemplaires. #00 s'y trouve 28 fois. En fait, certains #00 correspondent aux touches "manquantes" (voir les "m" dans le premier tableau ci-dessus). Mais il y a quand même 17 codes #00 qui peuvent être réutilisés, sans compter de multiples #09, #0C et #1C. D'autres codes sont complètement absents: il n'y a aucun code de #20 à #7F (mots-clés Sédoric) et il manque 9 des 16 commandes prédéfinies (pourquoi alors les avoir prédéfinies!) etc...

L'attribution des diverses fonctions aux 256 codes de fonctions est elle même anarchique. D'une part, 126 codes (de #80 à #FD) ont été alloués aux 119 mots-clés BASIC (token de #80 à #F6) les 7 derniers codes de fonctions (de #F7 à #FD) sont donc inutilisés. D'autre part, la table des mots-clés Sédoric comporte 104 entrées (dont certaines en double à cause d'un codage intempestif lorsque ces mots-clés sont tapés en majuscules et qu'ils contiennent un mot-clé BASIC) or de #20 à #7F il n'y a que 96 codes, résultat: les dernières commandes Sédoric ne sont pas accessibles!

Comme vous allez le voir plus loin le s/p traitant des codes corresp aux mots-clés Sédoric est complètement bogué et ne marche pas. Il est clair qu'au dernier moment les codes Sédoric ont été remplacés par des #00 dans le tableau C800!

L'utilisateur ne peut en principe intervenir qu'au moyen des commandes KEYUSE (permettant de redéfinir les 16 premières commandes de code #00 à #0F) et KEYDEF (permettant de réattribuer les "256" codes de commandes disponibles aux diverses combinaisons FUNCT+touche et FUNCT+SHIFT+touche). Il est évident qu'il faudrait revoir les commandes prédéfinies où les HIRES et autres RUN, accessibles directement par leur token BASIC, devront être remplacés par les 5 commandes Sédoric UNPROT, VUSER, WIDTH, WINDOW et RESTORE qui sont complètement inaccessibles. Nous pensons revoir tout cela dans la version 3 du Sédoric!

Analyse du type de commande

Les codes de commandes (page 102 du manuel Sédoric) sont de 4 types:

n°#00 à #1F: commandes redéfinissables (#00 à #0F) et prédéfinies (#10 à #1F)

n°#20 à #7F: Mots-clés Sédoric. On ne peut accéder aux derniers mots-clés (voir page 103).
Ce n° est ramené de #00 à #5F en soustrayant #20 pour obtenir le n° d'ordre des mots-clés Sédoric

n°#80 à #FD: Mots-clés BASIC.
Ce n° est ramené de #00 à #DD en soustrayant #80 pour obtenir le n° d'ordre des mots-clés BASIC

n°#FE et #FF qui sont traités à part (DEL TAMPON et NUM AUTO).

A chaque combinaison FUNCT+touche ou FUNCT+SHIFT+touche correspond un de ces codes de commandes.

Cette correspondance est donnée par la table "KEYDEF" située en C800 et rappelée ci-dessus, qui liste les #40 codes correspondant aux #40 combinaisons FUNCT+touche, puis les #40 codes corresp aux #40 combinaisons FUNCT+SHIFT+touche.

8D8-	AA	TAX	X est un index pour
8D9-	BD 00 C8	LDA C800,X	lire la table des codes de fonctions
8DC-	A8	TAY	copie code lu dans Y (pour tester si #FE ou #FF)
8DD-	C8	INY	incrémente Y pour tester si c'était #FF (car #FF + 1 = #00 et C = 1)
8DE-	D0 03	BNE D8E3	sinon, saute la ligne suivante
8E0-	4C 63 D9	JMP D963	si oui, continue vers le s/p NUM AUTO
8E3-	C8	INY	incrémente Y pour tester si c'était #FE (car #FE + 2 = #00)
8E4-	F0 6C	BEQ D952	si oui, continue vers le s/p DEL TAMPON
8E6-	C9 20	CMP #20	met C à 0, si A < #20 ou C à 1 si A >= #20
8E8-	6A	ROR	C->b7...b0->C (c'est à dire force b7 selon C)
8E9-	8D 48 C0	STA C048	et sauve le résultat dans C048 (type de code de fonction: b7 à 1 = flag pour code >= #20; b6 à 1 = flag pour code >= #80). Le b6 indique donc aussi qu'il faudra lire le mot-clé cherché en ROM (b6 à 1 pour la table des mots-clés BASIC en C0EA) ou en RAMOV (b6 à 0 pour la table des commandes redéfinissables en C880 ou celle des commandes prédéfinies en C980 ou encore celle des mots-clés Sédoric en C9DE).
8EC-	2A	ROL	C<-b7...b0<-C (restaure les valeurs initiales)
8ED-	30 04	BMI D8F3	saute soustraction suivante si A>=#80 (token BASIC)
8EF-	90 02	BCC D8F3	saute la soustraction suivante si A<#20 (commande redéfinie ou prédéfinie)
8F1-	E9 20	SBC #20	restent les codes Sédoric (#20 à #7F) dont on calcule le n° d'ordre (#00 à #5F) en retirant #20
8F3-	29 7F	AND #7F	0111 1111 force à 0 le b7 de A (ne s'applique en fait qu'aux token BASIC dont on calcule le n° d'ordre de #00 à #7F)
8F5-	AA	TAX	copie dans X le n° d'ordre pour servir
8F6-	A9 E9	LDA #E9	d'index plus loin
8F8-	A0 C0	LDY #C0	AY = C0E9 (table des mots-clés BASIC en ROM)
8FA-	2C 48 C0	BIT C048	positionne N et V selon b7 et b6 (type de code)
8FD-	70 29	BVS D928	continue en D928 si b6 à 1 (token BASIC)
8FF-	30 06	BMI D907	continue en D907 si b7 à 1 (mots-clés Sédoric)
901-	A9 7F	LDA #7F	restent codes de #00 à #1F (commandes redéfinies et prédéfinies)
903-	A0 C8	LDY #C8	AY = C87F (table des commandes redéfinies et prédéfinies)
905-	D0 21	BNE D928	branchement forcé en D928

Cherche la 1ère lettre du mot-clé Sédoric de n° d'ordre X

Ce s/p est complètement bogué, ce qui explique la mise à zéro hâtive de certains codes de la table C800: on y a supprimé tous les codes Sédoric!

907-	A5 F2	LDA F2	
909-	48	PHA	sauve F2 sur la pile (longueur chaîne LINPUT)
90A-	86 F2	STX F2	met X dans F2 (n° d'ordre du mot-clé Sédoric)
90C-	A2 3F	LDX #3F	code ASCII de "?" (caractère précédant "@") C'est là que commence la série des bogues, LDX #40 aurait été préférable!
90E-	A0 00	LDY #00	index pour lire dans sous-table des mots-clés
910-	B9 BD CB	LDA CBBD,Y	lit un numéro d'ordre des mots-clés Sédoric... et ça continue: CBC1 aurait été mieux, quoique de toute façon...
913-	E8	INX	code ASCII suivant ("@", "A", "B" etc...)
914-	C8	INY	
915-	C8	INY	Y = Y + 4
916-	C8	INY	Y vise le numéro d'ordre suivant
917-	C8	INY	
918-	C5 F2	CMP F2	n° d'ordre lu < n° d'ordre cherché?
91A-	90 F4	BCC D910	si oui, pas dépassé, on reboucle en D910 L'organigramme est mal pensé et irrécupérable... vivement la version 3 de Sédoric!
91C-	8E 4B C0	STX C04B	sinon, trouvé ou dépassé, sauve code ASCII
91F-	A6 F2	LDX F2	récupère X (n° d'ordre du mot-clé Sédoric)
921-	68	PLA	
922-	85 F2	STA F2	et récupère F2 (longueur chaîne LINPUT)
924-	A9 DD	LDA #DD	AY = C9DD (table des mots-clés Sédoric)
926-	A0 C9	LDY #C9	pour recherche suite chaîne (terminée par 0)

Cherche la fin de la Xème chaîne dans tableau AY

Exemple: si le n° d'ordre de la chaîne à chercher est X = 2, la 1ère chaîne ayant le n° 0, il faut rechercher la fin de la 2ème chaîne pour être au début de la 3ème qui est donc la bonne (n°2).

928-	85 16	STA 16	adresse du tableau qu'il faudra explorer
92A-	84 17	STY 17	(C0E9/ROM, C9DD/RAMOV ou C87F/RAMOV)

```
D92C- CA          DEX        X = n° d'ordre de la chaîne dans le tableau
D92D- 30 07       BMI D936   continue en D936 lorsque X chaînes parcourues
D92F- 20 1C D8    JSR D81C   lecture d'un octet selon l'adresse en 16/17
D932- 10 FB       BPL D92F   reboucle en D92F tant que b7 de cet octet à 0
D934- 30 F6       BMI D92C   reboucle en D92C lorsque b7 à 1 (fin de chaîne)
```

Recherche le 1er caractère significatif de la chaîne de n° d'ordre X

```
D936- 20 1C D8    JSR D81C   la chaîne n° X est trouvée, lit l'octet selon la valeur actuelle de l'adresse en 16/17
D939- C9 20       CMP #20    est-ce un espace? (les chaînes sont justifiées)
D93B- F0 F9       BEQ D936   si oui, reboucle en D936 pour lire le suivant
D93D- A5 16       LDA 16
D93F- D0 02       BNE D943   1er caractère significatif trouvé, décrémente
D941- C6 17       DEC 17     l'adresse en 16/17 pour pointer sur lui
D943- C6 16       DEC 16
D945- AD 4B C0    LDA C04B   A = 0 (cas général) ou A = code ASCII de la 1ère lettre (cas d'un mot-clé Sédoric)
D948- 38          SEC        force à 1 le b7 de C049, c'est à dire du
D949- 6E 49 C0    ROR C049   flag "code de fontion en cours"
D94C- AE 47 C0    LDX C047   récupère X = nombre de caractères dans buffer
D94F- 4C 6F D8    JMP D86F   et termine en D86F
```

Effacer la mémoire tampon: s/p DEL TAMPON

```
D952- A9 7F       LDA #7F    A = carré de couleur INK utilisé par DEL
D954- 2C 4A C0    BIT C04A   N selon b7 de C04A (point d'entrée s/p D843/45)
D957- 30 F3       BMI D94C   si D843 (LINPUT) continue en D94C avec A = #7F
D959- AE 47 C0    LDX C047   si D845 (ROM etc) récupère X = nombre de caractères
D95C- F0 EE       BEQ D94C   si buffer vide continue en D94C avec A = #7F
D95E- CA          DEX        sinon décrémente le nombre de caractère X
D95F- A9 08       LDA #08    et remplace par A = #08 (flèche gauche)
D961- D0 EC       BNE D94F   branchement forcé en D86F (sortie) avec A = #08
```

Numérotation automatique des lignes de programme: s/p NUM AUTO

```
D963- AC 42 C0    LDY C042
D966- AD 43 C0    LDA C043   YA contient le n° de ligne BASIC
D969- 20 CA D2    JSR D2CA   JSR DF40/ROM transfère un nombre non signé YA dans ACC1 (floating point accumulator)
D96C- 20 D2 D2    JSR D2D2   E0D5/ROM convertit ACC1 en chaîne décimale AY située à partir de #0100 (qui contient le signe, ici un espace) et terminée par #00
D96F- A2 00       LDX #00
D971- 86 17       STX 17     #00FF -> 16/17 qui pointe donc
D973- CA          DEX        juste avant le début de la chaîne
D974- 86 16       STX 16
D976- E8          INX        index X = 0 au premier tour puis augmente
D977- BD 01 01    LDA 0101,X recherche le 0 qui marque la fin de la chaîne
D97A- D0 FA       BNE D976   reboucle en D976 tant que pas trouvé ce 0
D97C- 9D 02 01    STA 0102,X si trouvé, le déplace d'un cran
D97F- 8A          TXA        (prévoit un allongement de la chaîne)
D980- 48          PHA        puis empile la nouvelle position de ce 0
D981- AD 42 C0    LDA C042
D984- AC 43 C0    LDY C043   copie le n° de ligne de C042/C043 en 33/34
D987- 85 33       STA 33     (tampon pour nombre sur deux octets)
D989- 84 34       STY 34
D98B- 20 9C D1    JSR D19C   JSR C6B3/ROM "Recherche d'une ligne BASIC", si pas trouvée C = 0 et CE/CF pointe sur la ligne suivante ou sur la fin du programme, si trouvée C = 1 et CE/CF pointe sur l'octet de début de ligne
D98E- 68          PLA        récupère position du 0
D98F- AA          TAX        et la remet dans X
D990- A9 20       LDA #20    A = #20 (valeur par défaut si pas trouvée)
D992- 90 02       BCC D996   saute ligne suivante si pas trouvé ligne BASIC
D994- A9 2A       LDA #2A    A = #2A (valeur si ligne BASIC trouvée)
D996- 9D 01 01    STA 0101,X place A sur la pile à l'ancienne position du 0
D999- 18          CLC        prépare une addition
D99A- AD 44 C0    LDA C044
D99D- 6D 42 C0    ADC C042   C042/C043 = dernier n° de ligne
D9A0- 8D 42 C0    STA C042   C044/C045 = "pas" de numérotation
D9A3- AD 45 C0    LDA C045   mise à jour du n° de ligne BASIC en C042/C043
D9A6- 6D 43 C0    ADC C043   [42/43] = [42/43] + [44/45]
D9A9- 8D 43 C0    STA C043
D9AC- A9 0D       LDA #0D    A = #0D
D9AE- D0 98       BNE D948   et branchement forcé vers D948
```

SUITE DES COMMANDES SÉDORIC

EXECUTION 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 Sédoric (16 caractères alphanumériques au maximum) à un code de fonctions redéfinissables (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.

Analyse de la syntaxe et saisie des paramètres

9B0-	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)
9B3-	E0 10	CPX #10	X est-il >= #10?
9B5-	B0 3E	BCS D9F5	si oui, continue en D9F5
9B7-	8A	TXA	sinon, c'est une commande redéfinissable (code de
9B8-	0A	ASL	#00 à #0F) qui passe dans A et dont les 4 bits
9B9-	0A	ASL	faibles (b0 à b3) sont "shiftés" dans les
9BA-	0A	ASL	4 bits forts pour obtenir un index permettant
9BB-	0A	ASL	d'accéder aux différentes entrées
9BC-	48	PHA	de la table C880 enfin le résultat est empilé
9BD-	20 2C D2	JSR D22C	D067/ROM exige une "," place TXTPTR sur l'octet suivant
9C0-	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
9C3-	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
9C6-	C9 11	CMP #11	la longueur est-elle >= #11 (17)
9C8-	B0 2E	BCS D9F8	si oui, continue en D9F8 (erreur car 16 maxi)
9CA-	A8	TAY	teste encore la longueur
9CB-	F0 2B	BEQ D9F8	la longueur est-elle nulle (erreur car 1 mini)
9CD-	68	PLA	récupère le n° de code "shifté" et le passe
9CE-	AA	TAX	dans X (index d'entrée dans la table C880)
9CF-	A9 10	LDA #10	
9D1-	85 F2	STA F2	F2 = #10 pour copier 16 caractères
9D3-	A9 20	LDA #20	Efface la commande actuelle en remplissant
9D5-	9D 80 C8	STA C880,X	la chaîne corresp avec des espaces
9D8-	E8	INX	emplacement suivant dans la chaîne
9D9-	C6 F2	DEC F2	décrémente le nombre de copies à faire
9DB-	D0 F8	BNE D9D5	et reboucle en D9D5 tant qu'il en reste
9DD-	88	DEY	Y = longueur de la chaîne - 1
9DE-	CA	DEX	X = pointeur dans chaîne ajust sur dernière pos
9DF-	B1 91	LDA (91),Y	lit le dernier caractère de la chaîne saisie
9E1-	09 80	ORA #80	force b7 à 1 (marque le dernier caractère) et
9E3-	9D 80 C8	STA C880,X	le met en place en dernière position dans table
9E6-	CA	DEX	position précédente dans la table C880
9E7-	88	DEY	nombre de caractères restant à copier
9E8-	30 35	BMI DA1F	simple RTS s'il n'en reste plus
9EA-	B1 91	LDA (91),Y	lit un caractère de la chaîne (par la fin)
9EC-	F0 07	BEQ D9F5	caractère nul interdit, branche en D9F5 (erreur)
9EE-	30 05	BMI D9F5	caractère >= #80 (token) interdit, branche idem. Quel dommage d'interdire les token, 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! Vive la version 3 de Sédoric!
9F0-	9D 80 C8	STA C880,X	et le met en place dans la chaîne de la table
9F3-	90 F1	BCC D9E6	rebouclage forcé en D9E6 (C mis à 0 en D9C6)
9F5-	A2 08	LDX #08	pour "ILLEGAL QUANTITY ERROR"
9F7-	2C A2 12	BIT 12A2	et continue en D9FA
9F8-	A2 12	LDX #12	pour "STRING TOO LONG ERROR"
9FA-	4C 7E D6	JMP D67E	incrémente X et traite l'erreur n° X

EXECUTION 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) à la ou les touches qui seront pressée immédiatement après. Si une seule touche est pressée, la fonction corresp sera accessible à l'aide de la combinaison FUNCT+touche. Si les touches SHIFT+touche sont pressées, la fonction corresp 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 Sédoric (voir 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.

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 corresp, 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.

EXECUTION 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 corresp 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 Sédoric: 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 (code de la touche selon l'annexe 7)
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 corresp 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

A45- 20 EB D1 JSR D1EB JSR CA73/ROM exécute la commande "IF"
A48- 4E FC 04 LSR 04FC force à 0 le b7 de 04FC (flag "IF")
A4B- 60 RTS

AUTRE SÉRIE DE ROUTINES GÉNÉRALES SÉDORIC

XPMAP prend le secteur bitmap, vérifie le format

A4C- A9 14 LDA #14 piste 20
A4E- A0 02 LDY #02 secteur 2
A50- 20 60 DA JSR DA60 bitmap chargée dans BUF2
A53- AE 00 C2 LDX C200 teste le 1er octet (n°0) (#FF en principe)
A56- E8 INX qui doit passer à 1 (donc Z = 1)
A57- F0 74 BEQ DACD si oui, OK branche sur un simple RTS
A59- A2 0A LDX #0A sinon, prépare une "UNKNOW'N FORMAT ERROR"
A5B- D0 22 BNE DA7F et la traite en D67E

Version 2.0 GB: XPMAP prend le secteur bitmap, vérifie le format

Quatre octets diffèrent entre les versions 1.006 et 2.0 GB

A4C- **20 43 FF** JSR FF43 nouveau s/p qui ajoute à l'initialisation A = piste n°20 et Y = secteur n°2, la **mise à zéro du b7 de 2F** (flag pour 1ère bitmap)
A4F- **EA** NOP un petit repos ne fait pas de mal
A50- 20 60 DA JSR DA60 bitmap chargée dans BUF2 (2ème secteur de piste 20)
A53- AE 00 C2 LDX C200 teste le 1er octet (n°#00) (vaut #FF en principe)
A56- E8 INX qui doit passer à 1
A57- F0 74 BEQ DACD si oui, OK branche sur un simple RTS
A59- A2 0A LDX #0A sinon, prépare une "UNKNOW'N FORMAT ERROR"
A5B- D0 22 BNE DA7F et la traite en D67E

XPBUF1 charge dans BUF1 le secteur Y de la piste A

A5D- A2 C1 LDX #C1 X = HH de BUF1
A5F- 2C A2 C2 BIT C2A2 et continue en DA65

XPBUF2 charge dans BUF2 le secteur Y de la piste A

A60- A2 C2 LDX #C2 X = HH de BUF2
A62- 2C A2 C3 BIT C3A2 et continue en DA65

XPBUF3 charge dans BUF3 le secteur Y de la piste A

A63- A2 C3 LDX #C3 X = HH de BUF3

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

A65- 8E 04 C0 STX C004 HH de l'adresse de chargement d'un secteur
A68- A2 00 LDX #00 prépare LL de RWBUF
A6A- 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

A6D- 8D 01 C0 STA C001 piste A -> PISTE
A70- 8C 02 C0 STY C002 secteur Y -> SECTEUR

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

A73- A2 88 LDX #88 paramètre lecture pour XRWTS (gestion des drives)
A75- 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 ecriture)
A78- F0 53 BEQ DACD si pas d'erreur branche en DACD (simple RTS)
A7A- A2 03 LDX #03 prépare "DISK I/O ERROR"
A7C- 50 01 BVC DA7F saute si ce n'était pas une disquette protégée
A7E- E8 INX sinon prépare "WRITE PROTECTED ERROR"
A7F- 4C 7E D6 JMP D67E et traite l'erreur n° 4 ou 5

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

A82- AD 25 C0 LDA C025 POSNMP piste du nom cherché dans catalogue
A85- AC 26 C0 LDY C026 POSNMS seteur du nom cherché dans catalogue
A88- D0 0A BNE DA94 branchement forcé: n° de secteur jamais nul

XSMAP sauve le secteur de bitmap sur la disquette

A8A- A9 14 LDA #14 piste 20
A8C- A0 02 LDY #02 secteur 2
A8E- A2 C2 LDX #C2 HH de BUF2 (secteur de bitmap)
A90- 2C A2 C1 BIT C1A2 continue en #DA96

Version 2.0 GB: XSMAP sauve le secteur de bitmap sur la disquette

Trois octets diffèrent entre les versions 1.006 et 2.0 GB

DA8A- **4C 80 DC** JMP DC80 nouvelle routine qui teste le b7 de 2F, s'il est nul, sauve seulement le 1er secteur de bitmap et s'il est à 1, sauve d'abord le 2ème, puis le 1er secteurs de bitmap. Voici cette nouvelle routine:

DC80- **24 2F** BIT 2F teste le b7 de 2F (flag 1 ou 2 secteurs de bitmap)
DC82- **10 05** BPL DC89 continue en DC89 si bitmap normale
DC84- **08** PHP sinon, sauve les registres
DC85- **20 4F FF** JSR FF4F empile les octets C202 à C208, sauve BUF2 sur la disquette dans le 3ème secteur de la piste 20, charge le 2ème secteur de la piste 20, restaure les octets C202 à C208 et enfin force C = 1
DC88- **28** PLP récupère les registres sauvés ci-dessus
DC89- **A0 02** LDY #02 2ème secteur
DC8B- **A9 14** LDA #14 de la piste n°20
DC8D- **4C 8E DA** JMP DA8E reprend le cours normal de XSMAP en DA8E

DA8D- 02 ?? résidu: un NOP eût été beaucoup plus propre
DA8E- A2 C2 LDX #C2 sauve BUF2 dans le Yème secteur de la piste n°20
DA90- 2C A2 C1 BIT C1A2 continue en #DA96

XSBUF1 sauve BUF1 à la piste A et le secteur Y
DA91- A2 C1 LDX #C1 HH de BUF1
DA93- 2C A2 C3 BIT C3A2 continue en #DA96

XSBUF3 sauve BUF3 à la piste A et le secteur Y
DA94- A2 C3 LDX #C3 HH de BUF3

Initialise RWBUF selon HH=X et LL=00
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 le secteur visé par RWBUF à piste A secteur Y
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 de 0 le BUFFER1
DACE- A9 C1 LDA #C1 A = HH de BUF1
DAD0- 2C A9 C2 BIT C2A9 continue en DAD6

XVBUF2 Rempli de 0 le BUFFER2
DAD1- A9 C2 LDA #C2 A = HH de BUF2
DAD3- 2C A9 C3 BIT C3A9 continue en DAD6

XVBUF3 Rempli de 0 le BUFFER3
DAD4- A9 C3 LDA #C3 A = HH de BUF3

Rempli de 0 une page mémoire à partir de HH = A et LL=#00

AD6- 85 0F STA 0F
AD8- A9 00 LDA #00 adresse du buffer à vider -> 0E/0F
ADA- 85 0E STA 0E
ADC- A0 00 LDY #00 remet à zéro index Y
ADE- 98 TYA remet à zéro X
ADF- 91 0E STA (0E),Y copie X à l'adresse pointée
AE1- C8 INY vise adresse suivante
AE2- D0 FB BNE DADF et reboucle tant que pas nul
AE4- 60 RTS (vide de LL=#00 à LL=#FF soit 256 octets)

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

AE5- AD 25 C0 LDA C025 piste (POSNMP)
AE8- AC 26 C0 LDY C026 secteur (POSNMS)
AEB- 20 63 DA JSR DA63 XPBUF3 charge dans BUF3 le secteur Y de la piste A

XBUCA transfère BUFNOM dans BUF3, à la position POSNMX

AEE- AE 27 C0 LDX C027 position du nom de fichier dans le secteur de catalogue
AF1- A0 F0 LDY #F0 artifice génial! lit dans BUFNOM à partir de
AF3- B9 39 BF LDA BF39,Y BF39 + F0 = C029 = 1er caractère du nom dans BUFNOM
AF6- 9D 00 C3 STA C300,X et écrit à partir de POSNMX dans BUF3
AF9- E8 INX index écriture caractère suivant
AFA- C8 INY index lecture caractère suivant
AFB- D0 F6 BNE DAF3 jusqu'à ce que Y atteigne #00 soit 16 octets
AFD- 60 RTS (de #F0 à #(1)00)

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

AFE- AD 25 C0 LDA C025 piste (POSNMP)
B01- AC 26 C0 LDY C026 secteur (POSNMS)
B04- 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

B07- AE 27 C0 LDX C027 position du nom de fichier dans le secteur de catalogue
B0A- A0 F0 LDY #F0 artifice génial! (si, si, si!) (voir ci-dessus en DAF3)
B0C- BD 00 C3 LDA C300,X lit à partir de POSNMX dans BUF3 puis
B0F- 99 39 BF STA BF39,Y écrit dans BUFNOM à partir de C029
B12- E8 INX index lecture caractère suivant
B13- C8 INY index écriture caractère suivant
B14- D0 F6 BNE DB0C jusqu'à ce que Y atteigne #00 soit 16 octets
B16- 60 RTS (de #F0 à #(1)00)

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

B17- A0 F4 LDY #F4 lit dans BUFNOM à partir de
B19- B9 35 BF LDA BF35,Y BF35 + F4 = C029 = 1er caractère du nom dans BUFNOM
B1C- C9 3F CMP #3F est-ce un "?"? (joker)
B1E- F0 05 BEQ DB25 si oui, branche en DB25 (saute la comparaison)
B20- DD 00 C3 CMP C300,X sinon, compare avec le caractère corresp du catalogue
B23- D0 1C BNE DB41 si différent, branche en DB41 (ajuste POSNMX sur l'"entrée" suivante et reprend la comparaison en DB17
B25- E8 INX si identique, incrémente
B26- C8 INY les index X et Y (vise caractère suivant)
B27- D0 F0 BNE DB19 reboucle jusqu'à Y = #00 (soit 12 caractères)
B29- AE 27 C0 LDX C027 nom identique: remet X = POSNMX
B2C- 60 RTS (X entrée nom à comparer = X sortie nom trouvé)

**Vérifie que la disquette en place est bien une disquette Sédoric,
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**

B2D- 20 4C DA JSR DA4C XPMAP lit secteur de bitmap, vérifie le format

**XTVNM cherche le fichier indiqué dans BUFNOM,
retourne X = POSNMX POSNMP et POSNMS ou Z = 1**

B30- A9 14 LDA #14 piste 20
B32- A0 04 LDY #04 secteur 4 = 1er secteur de catalogue
B34- 8D 25 C0 STA C025 POSNMP piste du nom cherché dans le catalogue
B37- 8C 26 C0 STY C026 POSNMS secteur du nom cherché dans le catalogue
B3A- 20 63 DA JSR DA63 XPBUF3 charge dans BUF3 le secteur Y de la piste A
B3D- A2 10 LDX #10 pointe sur la 1ère entrée (1er nom de fichier)

DB3F- D0 07 BNE DB48 branchement forcé en DB48: mise à jour de POSNMX

Ajuste X = POSNMX sur l'"entrée" suivante et reprend la comparaison

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 maxi = (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 place dans DIR -> POSNMX, POSNMP et POSNMS

DB59- 20 A5 DB JSR DBA5 cherche POSNMX 1ère 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 dèjà 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 XSMAP sauve BUF2 (bitmap) sur la disquette
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 1ère 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 (1ère 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 1ère 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 (1er 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 maxi c'est la position de la 1ère 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)

Ecriture du ou des descripteurs du fichier à sauver
Reviens avec le nombre de secteurs à sauver dans NSSAV (C05A/5B), les coordonnées du 1er secteur descripteur dans

PSDESC (C05C/5D), le nombre de descripteurs utilisés dans NSDESC (C05E) et 1er descripteur en place.

BC0- 8D 58 C0 STA C058 AY -> NSRSAV
BC3- 8C 59 C0 STY C059 (nombre de secteurs restant à sauver)
BC6- 8D 5A C0 STA C05A AY -> NSSAV
BC9- 8C 5B C0 STY C05B (nombre de secteurs à sauver)

Ecriture du descripteur n°1

BCC- 20 CE DA JSR DACE XVBUF1 rempli BUF1 de 0 (futur descripteur)
BCF- A2 01 LDX #01
BD1- 8E 5E C0 STX C05E 1 -> NSDESC (nombre de descripteurs utilisés)
BD4- 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")
BD7- 8D 5C C0 STA C05C AY -> PSDESC (coordonnées 1er secteur descript)
BDA- 8C 5D C0 STY C05D (n° piste -> C05C et n° secteur -> C05D)
BDD- 8D 01 C0 STA C001 n° piste -> PISTE
BE0- 8C 02 C0 STY C002 n° secteur -> SECTEUR
BE3- A2 08 LDX #08
BE5- BD 51 C0 LDA C051,X copie 9 octets de C051 à C059 en C103 à C10B
BE8- 9D 03 C1 STA C103,X soit FTYPE, DESALO, FISALO, EXSALO et NSRSAV
BEB- CA DEX (c'est à dire les 9 octets de STATUS)
BEC- 10 F7 BPL DBE5
BEE- 8E 02 C1 STX C102 soit #FF -> C102 marque du 1er secteur descripteur
BF1- 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

BF3- 8E 5F C0 STX C05F X -> C05F (PTDESC = pointeur descripteur)
BF6- AD 58 C0 LDA C058 teste NSRSAV
BF9- 0D 59 C0 ORA C059 reste t-il des secteurs à sauver?
BFC- F0 58 BEQ DC56 sinon branche en DC56 (fini)
BFE- AD 58 C0 LDA C058
C01- D0 03 BNE DC06
C03- CE 59 C0 DEC C059 décrémente le nombre de secteurs à sauver
C06- CE 58 C0 DEC C058
C09- 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")
C0C- AE 5F C0 LDX C05F X pointe dans la liste des coordonnées du descripteur
C0F- 9D 00 C1 STA C100,X écrit dans la liste des coordonées du
C12- E8 INX descripteur une paire piste/secteur
C13- 98 TYA pointant sur chacun des secteurs
C14- 9D 00 C1 STA C100,X constituants le fichier
C17- E8 INX vise octet suivant du descripteur
C18- D0 D9 BNE DBF3 reboucle si fin du descripteur pas atteinte
C1A- AD 58 C0 LDA C058 si fin du descripteur atteinte,
C1D- 0D 59 C0 ORA C059 teste s'il reste des secteurs à sauver
C20- F0 34 BEQ DC56 s'il ne reste rien branche en DC56 (fini)
C22- AC 01 C1 LDY C101 s'il en reste, teste le n° de secteur du descripteur suivant
C25- D0 1C BNE DC43 si déjà validé, branche en DC43
C27- 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")
C2A- 8D 00 C1 STA C100 écrit le n° de piste en C100
C2D- 48 PHA et l'empile
C2E- 8C 01 C1 STY C101 écrit le n° de secteur en C101
C31- 98 TYA et l'empile
C32- 48 PHA
C33- 20 A8 DA JSR DAA8 sauve BUF1 selon DRIVE, PISTE et SECTEUR
C36- 68 PLA
C37- 8D 02 C0 STA C002 récupère dans PISTE et SECTEUR les
C3A- 68 PLA coordonnées du descripteur suivant
C3B- 8D 01 C0 STA C001
C3E- EE 5E C0 INC C05E augmente NSDESC (nombre de descripteurs utilisés)
C41- D0 0C BNE DC4F branchement forcé en DC4F

Sauve le descripteur courant et charge le suivant

C43- 20 A8 DA JSR DAA8 sauve BUF1 selon DRIVE, PISTE et SECTEUR
C46- AD 00 C1 LDA C100
C49- AC 01 C1 LDY C101 AY coordonnées du descripteur suivant
C4C- 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 1er 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 A = n° de la piste et Y = n° du secteur libre trouvé. Sinon "DISK FULL ERROR"

Cette entrée ne semble 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 avec X = LL
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"

Cherche un secteur libre

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 1er octet représente l'état du secteur n°#01 de la piste n°#00. L'état du Nème 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. Ainsi sur une disquette formatée en 17 secteurs par piste, pour le 2ème 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 7ème de la 20ème piste). Or 347 / 8 = 43 (#2B) et on a r = 3. Le 2ème secteur de catalogue est donc représenté par le b2 (3ème bit de poids faible) de l'octet n°#10 + #2B = #3B.

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

DC7D- 8A TXA
DC7E- D0 03 BNE DC83 décrémente le nombre de secteurs libres
DC80- CE 03 C2 DEC C203 (octets n°#02/03 du secteur de bitmap)
DC83- CE 02 C2 DEC C202
DC86- A2 00 LDX #00 X pointe au début de la liste des secteurs
DC88- BD 10 C2 LDA C210,X teste l'octet visé par X dans liste des secteurs
DC8B- D0 03 BNE DC90 si présence de secteurs libres, branche en DC90
DC8D- E8 INX sinon, vise octet suivant
DC8E- D0 F8 BNE DC88 branchement forcé, reboucle en DC88
DC90- A9 01 LDA #01 0000 0001 masque pour 1er bit
DC92- A0 00 LDY #00 n° du bit éxaminé (1er = 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)

Version 2.0 GB: Cherche un secteur libre

La routine DC7D a été remplacée par la routine FF94 dont on pourra consulter le listing de désassemblage à la fin de cet ouvrage. Un nouveau sous-programme en DC80 a été mis en place dans les octets ainsi libérés. Ce sous-programme 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. L'ensemble de ces modifications affecte 19 octets.

C7D- **4C 94 FF** JMP FF94 nouvelle routine "Cherche un secteur libre"

Sauve la bitmap sur la disquette

C80- **24 2F** BIT 2F teste le b7 de 2F (flag 1ère ou 2ème page active)
C82- **10 05** BPL DC89 continue en DC98 si bitmap normale (1ère page active)
C84- **08** PHP sinon (2ème page active), sauve les registres
C85- **20 4F FF** JSR FF4F empile les octets C202 à C208, sauve BUF2 sur la disquette dans le 3ème secteur de la piste 20, charge le 2ème secteur de la piste 20, restaure les octets C202 à C208, force C = 1 et revient ici
C88- **28** PLP récupère les registres sauvés ci-dessus

Sauve la 1ère page de bitmap

C89- **A0 02** LDY #02 2ème secteur
C8B- **A9 14** LDA #14 de la piste n°20
C8D- **4C 8E DA** JMP DA8E reprend le cours normal de XSMAP: sauve BUF2 sur la disquette dans le Yème secteur de la piste n°20

La suite, c'est à dire la fin de l'ancien sous-programme "Chercher un secteur libre" en DC90 etc.. est inchangée et toujours utilisée par le nouveau sous-programme FF94 "Chercher un secteur libre" en FFBB et FFC3.

inverse le bit corresp et met à jour la bitmap

C9F- 68 PLA récupère le masque et en inverse tous les bits
CA0- 49 FF EOR #FF (secteur libre est pointé par le 0 du masque)
CA2- 3D 10 C2 AND C210,X met à zéro le bit du secteur à réserver
CA5- 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.

CA8- **A9** 00 LDA #00 (ou RTS, voir plus loin la modification V2.0 GB)
CAA- 85 F3 STA F3 F3 = #00 (sera le HH du nombre de secteurs)
CAC- 8A TXA nombre d'octets situés avant l'octet modifié
CAD- 0A ASL pour multiplier A par 8,
CAE- 26 F3 ROL F3 on pousse les 3 bits de poids fort de A,
CB0- 0A ASL dans les 3 bits de poids faible de F3,
CB1- 26 F3 ROL F3 qui à la fin contient le HH du résultat,
CB3- 0A ASL le LL étant dans A
CB4- 26 F3 ROL F3 F2/F3 contient le nombre N de secteurs
CB6- 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
CB8- 98 TYA récupère n° du bit modifié (codé par b0 b1 b2)
CB9- 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 (1er secteur = n°0).

Version 2.0 GB
La nouvelle routine "Cherche un secteur libre" implantée en FF94 place temporairement la valeur #60 (RTS) dans l'octet DCA8 au début du sous-programme "Calcule le nombre de secteurs du début de la disquette jusqu'au secteur trouvé". La valeur #A9 d'origine est ensuite restaurée.

Calcule le n° de piste A et le n° de secteur Y
CBB- 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.
CBD- 38 SEC prépare soustraction
CBE- E8 INX nombre de cycles de soustraction
CBF- A8 TAY LL du reste actuel
CC0- ED 07 C2 SBC C207 A = A - nombre de secteurs par piste
CC3- B0 F8 BCS DCBD reboucle tant que pas dépassé
CC5- C6 F3 DEC F3 si dépassé, décrémente aussi HH
CC7- 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  1ère face si A < nombre de pistes par face
DCCF- ED 06 C2    SBC C206  si 2ème 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 #10ème octet du secteur de bitmap) et avec dans A un masque dont un seul bit est à 1 représentant le secteur YA à libérer.

L'état du N^{ème} 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 2ème 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 7ème de la 20ème piste). Or 347 / 8 = 43 (#2B) et on a r = 3. Le 2ème secteur de catalogue est donc représenté par le b2 (3ème bit de poids faible) de l'octet n° X = #2B. 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
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 1ère face
DCDA- 29 7F       AND #7F   sinon, force à zéro le flag de 2ème face, c'est à dire A = n° de la piste dans la 2ème 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 1ère face + n° de la piste dans la 2ème face
DCE0- AA          TAX       X = nouveau n° de piste (sans saut à #80)
```

Nombre de secteurs dans les pistes précédantes

```
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

```
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. Le nouveau LL représente le résultat de la division, car compte tenu de la capacité maximale des disquettes Sédoric (1920 secteurs), le HH du résultat de la division est toujours nul. L'opération est donc très simple et pourtant le calcul qui suit est complètement faux!

Calcul du reste de la division

```
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 la position de l'octet à modifier dans la bitmap

```
D02-  68          PLA          récupère le LL du totalisateur dans A
D03-  46 F3       LSR F3
D05-  6A          ROR
D06-  46 F3       LSR F3       b0 b1 b2 du HH sont récupérés
D08-  6A          ROR          dans b5 b6 b7 du LL
D09-  46 F3       LSR F3
D0B-  6A          ROR
D0C-  AA          TAX          X vise l'octet de la bitmap qu'il faut modifier
```

Sédoric V2.0 GB

La fin du sous-programme "Calcule à quel octet et à quel bit de la bitmap correspond le secteur YA à libérer" a été détourné vers le sous-programme complémentaire FFD9 en modifiant les 3 octets DD0B/DD0D:

D0B- **4C D9 FF** JMP FFD9 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 2ème secteur de bitmap. Voici le sous- programme complémentaire:

```
FD9-  6A          ROR          remplace le ROR écrasé en DD0B
      A est l'octet de la bitmap qu'il faut modifier
FDA-  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
FDC-  D0 04       BNE FFE2     si c'est le cas, saute les 2 instructions suivantes
FDE-  C9 F0       CMP #F0      teste si A < #F0 c'est à dire si l'octet à modifier est sur le 1er secteur de bitmap
FE0-  90 0F       BCC FFF1     si oui, continue en FFF1
FE2-  24 2F       BIT 2F       teste si le b7 de 2F est à 1, c'est à dire si le 2ème secteur de bitmap est présent dans BUF2
FE4-  30 03       BMI FFE9     si oui, saute l'instruction suivante
FE6-  20 51 FF    JSR FF51     sauve le 1er en place dans BUF2 et charge le 2ème
FE9-  38          SEC          pour faire une soustraction
FEA-  E9 F0       SBC #F0      calcule l'octet du 2ème qui sera modifié
FEC-  AA          TAX          remplace le TAX écrasé en DD0C
      A est l'octet du 2ème secteur de bitmap qu'il faut modifier
FED-  38          SEC          remplace le SEC écrasé en DD0D
FEE-  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
FF1-  24 2F       BIT 2F       teste si le 1er secteur de bitmap est présent dans BUF2
FF3-  10 F7       BPL FFEC     si oui, reprend en FFEC
FF5-  20 4F FF    JSR FF4F     sinon, sauve le 2ème secteur de bitmap et charge le 1er
FF8-  B0 F2       BCS FFEC     et reprend en FFEC (C mis à 1 par s/p FF4F)
```

Elabore un masque dont un bit représente le secteur à libérer

```
D0D-  38          SEC          le bit significatif sera à 1
D0E-  A9 00       LDA #00      le masque est à zéro au départ
D10-  2A          ROL          effectue un nombre de rotation à gauche égal à
D11-  88          DEY          la retenue de la division, ce qui amène C à la
D12-  10 FC       BPL DD10     position qui représente le secteur à libérer
D14-  60          RTS
```

XDETSE Libère le secteur Y de la piste A sur le bitmap courant
Retourne avec C = 1 si ce secteur était déja libre sinon avec C = 0.

```
D15-  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
D18-  1D 10 C2    ORA C210,X   prend dans A l'octet situé à la Xème position après le début de la bitmap et force à 1 le bit
      corresp au masque
D1B-  DD 10 C2    CMP C210,X   teste si l'octet est inchangé
D1E-  F0 0C       BEQ DD2C     si oui, simple RTS en DD2C avec C = 1
D20-  9D 10 C2    STA C210,X   réécrit cet octet s'il a été modifié
D23-  EE 02 C2    INC C202     et incrémente le nombre d'octets libres
D26-  D0 04       BNE DD2C
D28-  EE 03 C2    INC C203
D2B-  18          CLC          C = 0 si libération effective
D2C-  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 déja occupé sinon avec C = 0.

```
D2D-  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
D30-  49 FF       EOR #FF      inverse tous les bits du masque A
D32-  3D 10 C2    AND C210,X   prend dans A l'octet situé à la Xème position après le début de la bitmap et force à 0 le bit
```

```
        corresp 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 SÉDORIC

(AVEC QUELQUES ROUTINES ASSOCIÉES, D'USAGE GÉNÉRAL)

EXECUTION COMMANDE SEDORIC SAVEM

Rappel de la syntaxe
SAVEM nom_de_fichier(,A début)(,E fin)(,T adresse_d'exécution)(,AUTO)

Sauve sur disquette le programme BASIC ou le bloc mémoire (si les paramètres ,A ,E et ,T sont indiqués) et ce en mode d'exécution au chargement si ,AUTO est précisé. Le fichier sauvegardé est ajouté à la suite du fichier existant.

Non documenté
Voir ci-sessous à la commande SAVE.

```
D4A- A9 40      LDA #40      code #40 pour SAVEM
D4C- 2C A9 C0   BIT C0A9     et continue en DD55...
```

EXECUTION COMMANDE SEDORIC SAVEU

Rappel de la syntaxe
SAVEU nom_de_fichier(,A début)(,E fin)(,T adresse_d'exécution)(,AUTO)

Sauve sur disquette le programme BASIC ou le bloc mémoire (si les paramètres ,A ,E et ,T sont indiqués) et ce en mode d'exécution au chargement si ,AUTO est précisé. Si un fichier de même nom existe déja, il sera conservé avec l'extension ".BAK".

Non documenté
Voir ci-sessous à la commande SAVE.

```
D4D- A9 C0      LDA #C0      CODE #C0 pour SAVEU
D4F- 2C A9 80   BIT 80A9     et continue en DD55...
```

EXECUTION COMMANDE SEDORIC SAVE

Rappel de la syntaxe
SAVE nom_de_fichier(,A début)(,E fin)(,T adresse_d'exécution)(,AUTO)

Sauve sur disquette le programme BASIC ou le bloc mémoire (si les paramètres ,A ,E et ,T sont indiqués) et ce en mode d'exécution au chargement si ,AUTO est précisé. Si un fichier de même nom existe déja, un message d'erreur sera généré.

Non documenté
Il y a incompatibilité entre les options ",T" (programme LM seulement) et ",AUTO" (programme BASIC seulement).
Enfin, toujours le même problème avec l'utilisation possible des minuscules pour entrer les commandes Sédoric: ça ne marche pas à tous les coups (bogue). Ici, le ",AUTO" doit être tapé en majuscules.

```
D50- A9 80      LDA #80      CODE #80 pour SAVE
D52- 2C A9 00   BIT 00A9     et continue en DD55...
```

EXECUTION COMMANDE SEDORIC SAVEO

Rappel de la syntaxe
SAVEO nom_de_fichier(,A début)(,E fin)(,T adresse_d'exécution)(,AUTO)

Sauve sur disquette le programme BASIC ou le bloc mémoire (si les paramètres ,A ,E et ,T sont indiqués) et ce en mode d'exécution au chargement si ,AUTO est précisé. Si un fichier de même nom existe déja, il sera écrasé.

Non documenté
Voir ci-sessous à la commande SAVE.

```
D53- A9 00      LDA #00      CODE #00 pour SAVEO
```

Entrée commune SAVEM, SAVEU, SAVE et SAVEO: analyse 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 recherche "?" dans BUFNOM revient avec C = 1 si pas trouvé ou "WILDCARD(S) NOT ALLOWED ERROR" si trouvé
DD5E- 20 9E D3 JSR D39E XCRGOT relit le caractère à TXTPTR et le convertit en MAJUSCULE
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

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 lit le caractère suivant à TXTPTR et le convertit en MAJUSCULE
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
DD82- A9 40 LDA #40
DD84- 8D 51 C0 STA C051 FTYPE = #40 (bloc de données)
DD87- D0 D5 BNE DD5E rebouclage forcé en DD5E
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 lit le caractère suivant à TXTPTR et le convertit en MAJUSCULE
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)
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 lit le caractère suivant à TXTPTR et le convertit en MAJUSCULE
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)
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 lit le caractère suivant à TXTPTR et le convertit en MAJUSCULE
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)

EXECUTION COMMANDE SEDORIC KEYSAVE

Rappel de la syntaxe
KEYSAVE nom_de_fichier

Cette commande, qui est équivalante à SAVE nom_de_fichier,A#C800,E#C97F permet de sauvegarder la table "KEYDEF" (#C800/#C87F) et celle des 16 commandes redéfinissables (#C880/#C97F), 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.

Saisie du paramètre et exécution

DDCD- 20 4F D4 JSR D44F XNF lit nom de fichier non-ambigu à TXTPTR et l'écrit dans BUFNOM

```
DD0- A9 00      LDA #00
DD2- A0 C8      LDY #C8
DD4- 8D 52 C0   STA C052    adresse C800 -> DESALO (début de fichier)
DD7- 8C 53 C0   STY C053
DDA- A9 7F      LDA #7F
DDC- A0 C9      LDY #C9     adresse C97F -> AY pour FISALO (fin de fichier)
DDE- D0 1E      BNE DDFE    branchement forcé en #DDFE car Y non nul
```

EXECUTION COMMANDE SEDORIC ESAVE

Rappel de la syntaxe
ESAVE nom_de_fichier

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

```
DE0- 20 4F D4   JSR D44F    XNF lit nom de fichier non-ambigu à TXTPTR et l'écrit dans BUFNOM
DE3- AD 1F 02   LDA 021F    0 si mode TEXT, 1 si mode HIRES
DE6- D0 08      BNE DDF0    branche si mode HIRES
DE8- A2 80      LDX #80
DEA- A0 BB      LDY #BB     valeurs pour adresse BB80 et BFDF (écran TEXT)
DEC- A9 DF      LDA #DF
DEE- D0 06      BNE DDF6    branchement forcé en #DDF6 car A non nul
DF0- A2 00      LDX #00
DF2- A0 A0      LDY #A0     valeurs pour adresse A000 et BF3F (écran HIRES)
DF4- A9 3F      LDA #3F
DF6- 8E 52 C0   STX C052    mise à jour de DESALO (début de fichier)
DF9- 8C 53 C0   STY C053
DFC- A0 BF      LDY #BF     valeur pour adresse BFDF (TEXT) ou BF3F (HIRES)
DFE- A2 40      LDX #40     X = # 40 pour FTYPE "bloc de données"
E00- 20 3B DE   JSR DE3B    AY -> FISALO, X -> FTYPE et #0000 -> EXSALO
E03- A9 C0      LDA #C0
E05- 8D 4D C0   STA C04D    #C0 -> VSALO0 (code de type SAVEU)
E08- 20 9E D7   JSR D79E    recherche "?" dans BUFNOM revient avec C = 1 si pas trouvé ou "WILDCARD(S) NOT
     ALLOWED ERROR" si trouvé
```

Suite commune pour SAVE*, KEYSAVE, ESAVE et CREATEW

```
E0B- 38         SEC         prépare une soustraction
E0C- AD 54 C0   LDA C054
E0F- ED 52 C0   SBC C052
E12- 8D 4F C0   STA C04F    calcule FISALO - DESALO (adresse fin - adresse début)
E15- AD 55 C0   LDA C055    et résultat dans LGSALO (longueur du fichier)
E18- ED 53 C0   SBC C053
E1B- 8D 50 C0   STA C050
E1E- B0 7C      BCS DE9C    si adresse fin > adresse début, continue au s/p XSAVEB (sauve fichier selon BUFNOM,
     VSALO0, VSALO1, DESALO, FISALO, EXSALO)
E20- A2 08      LDX #08     sinon donnera une "ILLEGAL QUANTITY ERROR"
E22- 2C A2 09   BIT 09A2    et continue en D67E
E23- A2 09      LDX #09     pour "SYNTAX ERROR"
E25- 4C 7E D6   JMP D67E    incrémente X et traite l'erreur n° X
```

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

```
E28- 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
E2B- A5 9A      LDA 9A
E2D- A4 9B      LDY 9B      pointeur "début de BASIC"
E2F- 8D 52 C0   STA C052    copié dans DESALO (début de fichier)
E32- 8C 53 C0   STY C053
E35- A5 9C      LDA 9C
E37- A4 9D      LDY 9D      pointeur "fin de BASIC/début des variables"
E39- A2 80      LDX #80     code pour fichier BASIC non AUTO
E3B- 8D 54 C0   STA C054    copié dans FISALO (fin de fichier)
E3E- 8C 55 C0   STY C055
E41- 8E 51 C0   STX C051    et dans FTYPE (type de fichier)
E44- A9 00      LDA #00
E46- 8D 56 C0   STA C056    mise à zéro de EXSALO (adresse d'exécution)
E49- 8D 57 C0   STA C057
E4C- 60         RTS
```

EXECUTION COMMANDE SEDORIC CREATEW

Rappel de la syntaxe
CREATEW nom_de_fichier

Permet de créer un masque de saisie de 25 lignes de 40 caractères utilisable avec la commande WINDOW. Cette commande, qui est inderdite 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) 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 si bien mode TEXT et passe en saisie
DE53-	20 02 D3	JSR D302	JSR EB78/ROM si touche frappée alors N = 1 et A = code ASCII corresp, 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
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)
DE60-	20 40 D7	JSR D740	si oui s/p "Curseur OFF"
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	s/p "CURSEUR ON"
DE79-	4C 53 DE	JMP DE53	reboucle en attente de touche
DE7C-	C9 17	CMP #17	est-ce CTRL/W? (pour champ de données)
DE7E-	D0 0E	BNE DE8E	sinon continue en DE8E (suite analyse syntaxe)
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
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 Sauv fichier (BUFNOM, VSALO0/1, DESALO, FISALO, EXSALO)
(suite 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 Sédoric, 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

s/p "il existe déjà un fichier de ce nom dans le catalogue"

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

EAF- 20 07 DB JSR DB07 XCABU copie la ligne d'"entrée" de catalogue à POSNMX (BUF3) dans BUFNOM (cette mise à jour inclu PSDESP coordonnées du descripteur principal et NSTOTP nombre de secteurs totaux + PROT)

EB2- 4C 1B DF JMP DF1B et continue en DF1B

Traitement des erreurs

EB5- A2 02 LDX #02 pour une "INVALID FILE NAME ERROR"

EB7- 2C A2 06 BIT 06A2 et continue en DEBA

suite pour SAVE (erreur)

EB8- A2 06 LDX #06 pour une "FILE ALREADY EXIST ERROR"

EBA- 4C 7E D6 JMP D67E incrémente X et traite l'erreur n° X

suite pour SAVEO (DEL ancien fichier)

EBD- 20 64 E2 JSR E264 DELète le fichier indexé à POSNMX dans BUF3

EC0- B0 2D BCS DEEF si C = 1 (nom + IS PROTECTED) simple CLI et RTS

EC2- 4C 11 DF JMP DF11 si C = 0 continue en DF11

suite pour SAVEU, KEYSAVE, ESAVE et CREATEW

EC5- A0 02 LDY #02

EC7- B9 32 C0 LDA C032,Y lit dans BUFNOM les 3 caractères de

ECA- 48 PHA l'extension et les empile (pour sauver

ECB- 88 DEY ultérieurement le nouveau fichier)

ECC- 10 F9 BPL DEC7

ECE- A0 02 LDY #02 index pour lecture 3 caractères

ED0- B9 32 C0 LDA C032,Y lit dans BUFNOM les caractères de l'extension et

ED3- D9 FA CC CMP CCFA,Y les compare avec BAK (table des extensions)

ED6- D0 05 BNE DEDD si différent, continue en DEDD

ED8- 88 DEY caractère suivant

ED9- 10 F5 BPL DED0 reboucle tant que 3 caractères n'ont pas été lus

EDB- 30 D8 BMI DEB5 si aucune différence trouvée, "INVALID FILE NAME ERROR" car on ne peut sauver avec SAVEU un fichier "*.BAK"

EDD- A2 03 LDX #03 index pour lecture dans la table des extensions

EDF- 20 4A D3 JSR D34A lit "BAK" dans table CCF7, le copie dans BUFNOM à la place de l'extension du fichier à sauver

EE2- 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

EE5- F0 0A BEQ DEF1 si pas trouvé, OK continue en DEF1

EE7- 20 64 E2 JSR E264 si trouvé, DELète ce "nom.BAK" (sera remplacé)

EEA- 90 05 BCC DEF1 si délétion réussie (C = 0), continue en DEF1

EEC- 68 PLA si le fichier était protégé

EED- 68 PLA dépile les 3 caractères de l'extension

EEE- 68 PLA devenus inutiles

EEF- 58 CLI autorise les interruptions et retourne

EF0- 60 RTS

Suite pour SAVEU, KEYSAVE, ESAVE et CREATEW
(OK pour renommer l'ancien fichier)

EF1- A0 00 LDY #00

EF3- 68 PLA

EF4- 99 32 C0 STA C032,Y récupère les 3 caractères de l'extension

EF7- C8 INY et les remet dans BUFNOM pour la recherche de

EF8- C0 03 CPY #03 l'ancien fichier et la sauvegarde du nouveau

EFA- D0 F7 BNE DEF3

EFC- 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

EFF- AE 27 C0 LDX C027 X = POSNMX début de l'"entrée" dans le secteur de catalogue

F02- B9 FA CC LDA CCFA,Y lit BAK dans la table des extensions

F05- 9D 09 C3 STA C309,X et le copie dans l'"entrée" du secteur de catalogue

F08- E8 INX caractère suivant dans "entrée" du secteur de catalogue

F09- C8 INY caractère suivant dans table des extensions

F0A- C0 03 CPY #03 et reboucle tant que 3 caractères

F0C- D0 F4 BNE DF02 n'ont pas été copiés

F0E- 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")

F11- A2 03 LDX #03

F13- 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 1er secteur descripteur dans PSDESC (C05C/5D), le nombre de descripteurs utilisés dans NSDESC (C05E) et 1er 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 NSTOT (mise à 0 des 4 bits poids fort)
DF96- 29 0F AND #0F +

```
F98-  6D 5B C0   ADC C05B   HH de NSSAV (+ retenue éventuelle de l'addition précédente)
F9B-  09 40      ORA #40    force b6 du résultat
F9D-  8D 38 C0   STA C038   -> HH de NSTOTP (inclus secteur précédent si SAVEM)
FA0-  AD 35 C0   LDA C035   AY coordonnées secteur descripteur principal
FA3-  AC 36 C0   LDY C036   (a été mis à jour seulement si SAVEM)
FA6-  F0 1D      BEQ DFC5   branche si pas valable (n° secteur jamais nul)
```

SAVEM: mise à jour du "lien"

```
FA8-  20 5D DA   JSR DA5D   XPBUF1 charge dans BUF1 secteur descrip principal
FAB-  AD 00 C1   LDA C100   lit AY coordonnées secteur descrip suivant
FAE-  AC 01 C1   LDY C101   (cherche le dernier secteur descripteur)
FB1-  D0 F5      BNE DFA8   reboucle si valable (n° secteur jamais nul)
FB3-  AD 5C C0   LDA C05C   si dernier descripteur trouvé (Y = #00)
FB6-  AC 5D C0   LDY C05D   lit et copie coordonnées du 1er secteur descripteur du fichier
FB9-  8D 00 C1   STA C100   sauvé dans le "lien" du dernier descripteur
FBC-  8C 01 C1   STY C101
FBF-  20 A4 DA   JSR DAA4   XSVSEC sauve descripteur selon DRIVE, PISTE, SECTEUR et RWBUF
FC2-  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

```
FC5-  AD 5C C0   LDA C05C
FC8-  AC 5D C0   LDY C05D   copie AY coordonnées 1er secteur descripteur
FCB-  8D 35 C0   STA C035   dans PSDESP coordonnées du secteur descripteur principal
FCE-  8C 36 C0   STY C036
FD1-  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

```
FD4-  20 8A DA   JSR DA8A   XSMAP sauve secteur de bitmap sur la disquette
FD7-  20 EE DA   JSR DAEE   XBUCA copie BUFNOM dans BUF3, à position POSNMX
FDA-  58         CLI        autorise les interruptions
FDB-  4C 82 DA   JMP DA82   sauve BUF3 (catalogue) selon POSNMP et POSNMS (RTS)
```

Vérifie mode TEXT

```
FDE-  AD 1F 02   LDA 021F   0 si mode TEXT, 1 si mode HIRES
FE1-  F0 10      BEQ DFF3   si mode TEXT, simple RTS
FE3-  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

```
FE6-  A9 00      LDA #00    pour remise à zéro
FE8-  A2 03      LDX #03    de 4 octets
FEA-  9D 4D C0   STA C04D,X remet à zéro VSALO0, VSALO1 et 2 octets suivants
FED-  CA         DEX        c'est à dire les adresse C04D, C04E, C04F et C050
FEE-  10 FA      BPL DFEA   et reboucle tant que X n'est pas négatif
FF0-  8E 72 C0   STX C072   #FF -> C072 (flag AUTO si b7=1, STOP si b7=0)
FF3-  60         RTS        et retourne

FF4-  4C 23 DE   JMP DE23   "SYNTAX ERROR"
```

EXECUTION COMMANDE SEDORIC LOAD

Rappel de la syntaxe
LOAD nom_de_fichier(,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 avec "")
DFF9- 20 54 D4 JSR D454 évalue le nom_de_fichier présent à TXTPTR, le met dans BUFNOM, après avoir convertit les "*" 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 recherche "?" dans BUFNOM revient avec C = 1 si pas trouvé ou "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 et le convertit en MAJUSCULE
E005- F0 4B BEQ E052 si fin de commande, charge le fichier
E007- D0 05 BNE E00E sinon, continue en E00E (cherche paramètres)
E009- 20 98 D3 JSR D398 XCRGET lit le caractère suivant à TXTPTR et le convertit en MAJUSCULE
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 (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 (pour 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 coexistance de ",V" et ",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 coexistance 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 saisit dans A caractère suivant
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)
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 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 paramètre ",V"
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

```
082-  20 FB D7   JSR D7FB     mise à jour de la variable ED (HH=Y et LL=A)
085-  AD 4D C0   LDA C04D     A = VSALO0 dont b6 = flag ",V" et b7 = flag ",N"
088-  0A          ASL          ancien b7 -> C et ancien b6 -> b7 et N
089-  30 50       BMI E0DB     si N = 1 (pour ",V"), branche (simple CLI et RTS)
08B-  2A          ROL          si N = 0, C -> b0 (ancien flag ",N")
08C-  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
08E-  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 demandé
091-  4A          LSR          b0 -> C (à 1 l'exécution du fichier sera lancée)
092-  AD 51 C0    LDA C051     A = FTYPE dont le b7 sert de flag "BASIC"
095-  10 0D       BPL E0A4     branche si pas "BASIC"
097-  08          PHP          si "BASIC", sauvegarde les indicateurs 6502
098-  20 B4 E0    JSR E0B4     restaure les liens de lignes et les pointeurs
09B-  28          PLP          récupère les indicateurs
09C-  90 03       BCC E0A1     saute la ligne suivante si C = 0 (pas de RUN)
09E-  4C AC D1    JMP D1AC     JSR C73A/ROM (place TXTPTR au début du BASIC et RUN)
0A1-  4C 80 D1    JMP D180     JSR C4A8/ROM (retour simple au Ready)
0A4-  90 35       BCC E0DB     autorise les interruptions et RTS si C = 0
0A6-  AD 56 C0    LDA C056     si C = 1 prend EXSALO (adresse exécution fichier)
0A9-  AC 57 C0    LDY C057     dans AY et exécute
0AC-  4C 6B 04    JMP 046B
```

EXECUTION 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.

```
0AF-  A0 01       LDY #01      Y = #01 pour indexer adresse suivant 0501 soit 0502
0B1-  98          TYA          A = #01 doit seulement être <> #00
0B2-  91 9A       STA (9A),Y   écrit #01 en 0502
0B4-  08          PHP          sauvegarde les indicateurs 6502
0B5-  78          SEI          interdit les interruptions
0B6-  20 88 D1    JSR D188     JSR C563/ROM restaure les liens des lignes
0B9-  A4 92       LDY 92
0BB-  A5 91       LDA 91       prend dans AY l'adresse de fin de BASIC -2
0BD-  18          CLC          prépare une addition
0BE-  69 02       ADC #02      ajoute 2 à LL
0C0-  90 01       BCC E0C3     répercute éventuellement
0C2-  C8          INY          la retenue sur HH
0C3-  85 9C       STA 9C
0C5-  84 9D       STY 9D       met à jour pointeur "fin BASIC/début variables"
0C7-  85 9E       STA 9E
0C9-  84 9F       STY 9F       met à jour pointeur "fin var/début tableaux"
0CB-  85 A0       STA A0
0CD-  84 A1       STY A1       met à jour pointeur "fin tab/début free RAM"
0CF-  A5 A6       LDA A6
0D1-  A4 A7       LDY A7       met à jour HIMEM "fin chaînes/début zone LM"
0D3-  85 A2       STA A2
0D5-  84 A3       STY A3       met à jour pointeur "fin free RAM/début chaînes"
0D7-  28          PLP          récupère les indicateurs
0D8-  4C CC D1    JMP D1CC     JSR C952/ROM commande "Restore" = mise à jour du pointeur DATA (B0/B1) sur le
début du texte BASIC
0DB-  58          CLI          autorise les interruptions
0DC-  60          RTS
0DD-  A2 00       LDX #00      pour "FILE NOT FOUND ERROR"
0DF-  2C A2 0C    BIT 0CA2     continue en D67E pour traiter l'erreur
0E0-  A2 0C       LDX #0C      pour "FILE TYPE MISMATCH ERROR"
0E2-  4C 7E D6    JMP D67E     continue en D67E pour traiter l'erreur
```

XLOADA charge programme selon BUFNOM, VSALO0, VSALO1, DESALO

```
0E5-  20 2D DB    JSR DB2D     vérifie que la disquette en place est bien une disquette Sédoric, 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 1ère 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 le(s) 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 1er 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 1er 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 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°00/01 = "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 maxi).

Pour les fichiers "joints", le "lien" du dernier descripteur de chaque fichier indique les coordonnées du 1er 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)
E10E- BD 01 C1 LDA C101,X vise 1er 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

```
125-  A5 9C     LDA 9C      A = LL du pointeur "fin du programme BASIC/début des variables"
127-  E9 02     SBC #02     calcule adresse "fin du programme BASIC" en diminuant LL du
129-  B0 09     BCS E134    pointeur de 2. Fini si C = 1 (pas de retenue),
12B-  88        DEY         mais répercussion sur HH si C = 0 (DEY ne touche
12C-  90 06     BCC E134    pas C) et branchement final forcé en E134
```

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

```
12E-  BD 02 C1  LDA C102,X  vise 2 et 3ème octets après X et prend dans AY
131-  BC 03 C1  LDY C103,X  l'adresse (normale) de début de fichier
134-  8D 52 C0  STA C052    AY = adresse où sera chargé le fichier
137-  8C 53 C0  STY C053    et l'écrit dans DESALO (nouveau) début
```

Entrée secondaire ",A adresse_de_chargement"

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 s/p E123) ou la valeur normale de début si ni ",A" ni ",J" (mise à jour par le s/p E12E).

```
13A-  38        SEC         prépare une soustraction
13B-  BD 04 C1  LDA C104,X  vise 2, 3, 4 et 5ème octets après X
13E-  FD 02 C1  SBC C102,X  calcule la longueur du fichier à charger
141-  8D 4F C0  STA C04F    (adresse de fin - adresse de début)
144-  BD 05 C1  LDA C105,X
147-  FD 03 C1  SBC C103,X  et l'écrit dans LGSALO (C04F/C050)
14A-  8D 50 C0  STA C050
14D-  18        CLC         prépare une addition
14E-  AD 52 C0  LDA C052    met LL de DESALO (adresse_de_chargement)
151-  8D 03 C0  STA C003    dans LL de RWBUF (adresse écriture du prochain secteur)
154-  6D 4F C0  ADC C04F    calcule A = LL de DESALO + LL de LGSALO
157-  48        PHA         et empile LL de (nouvelle) fin
158-  AD 53 C0  LDA C053    prend HH de DESALO (adresse de chargement),
15B-  A8        TAY         le décrémente et place le résultat dans HH de
15C-  88        DEY         RWBUF (donc RWBUF = adresse de chargement du fichier - #100)
15D-  8C 04 C0  STY C004    (sera incrémenté de #100 en début de boucle)
160-  6D 50 C0  ADC C050    A = HH DESALO + HH LGSALO = HH (nouvelle) fin
163-  A8        TAY         le copie dans Y. L'adresse de (nouvelle) fin, formée de LL/pile et HH dans Y servira pour
                            ",V"
```

AUTO/STOP

```
164-  2C 72 C0  BIT C072    teste le b7 de C072 (AUTO si b7=1 STOP si b7=0)
167-  10 0C     BPL E175    si STOP branche en E175 (EXSALO pas mis à jour)
169-  BD 06 C1  LDA C106,X  si AUTO, vise 6 et 7ème octets après X
16C-  8D 56 C0  STA C056    lit adresse d'exécution
16F-  BD 07 C1  LDA C107,X  dans secteur descripteur
172-  8D 57 C0  STA C057    et l'écrit dans EXSALO
```

F7/F8 nombre de secteurs à charger

```
175-  BD 08 C1  LDA C108,X  vise 8 et 9ème octets après X
178-  85 F7     STA F7      lit dans secteur descripteur
17A-  BD 09 C1  LDA C109,X  le nombre de secteurs à charger
17D-  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!

```
17F-  2C 4D C0  BIT C04D    teste b6 de VSALO0 (à 1 si ",V")
182-  50 35     BVC E1B9    branche en E1B9 si pas ",V"
184-  AD 53 C0  LDA C053    si ",V" lit HH de DESALO
187-  20 13 D6  JSR D613    XAFHEX affiche en hexadécimal le contenu de A
18A-  AD 52 C0  LDA C052    lit LL de DESALO
18D-  20 13 D6  JSR D613    XAFHEX affiche en hexadécimal le contenu de A
190-  20 28 D6  JSR D628    affiche un espace
193-  98        TYA         prend dans A le HH de fin de fichier
194-  20 13 D6  JSR D613    XAFHEX affiche en hexadécimal le contenu de A
197-  68        PLA         prend dans A le LL de fin de fichier
198-  20 13 D6  JSR D613    XAFHEX affiche en hexadécimal le contenu de A
19B-  20 28 D6  JSR D628    affiche un espace
19E-  A5 F9     LDA F9      prend dans A le type de fichier
1A0-  20 13 D6  JSR D613    XAFHEX affiche en hexadécimal le contenu de A
1A3-  20 28 D6  JSR D628    affiche un espace
1A6-  AD 57 C0  LDA C057    prend dans A le HH de l'adresse d'exécution
1A9-  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 8ème 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 8ème octet après #FF, puis visera le 10ème en E1BF et finalement le 12ème (1ère paire de coordonnées) au début du s/p 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    s/p 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 charg)
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 1er 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 coordonnées dernier secteur) sur pile
E1EA- 48         PHA         (pour exploration ultérieure 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.

```
218-  4E 72 C0   LSR C072     si AUTO, remet b7 à 0 (STOP) les fichiers
21B-  A9 00      LDA #00      secondaires seront d'office ni ",J" ni ",A":
21D-  8D 4E C0   STA C04E     remet VSALO1 à 0 (ni ",J" ni ",A")
220-  A5 F9      LDA F9       seul le FTYPE du 1er fichier comptera pour le
222-  8D 51 C0   STA C051     RUN: remet le type de fichier dans FTYPE
225-  4C FA E0   JMP E0FA     et continue en E0FA (cherche octet #FF)
```

s/p 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

```
228-  C8         INY          Y = Y + 2
229-  C8         INY          Y vise coordonnées secteur suivant à charger
22A-  D0 1D      BNE E249     si Y valable branche en E249
22C-  AD 03 C0   LDA C003     si Y = 0, (Y dépasse la fin du descripteur)
22F-  48         PHA          lit adresse en RWBUF (C003/C004) et l'empile
230-  AD 04 C0   LDA C004     (adresse où charger le secteur suivant du fichier)
233-  48         PHA
234-  AD 00 C1   LDA C100     A = piste
237-  AC 01 C1   LDY C101     Y = secteur du secteur descripteur suivant
23A-  F0 0F      BEQ E24B     si Y = 0 (impossible sauf si pas de suivant)
23C-  20 5D DA   JSR DA5D     XPBUF1 charge dans BUF1 le secteur descripteur suivant
23F-  68         PLA
240-  8D 04 C0   STA C004
243-  68         PLA          récupère valeur initiale de l'adresse RWBUF
244-  8D 03 C0   STA C003
247-  A0 02      LDY #02      Y = 2 (vise 1ère paire de coordonnées du secteur suivant)
249-  18         CLC          et C = 0 (descripteur en place et Y pointant
24A-  60         RTS          sur coordonnées du secteur suivant à charger)
```

"Il n'y a pas de descripteur suivant"

```
24B-  38         SEC          C = 1 (pas de descripteur suivant)
24C-  68         PLA          élimine 2 octets de la pile
24D-  68         PLA          (adresse qui était en RWBUF, devenue inutile)
24E-  58         CLI          autorise les interruptions
24F-  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")

```
250-  B9 00 C1   LDA C100,Y   lit dans le descripteur les coordonnées
253-  8D 01 C0   STA C001     piste/secteur du prochain secteur à charger
256-  B9 01 C1   LDA C101,Y   et les écrit respectivement dans PISTE
259-  8D 02 C0   STA C002     et dans SECTEUR
25C-  2C 4D C0   BIT C04D     teste b6 de VSALO (à 1 si ",V")
25F-  70 E9      BVS E24A     si ",V", branche en E24A (simple RTS)
261-  4C 73 DA   JMP DA73     XPRSEC lit un secteur selon DRIVE, PISTE, SECTEUR et RWBUF Le retour se fera au
```

point d'appel du s/p 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 s/p 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" corresp au fichier à supprimer dans le catalogue est effacée et par conséquent, les coordonnées piste/secteur du descripteur principal sont perdues. Avec un bon editeur 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 s/p 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 s/p (E4A7) et permet de réduire le nombre de secteurs de catalogue utilisés et, éventuellement, de récupérer des secteurs.

A l'entrée C = 0 si NF et C = 1 si NFA (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 NF

E264- 18 CLC entrée avec C = 0 (flag NF non ambigu)
E265- 24 38 BIT 38 continue en E267

XNOMDE DEL NFA

E266- 38 SEC entrée avec C = 1 (flag NFA) 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 (13ème et 14ème 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 1er 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° 1er 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 corresp 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 le 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,
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 NFA, 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 "IS PROTECTED"
E2D7- 20 6C D3 JSR D36C affiche X+1ème 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

2DC-	BD 08 C1	LDA C108,X	
2DF-	85 F5	STA F5	copie le nombre de secteurs de ce fichier
2E1-	BD 09 C1	LDA C109,X	(nombre d'octets à libérer) dans F5/F6
2E4-	85 F6	STA F6	
2E6-	8A	TXA	X = X + 10
2E7-	18	CLC	(pour passer les octets de STATUS et viser
2E8-	69 0A	ADC #0A	la 1ère paire piste/secteur de coordonnées
2EA-	AA	TAX	des secteurs constituant le fichier)

Libère les secteurs constituant le fichier

2EB-	8A	TXA	
2EC-	48	PHA	empile X
2ED-	BD 00 C1	LDA C100,X	lit dans AY les coordonnées d'un
2F0-	BC 01 C1	LDY C101,X	secteur du fichier
2F3-	20 15 DD	JSR DD15	XDETSE libère ce secteur sur le bitmap courant
2F6-	68	PLA	
2F7-	AA	TAX	récupère X et l'augmente de 2
2F8-	E8	INX	pour viser la paire suivante
2F9-	E8	INX	
2FA-	D0 16	BNE E312	branche si fin du descripteur pas atteinte

Charge et libère un descripteur secondaire

2FC-	AD 00 C1	LDA C100	si la fin du descripteur est atteinte, lit
2FF-	AC 01 C1	LDY C101	dans AY les coordonnées du descripteur suivant
302-	F0 C6	BEQ E2CA	s'il n'y en a plus, branche en E2CA
304-	20 5D DA	JSR DA5D	XPBUF1 charge dans BUF1 ce descripteur
307-	AD 01 C0	LDA C001	lit dans AY les coordonnées
30A-	AC 02 C0	LDY C002	du secteur chargé
30D-	20 15 DD	JSR DD15	XDETSE libère ce secteur sur le bitmap courant
310-	A2 02	LDX #02	prépare X pour viser 1ère paire piste/secteur de coordonnées des secteurs constituant le fichier

Pour chaque secteur libéré, décrémente le nombre restant

312-	A4 F5	LDY F5	
314-	D0 02	BNE E318	
316-	C6 F6	DEC F6	décrémente le nombre de secteurs à libérer
318-	C6 F5	DEC F5	
31A-	A5 F5	LDA F5	
31C-	05 F6	ORA F6	reste t-il des secteurs à libérer?
31E-	D0 CB	BNE E2EB	si oui, branche en E2EB pour libérer le suivant
320-	F0 96	BEQ E2B8	sinon, branche en E2B8 pour chercher #FF suivant

Affiche nom_de_fichier et taille du fichier à POSNMX

322-	20 B4 DA	JSR DAB4	affiche le nom de fichier présent à POSNMX dans BUF3
325-	A9 20	LDA #20	place un espace dans DEFAFF
327-	8D 4C C0	STA C04C	pour l'afficher devant la taille du fichier
32A-	AE 27 C0	LDX C027	X = POSNMX = début de l'"entrée" du catalogue
32D-	BD 0F C3	LDA C30F,X	lit dernier octet d'"entrée" (HH taille + PROT)
330-	08	PHP	sauvegarde les indicateurs dont N (à 1 si PROT)
331-	29 0F	AND #0F	0000 1111 efface les 4 bits de poids fort (PROT)
333-	A8	TAY	et sauve HH de la taille du fichier dans Y
334-	BD 0E C3	LDA C30E,X	lit octet précédent (LL de taille du fichier)
337-	20 56 D7	JSR D756	affichage en décimal du nombre AY
33A-	A9 20	LDA #20	pour afficher un espace si pas PROT
33C-	28	PLP	récupère les indicateurs dont N
33D-	10 02	BPL E341	saute l'instruction suivante si pas PROTégé
33F-	A9 50	LDA #50	pour afficher un "P"
341-	4C 2A D6	JMP D62A	XAFCAR affiche le caractère ASCII contenu dans A

EXECUTION COMMANDE SEDORIC DIR

Rappel de la syntaxe

La syntaxe de la commande DIR est très simple: DIR NFA (nom de fichier ambigu), le NFA 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:

```
rive A (Master) NNNNNNNNNNNNNNNNNNNN

ASTER   .SYS 103
```

```
1223 sectors free (D/42/17),  1 Files


Drive A (Slave ) NNNNNNNNNNNNNNNNNNNNN

SLAVE    .SYS  17

1395 sectors free (D/42/17),  1 Files


Drive A (Type=G) NNNNNNNNNNNNNNNNNNNNN

GAMES    .SYS  26

1377 sectors free (D/42/17),  1 Files
```

Dans ces 3 exemples le nom de la disquette est NNNNNNNNNNNNNNNNNNNNN (DNAME, 21 caractères maxi). Ces disquettes comportent un seul fichier qui contient la copie des secteurs systèmes d'une disquette MASTER, SLAVE ou GAMEINIT. Il y a respectivement 102, 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 102 secteurs systèmes, moins le fichier (102 secteurs plus un secteur descripteur) soit 1428 - 102 - 103 = 1223 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_ (soit 38 caractères à chaque fois)

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 dans BUF2 secteur bitmap, vérifie le format
E353-	A2 05	LDX #05	indexe "CR LF Drive"
E355-	20 6C D3	JSR D36C	affiche X+1ème 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 7ème message (" (Master)")
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 18ème message (" (Slave)")
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+1ème 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+1ème 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 entraine 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

392-	78	SEI	interdit les interruptions
393-	20 1F E4	JSR E41F	JSR conditionnel à CBF0/ROM (Aller à la ligne)
396-	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

399-	F0 27	BEQ E3C2	branche en E3C2 si fin de catalogue atteinte
39B-	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
39E-	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)
3A1-	F0 1C	BEQ E3BF	branche si la fin du catalogue est atteinte (CRLF pour finir la ligne commencée puis affichage ligne finale de bilan)
3A3-	20 28 D6	JSR D628	
3A6-	20 28 D6	JSR D628	affiche deux espaces, puis le nom_du_fichier
3A9-	20 22 E3	JSR E322	à POSNMX, sa taille et la lettre P s'il est protégé

Gestion pause de l'affichage

3AC-	58	CLI	autorise les interruptions
3AD-	20 02 D3	JSR D302	JSR EB78/ROM si touche frappée alors N = 1 et A = code ASCII corresp, sinon N = 0
3B0-	10 E0	BPL E392	si pas touche, reboucle (continue d'afficher)
3B2-	20 02 D3	JSR D302	si touche, pause l'affichage et re-teste touche
3B5-	10 FB	BPL E3B2	poursuit pause jusqu'à obtenir une 2ème touche
3B7-	C9 20	CMP #20	2ème touche obtenue, est-ce un espace?
3B9-	F0 D7	BEQ E392	si oui, reboucle reprend l'affichage
3BB-	C9 1B	CMP #1B	sinon, est-ce un ESC?
3BD-	D0 F3	BNE E3B2	si pas ESC, reprend scrutation clavier en E3B2
3BF-	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!

Affiche ligne de bilan de catalogue

3C2-	20 06 D2	JSR D206	CBF0/ROM va à la ligne
3C5-	A9 2A	LDA #2A	place un "*" dans DEFAFF pour afficher dans les
3C7-	8D 4C C0	STA C04C	digits libres devant le nombre de "sectors free"
3CA-	AD 02 C2	LDA C202	
3CD-	AC 03 C2	LDY C203	lit dans AY le nombre de secteurs libres
3D0-	20 56 D7	JSR D756	et l'affiche en décimal sur 4 digits
3D3-	A2 07	LDX #07	indexe " sectors free ("
3D5-	20 6C D3	JSR D36C	affiche X+1ème message de zone CEE7 terminé par "caractère + 128"
3D8-	A9 00	LDA #00	place un "#00" dans DEFAFF pour afficher
3DA-	8D 4C C0	STA C04C	
3DD-	A9 44	LDA #44	A = "D"
3DF-	2C 09 C2	BIT C209	teste b7 de C209 (à 1 si double face)
3E2-	30 02	BMI E3E6	si double face, saute l'instruction suivante
3E4-	A9 53	LDA #53	A = "S"
3E6-	20 2A D6	JSR D62A	XAFCAR affiche le caractère ASCII contenu dans A
3E9-	A9 2F	LDA #2F	A = "/"
3EB-	20 2A D6	JSR D62A	XAFCAR affiche le caractère ASCII contenu dans A
3EE-	AD 06 C2	LDA C206	A = nombre de pistes par face
3F1-	20 4E D7	JSR D74E	affiche en décimal le nombre A sur 2 digits
3F4-	A9 2F	LDA #2F	A = "/"
3F6-	20 2A D6	JSR D62A	XAFCAR affiche le caractère ASCII contenu dans A
3F9-	AD 07 C2	LDA C207	A = nombre de secteurs par piste
3FC-	20 4E D7	JSR D74E	affiche en décimal le nombre A sur 2 digits
3FF-	A9 29	LDA #29	A = ")"
401-	20 2A D6	JSR D62A	XAFCAR affiche le caractère ASCII contenu dans A
404-	A9 2C	LDA #2C	A = ","
406-	20 2A D6	JSR D62A	XAFCAR affiche le caractère ASCII contenu dans A
409-	A9 20	LDA #20	place un espace dans DEFAFF pour afficher
40B-	8D 4C C0	STA C04C	devant le nombre de fichiers
40E-	AD 04 C2	LDA C204	
411-	AC 05 C2	LDY C205	lit dans AY le nombre de fichiers
414-	A2 01	LDX #01	nombre de 0 à 999
416-	20 58 D7	JSR D758	affiche en décimal le nombre AY sur 3 digits
419-	28	PLP	récupère les indicateurs

```
E41A-  A2 08       LDX #08       indexe " Files "
E41C-  20 6C D3    JSR D36C      affiche X+1ème 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"
```

EXECUTION 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".

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
```

EXECUTION COMMANDE SEDORIC DESTROY

Rappel dela syntaxe
DESTROY (nom_de_fichier_ambigu)

Efface, sans demande de confirmation, tous les fichiers corresp 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
```

EXECUTION COMMANDE SEDORIC DEL

Rappel de la syntaxe
DEL (nom_de_fichier_ambigu)

Efface après demande de confirmation, tous les fichiers corresp 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 corresp sera détruit sans demande de confirmation. Les fichiers protégés par PROT ne peuvent être effacés.

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)
```

44F- 20 2D DB JSR DB2D vérifie que la disquette en place est bien une disquette Sédoric, 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
452- D0 03 BNE E457 si trouvé, saute l'instruction suivante
454- 4C DD E0 JMP E0DD si pas trouvé, "FILE NOT FOUND ERROR"
457- 20 A0 D7 JSR D7A0 cherche "?" dans BUFNOM (C = 1 si pas trouvé)
45A- 90 17 BCC E473 branche en E473 (il y a des "?" dans BUFNOM)
45C- 20 64 E2 JSR E264 sinon, efface le fichier et libère ses secteurs
45F- 90 46 BCC E4A7 si OK restructure le catalogue en E4A7, sinon...
461- 60 RTS simple RTS

462- 20 2A D6 JSR D62A XAFCAR affiche le caractère ASCII contenu dans A
465- 20 06 D2 JSR D206 CBF0/ROM va à la ligne
468- 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)
46B- AE 27 C0 LDX C027 X = POSNMX
46E- 20 48 DB JSR DB48 vérifie que (éventuel met) le secteur de catalogue est en place
471- F0 34 BEQ E4A7 branche en E4A7 si fini
473- 20 B4 DA JSR DAB4 affiche le nom de fichier présent à POSNMX dans BUF3
476- 2C 72 C0 BIT C072 teste flag b7
479- 30 20 BMI E49B branche en E49B si DELBAK ou DESTROY
47B- A2 0A LDX #0A si DEL, indexe message " (Y)es or (N)o:"
47D- 20 6C D3 JSR D36C affiche X+1ème message de zone CEE7 terminé par "caractère + 128"
480- 20 02 D3 JSR D302 JSR EB78/ROM si touche frappée alors N = 1 et A = code ASCII corresp, sinon N = 0
483- 20 A1 D3 JSR D3A1 conversion minuscule en MAJUSCULE
486- C9 4E CMP #4E est-ce "N"?
488- F0 D8 BEQ E462 si oui, reboucle en E462 (au suivant)
48A- C9 1B CMP #1B est-ce "ESC"?
48C- F0 D3 BEQ E461 si oui, simple RTS
48E- C9 59 CMP #59 est-ce "Y"?
490- D0 EE BNE E480 si pas "Y", reprend la saisie
492- 20 2A D6 JSR D62A XAFCAR affiche le "Y" contenu dans A
495- 20 06 D2 JSR D206 CBF0/ROM va à la ligne
498- 20 B4 DA JSR DAB4 affiche le nom de fichier présent à POSNMX dans BUF3
49B- 20 66 E2 JSR E266 XNOMDE DELete ce fichier
49E- B0 C5 BCS E465 si erreur, branche en E465 (au suivant)
4A0- A2 0B LDX #0B si OK, indexe " DELETED CRLF"
4A2- 20 6C D3 JSR D36C affiche X+1ème message de zone CEE7 terminé par "caractère + 128"
4A5- 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).

4A7- A9 00 LDA #00 remise à 0 de F5
4A9- 85 F5 STA F5 (nombre de secteurs de catalogue nécessaires)
4AB- AD 04 C2 LDA C204 AX = nombre de fichiers présents
4AE- AE 05 C2 LDX C205 sur la disquette
4B1- 18 CLC C = 0, retenue pour la 1ère soustraction
4B2- 24 38 BIT 38 et continue en E4B4
4B3- 38 SEC C = 1, pas de retenue pour les soustractions suivantes
4B4- 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.
4B6- E6 F5 INC F5 augmente le nombre de soustractions effectuées
4B8- B0 F9 BCS E4B3 reboucle si pas dépassé
4BA- CA DEX décrémente HH du nombre de fichiers présents
4BB- 10 F6 BPL E4B3 et reboucle si pas négatif
4BD- AE 08 C2 LDX C208 si négatif, c'est fini, lit le nombre actuel
4C0- E4 F5 CPX F5 de secteurs de catalogue et compare au nombre
4C2- 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, le ou les "entrées" libres ont été groupées à la fin du secteur de catalogue (s/p 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
E4C5- A9 14 LDA #14
E4C7- A0 04 LDY #04 AY = piste/secteur du 1er 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 1ère "entrée" de BUF1
E4E4- 20 A5 DB JSR DBA5 cherche POSNMX de 1ère 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 1er caractère de la 1ère "entrée" dans BUF1
E4E9- CC 02 C1 CPY C102 le début de 1ère "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 le bitmap
E531- 20 8A DA JSR DA8A XSMAP sauve secteur de bitmap sur la disquette
E534- 4C A7 E4 JMP E4A7 et reboucle pour voir s'il y a encore un secteur de catalogue à récupérer

EXECUTION COMMANDE SEDORIC REN

Rappel de la syntaxe
REN ancien_nom_de_fichier_ambigu TO nouveau_nom_de_fichier

Renomme les fichiers corresp à l'ancien nom ambigu indiqué selon le modèle proposé avec le nouveau nom de fichier ambigu. 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 Sédoric: ça ne marche pas à tous les coups (bogue). Ici, le "TO" doit être tapé en majuscules.

Saisie des paramètres

537-	20 51 D4	JSR D451	XNFA lit l'ancien nom (NFAa) à TXTPTR et l'écrit dans BUFNOM
53A-	A2 0B	LDX #0B	pour copier 12 caractères de BUFNOM dans BUF1
53C-	BD 29 C0	LDA C029,X	lecture dans BUFNOM (de C029 à C034)
53F-	9D 00 C1	STA C100,X	écriture dans BUF1 (de C100 à C10B)
542-	CA	DEX	caractère suivant
543-	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)

545-	AD 28 C0	LDA C028	
548-	48	PHA	empile préfixe de BUFNOM (n° du drive)
549-	A9 C3	LDA #C3	demande le token "TO" (attention, bogue habituelle: "to" écrit en lettres minuscules ne marche pas: on obtient une "SYNTAX ERROR")
54B-	20 2E D2	JSR D22E	JSR D067/ROM et D3A1/RAMOV demande "TO" à TXTPTR, lit le caractère suivant à TXTPTR et le convertit en MAJUSCULE
54E-	20 51 D4	JSR D451	XNFA lit le nouveau nom (NFAn) à TXTPTR et l'écrit dans BUFNOM
551-	68	PLA	
552-	CD 28 C0	CMP C028	si les drives demandés ne sont pas identiques
555-	D0 13	BNE E56A	branche en D5AC ("INVALID FILE NAME")

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

557-	A2 0C	LDX #0C	pour comparer 13 caractères (bogue?)
559-	BC 29 C0	LDY C029,X	Y = caractère de NFAn dans BUFNOM (par la fin)
55C-	BD 00 C1	LDA C100,X	A = caractère corresp de NFAa dans BUF1
55F-	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
562-	C9 3F	CMP #3F	A est-il un "?"
564-	F0 07	BEQ E56D	si oui, branche en E56D
566-	C0 3F	CPY #3F	si A n'est pas un "?", Y est-il un "?"
568-	D0 07	BNE E571	si ni A ni Y ne sont des "?", continue en E571
56A-	4C AC D5	JMP D5AC	si l'un, mais pas l'autre, "INVALID FILE NAME"
56D-	C0 3F	CPY #3F	Y est-il aussi un "?"
56F-	D0 F9	BNE E56A	si ce n'est pas le cas, "INVALID FILE NAME"
			Les jokers doivent être à la même place dans l'ancien et le nouveau nom
571-	98	TYA	Y, qui est précaire, est sauvé dans BUF1
572-	9D 10 C1	STA C110,X	(dans la zone C110 à C11C qui reçoit NFAn)
575-	CA	DEX	indexe le caractère précédent dans NFAa et NFAn
576-	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

578-	20 2D DB	JSR DB2D	vérifie que la disquette en place est bien une disquette Sédoric, 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
57B-	D0 08	BNE E585	continue en E585 si NFAa existe
57D-	4C DD E0	JMP E0DD	si pas trouvé, "FILE NOT FOUND ERROR"
580-	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)
583-	F0 76	BEQ E5FB	simple RTS si terminé

Sauve les coordonnées de l'"entrée" corresp à l'ancien nom

585-	AD 25 C0	LDA C025	
588-	AC 26 C0	LDY C026	F5 reçoit POSNMP
58B-	85 F5	STA F5	F6 reçoit POSNMS
58D-	84 F6	STY F6	F7 reçoit POSNMX
58F-	86 F7	STX F7	

Construit le nouveau nom et l'écrit dans BUFNOM

591-	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 corresp du NFAn pris dans BUF1. Puis on écrit le caractère de A dans BUFNOM, qui est donc mis à jour.
593-	B9 29 C0	LDA C029,Y	lit caractère de NFAa (ancien) dans BUFNOM
596-	C9 3F	CMP #3F	est-ce un "?" (C = 1 si c'est un "?")
598-	D0 05	BNE E59F	sinon, continue en E59F pour saisir le nouveau

```
E59A-  BD 00 C3   LDA C300,X   si oui, lit caractère valide corresp 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 corresp 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 16ème 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 EXISTS" (le nouveau nom ne doit pas déjà exister)
E5C2-  20 6C D3   JSR D36C     affiche X+1ème 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+1ème 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
```

EXECUTION COMMANDE SEDORIC SEARCH

Rappel de la syntaxe
SEARCH nom_de_fichier_ambigu ou
SEARCH nom_de_fichier_non_ambigu

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

Pour tester le résultat il faut utiliser IF EF=1 ... ou IF -EF ... car une valeur logique n'est TRUE que si elle vaut -1 elle est FALSE pour toute autre valeur, y compris 1.

```
       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 Sédoric, 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
```

606- F0 02 BEQ E60A si pas trouvé, saute l'instruction suivante
608- A9 01 LDA #01 A = #01 (pour "trouvé")
60A- 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

60D- AC 09 C0 LDY C009 Y = DRVDEF (n° drive actif par défaut)
610- 20 9E D3 JSR D39E XCRGOT relit le caractère à TXTPTR et le convertit en MAJUSCULE
613- F0 0D BEQ E622 branche en E622 si fin d'instruction
615- E9 41 SBC #41 rappel: C = 1 si lettre = OK pour soustraction
617- C9 04 CMP #04 teste si c'est une lettre de A à D
619- B0 07 BCS E622 branche en E622 si ce n'est pas le cas
61B- A8 TAY A (n° de drive) -> Y (écrase DRVDEF)
61C- 20 C0 D7 JSR D7C0 vérifie si ce drive est connecté et le valide
61F- 4C 98 D3 JMP D398 XCRGET lit le caractère suivant à TXTPTR et le convertit en MAJUSCULE et retourne
622- 20 C0 D7 JSR D7C0 vérifie si ce drive est connecté et le valide
625- 4C 9E D3 JMP D39E XCRGOT relit le caractère à TXTPTR et le convertit en MAJUSCULE

628- 4C DD E0 JMP E0DD "FILE NOT FOUND ERROR"

62B- 4C D2 E2 JMP E2D2 "nom IS PROTECTED ERROR"

EXECUTION COMMANDE SEDORIC STATUS

Rappel de la syntaxe
STATUS NF (,A adresse_de_chargement) (,T adresse d'exécution) (,AUTO)

Les paramètres doivent être placés dans cet ordre, sous peine de "SYNTAX ERROR". S'il n'y a pas de paramètre, le fichier est forcé en "non AUTO". Le paramètre ",A" permet de changer l'adresse de chargement du fichier, c'est à dire l'adresse de début (l'adresse de fin est calculée et également mise à jour). Le paramètre ",T" permet de changer l'adresse d'exécution du fichier qui est aussi forcé en "AUTO". Le paramètre ",AUTO" force l'exécution automatique du fichier.

Non documenté

Le type de fichier n'est pas sérieusement pris en compte, et quand il l'est, c'est pire que s'il ne l'était pas. STATUS est capable de produire les résultats les plus atroces: attention aux incohérences! Par exemple, lorsque le paramètre ",AUTO" est utilisé, l'adresse d'exécution est mise à jour avec l'adresse de début du fichier (celle qui a été éventuellement mise à jour), si c'est un fichier autre que basic (!) et avec n'importe quoi si c'est un programme BASIC (!!).

On peut cumuler le paramètre ",A" avec ",T" ou avec ",AUTO", mais on ne peut pas cumuler ",T" et ",AUTO".

Pour changer le type du fichier (voir page 100 du manuel) il est nécessaire d'utiliser un éditeur de disquette et d'intervenir dans le secteur de descripteur (octet n°3, voir exemple dans BUF1 en C100).

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

Analyse de la syntaxe

62E- 20 4F D4 JSR D44F XNF lit nom de fichier non-ambigu à TXTPTR et l'écrit dans BUFNOM
631- 20 9E D7 JSR D79E recherche "?" dans BUFNOM revient avec C = 1 si pas trouvé ou "WILDCARD(S) NOT ALLOWED ERROR" si trouvé
634- 20 2D DB JSR DB2D vérifie que la disquette en place est bien une disquette Sédoric, 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
637- F0 EF BEQ E628 si Z = 1, branche vers "FILE NOT FOUND ERROR"
639- BD 0F C3 LDA C30F,X teste le dernier octet de l'"entrée" corresp
63C- 30 ED BMI E62B si b7=1, erreur (nom + "IS PROTECTED")
63E- BD 0C C3 LDA C30C,X
641- BC 0D C3 LDY C30D,X AY = piste/secteur du descripteur principal
644- 20 5D DA JSR DA5D XPBUF1 charge descripteur principal dans BUF1
647- 20 9E D3 JSR D39E XCRGOT relit le caractère à TXTPTR et le convertit en MAJUSCULE
64A- F0 73 BEQ E6BF si fin d'instuction continue en E6BF (non AUTO)
64C- 20 2C D2 JSR D22C D067/ROM exige une "," lit le caractère suivant et le convertit en MAJUSCULE
64F- C9 41 CMP #41 est-ce "A"?
651- D0 39 BNE E68C sinon, poursuit l'analyse en E68C
653- 20 98 D3 JSR D398 si oui, A = caractère suivant (mise à jour TXTPTR)
656- 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)

```
E659-  98          TYA        les octets du nombre YA sont inversés
E65A-  A4 34       LDY 34     YA -> AY (nouvelle adresse de chargement)
E65C-  48          PHA        empile A = LL de cette adresse (HH gardé dans Y)
E65D-  85 F6       STA F6
E65F-  84 F7       STY F7     sauve l'adresse AY dans F6/F7
E661-  38          SEC        prépare soustraction adresse_de_fin - adresse_de_début
E662-  AD 06 C1    LDA C106   LL de l'ancienne adresse de fin
E665-  ED 04 C1    SBC C104   LL de l'ancienne adresse de début
E668-  48          PHA        empile LL de la longueur du fichier
E669-  AD 07 C1    LDA C107   HH de l'ancienne adresse de fin
E66C-  ED 05 C1    SBC C105   HH de l'ancienne adresse de début
E66F-  AA          TAX        sauve dans X, HH de la longueur du fichier
E670-  68          PLA        récupère dans A, LL de la longueur du fichier
E671-  18          CLC        prépare l'addition nouvelle_adresse_de_début + longueur
E672-  65 F6       ADC F6     A + F6 = LL de la nouvelle adresse de fin
E674-  8D 06 C1    STA C106   sauve LL de la nouvelle adresse de fin
E677-  8A          TXA        récupère dans A, HH de la longueur du fichier
E678-  65 F7       ADC F7     A + F7 = HH de la nouvelle adresse de fin
E67A-  8D 07 C1    STA C107   sauve HH de la nouvelle adresse de fin
E67D-  68          PLA        A = LL de la nouvelle adresse de chargement
E67E-  8D 04 C1    STA C104   sauve AY nouvelle adresse de chargement
E681-  8C 05 C1    STY C105   c'est à dire nouvelle adresse de début
E684-  20 9E D3    JSR D39E   XCRGOT relit le caractère à TXTPTR et le convertit en MAJUSCULE
E687-  F0 44       BEQ E6CD   si fin d'instuction, fini, sauve le secteur descripteur
E689-  20 2C D2    JSR D22C   D067/ROM exige une "," lit le caractère suivant et le convertit en MAJUSCULE
E68C-  C9 54       CMP #54    est-ce "T"?
E68E-  D0 12       BNE E6A2   sinon, poursuit l'analyse en E6A2
E690-  20 98 D3    JSR D398   si oui, A = caractère suivant (mise à jour TXTPTR)
E693-  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)
E696-  A6 33       LDX 33     XY = nouvelle adresse d'exécution
E698-  A4 34       LDY 34     (sera sauvée dans tous les cas, même basic!)
E69A-  20 9E D3    JSR D39E   XCRGOT relit le caractère à TXTPTR et le convertit en MAJUSCULE
E69D-  F0 17       BEQ E6B6   si fin d'instuction, OK, continue en E6B6
E69F-  4C 23 DE    JMP DE23   sinon, "SYNTAX ERROR" (le seul paramètre encore non analysé est ",AUTO" qu'on ne
                              peut cumuler avec ",T EN". Si l'ordre des paramètres n'est pas correct, il y aura une "SYNTAX ERROR")
E6A2-  C9 C7       CMP #C7    est-ce le token "AUTO"?
E6A4-  D0 F9       BNE E69F   sinon, "SYNTAX ERROR" (tous les paramètres possibles ont été examinés; s'il reste
                              quelque chose, c'est une erreur)
E6A6-  20 98 D3    JSR D398   si oui, A = caractère suivant (mise à jour TXTPTR)
E6A9-  D0 F4       BNE E69F   si pas fin d'instuction, "SYNTAX ERROR"
E6AB-  2C 03 C1    BIT C103   teste b7 de flag "type de fichier" (1 = BASIC)
E6AE-  30 06       BMI E6B6   si fichier BASIC, continue en E6B6 (on ferme les yeux sur la valeur courante de XY, mais
                              on l'écrira quand même!)
E6B0-  AE 04 C1    LDX C104   si c'est un fichier autre que BASIC,
E6B3-  AC 05 C1    LDY C105   XY = adresse de début
E6B6-  8E 08 C1    STX C108   sauve nouvelle adresse d'exécution
E6B9-  8C 09 C1    STY C109   (une bogue si c'est un programme BASIC!)
E6BC-  A9 01       LDA #01    A = #01 (pour mettre en "AUTO")
E6BE-  2C A9 00    BIT 00A9   et continue en E6C1
E6BF-  A9 00       LDA #00    A = #00 (flag pas de paramètre = "non AUTO")
E6C1-  85 F5       STA F5     sauve temporairement le flag "AUTO/non AUTO"
E6C3-  AD 03 C1    LDA C103   indication du type de fichier
E6C6-  29 FE       AND #FE    1111 1110 mise à zéro du b0 (AUTO/non AUTO)
E6C8-  05 F5       ORA F5     transfert du nouveau flag "AUTO/non AUTO"
E6CA-  8D 03 C1    STA C103   et sauvegarde (bogue si fichier non exécutable)
E6CD-  4C A4 DA    JMP DAA4   XSVSEC re-écrit le secteur descripteur modifié selon DRIVE, PISTE, SECTEUR et
                              RWBUF
```

EXECUTION COMMANDE SEDORIC PROT

Rappel de syntaxe
PROT nom_de_fichier_ambigu ou
PROT nom_de_fichier_non_ambigu

Protège le ou les fichiers spécifiés contre DEL, DESTROY, SAVEO, STATUS etc...

```
E6D0-  A9 80       LDA #80    prépare un b7 à 1 pour PROT
E6D2-  2C A9 00    BIT 00A9   et continue en E6D5
```

EXECUTION COMMANDE SEDORIC UNPROT

Rappel de syntaxe
UNPROT nom_de_fichier_ambigu ou
UNPROT nom_de_fichier_non_ambigu

Annule la protection obtenue avec la commande PROT ci-dessus.

```
6D3- A9 00     LDA #00     prépare un b7 à 0 pour UNPROT
6D5- 85 F9     STA F9      sauve le flag PROT/UNPROT
6D7- 20 51 D4  JSR D451    XNFA lit un nom de fichier ambigu à TXTPTR et l'écrit dans BUFNOM
6DA- D0 61     BNE E73D    vers "SYNTAX ERROR" s'il y a des paramètres (fin d'instruction requise)
6DC- 20 2D DB  JSR DB2D    vérifie que la disquette en place est bien une disquette Sédoric, 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
6DF- F0 1E     BEQ E6FF    "FILE NOT FOUND ERROR"
6E1- 20 A0 D7  JSR D7A0    cherche "?" dans BUFNOM (C = 1 si pas trouvé)
6E4- 90 10     BCC E6F6    si "?" trouvé, continue en E6F6
6E6- AE 27 C0  LDX C027    POSNMX 1er octet de l'"entrée" dans le secteur de catalogue
6E9- BD 0F C3  LDA C30F,X  lit dernier octet de l'"entrée"
6EC- 29 7F     AND #7F     0111 1111 mise à zéro du b7 (flag PROT/UNPROT)
6EE- 05 F9     ORA F9      force b7 avec nouvelle valeur flag PROT/UNPROT
6F0- 9D 0F C3  STA C30F,X  et remet en place
6F3- 4C 82 DA  JMP DA82    XSCAT sauve le secteur de catalogue selon POSNMP et POSNMS
6F6- 20 E6 E6  JSR E6E6    mise à jour flag PROT/UNPROT "entrée" courante
6F9- 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)
6FC- D0 F8     BNE E6F6    reboucle tant qu'il en reste à mettre à jour
6FE- 60        RTS

6FF- 4C DD E0  JMP E0DD    "FILE NOT FOUND ERROR"
```

EXECUTION COMMANDE SEDORIC SYSTEM

Rappel de la syntaxe
SYSTEM lecteur

Définit le lecteur où Sédoric devra lire les blocs externes (banques interchangeables) pour exécuter certaines commandes (INIT, COPY etc...).

```
702- 20 0D E6  JSR E60D    valide drive si indiqué, sinon valide DRVDEF
705- D0 36     BNE E73D    vers "SYNTAX ERROR" s'il y a des paramètres (fin d'instruction requise)
707- 8C 0A C0  STY C00A    sauve drive indiqué ou DRVDEF dans DRVSYS
70A- 60        RTS
```

EXECUTION 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: le KEY SET ne peut pas entrer en action en mode direct. Utilisez le seulement en mode programme ou alors prévoir un reset!

Saisie du paramètre et exécution

```
70B- 20 4D E9  JSR E94D    teste si SET (C = 1) ou OFF (C = 0)
70E- 90 09     BCC E719    continue en E719 si OFF
710- AD 07 03  LDA 0307    si SET, lit HH du latch de T1
713- 8D 05 03  STA 0305    et le copie dans HH du compteur de T1
716- A9 40     LDA #40     pour autoriser les interruptions T1
718- 2C A9 00  BIT 00A9    continue en E71B
719- A9 00     LDA #00     pour interdire les interruptions T1
71B- 8D 0B 03  STA 030B    place A dans le registre des interruptions du 6522
71E- 60        RTS
```

EXECUTION 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"

EXECUTION 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 Sédoric: ça ne marche pas à tous les coups (bogue). Ici, il faut utiliser "LPRINT" et non "lprint".

Saisie du ou des paramètre(s)

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 lit le caractère suivant à TXTPTR et le convertit en MAJUSCULE
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 et le convertit en MAJUSCULE
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

768-	68	PLA	élimine le flag LPRINT (pas de différence)
769-	86 31	STX 31	met à jour le nombre de caractères par ligne pour le périphérique de sortie courant (écran ou imprimante)
76B-	8A	TXA	et garde une copie de cette valeur dans A

Calcule la position de la dernière tabulation

76C-	38	SEC	prépare une soustraction
76D-	E9 08	SBC #08	retire 8 et reboucle tant que le nombre de
76F-	B0 FC	BCS E76D	caractères par ligne est positif ou nul
771-	49 FF	EOR #FF	masque 1111 1111, inverse les bits du résultat
773-	E9 06	SBC #06	et retire 6
775-	18	CLC	prépare une addition
776-	65 31	ADC 31	résultat précédent + nombre de caractères par ligne
778-	85 32	STA 32	donne position maxi pour tabulation par ","
77A-	60	RTS	

WIDTH pour ROM V 1.1

77B-	2C F1 02	BIT 02F1	teste le flag imprimante (b7 à 1 si périphérique courant)
77E-	10 0A	BPL E78A	continue en E78A si l'écran est le périphérique courant

L'imprimante est le périphérique courant

780-	68	PLA	teste le flag "LPRINT présent dans la commande WIDTH"

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

781-	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

783-	8E 57 02	STX 0257	sinon met à jour la longueur d'une ligne écran
786-	8D 59 02	STA 0259	force à zéro la position horizontale écran
789-	60	RTS	

L'écran est le périphérique courant

78A-	68	PLA	teste flag LPRINT présent dans la commande WIDTH"

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

78B-	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

78D-	8E 56 02	STX 0256	sinon met à jour la longueur d'une ligne imprimante
790-	A9 00	LDA #00	
792-	8D 58 02	STA 0258	force à zéro la position horizontale imprimante
795-	60	RTS	

EXECUTION 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

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

Utilise la valeur indiquée comme germe

798-	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
79B-	4C E2 D2	JMP D2E2	JSR E37D/ROM génère un nombre entre 0 et 1

Utilise les timers du VIA comme germe

79E-	AD 04 03	LDA 0304	LL du compteur T1
7A1-	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"
```

EXECUTION 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).

```
Et voilà c'est parti!
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/RAMOV)
E7BD-  4C AD 04   JMP 04AD    et effectue un coldstart FFFC/ROM
```

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

Utilisation en LM
Pour PR SET, passer sur la RAMOV, 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    teste si SET (C = 1) ou OFF (C = 0)

PR OFF
E7C3-  90 11      BCC E7D6    continue en D1C4 si OFF (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
```

EXECUTION 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
```

EXECUTION COMMANDE SEDORIC RESTORE

Rappel de la syntaxe
!RESTORE (n°_de_ligne) ou **restore (n°_de_ligne)**
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 Sédoric, qui seul peut être renuméroté (RENUM), il faut utiliser impérativement la syntaxe indiquée ci-dessus. La première forme est impérativement recommandée, car 1) l'utilisation des caractères minuscles pour les commandes Sédoric est peu pratique, 2) elle est très imparfaite et souffre de beaucoup trop d'exceptions et surtout 3) la forme "restore" n'est pas reconnue par RENUM. Il serait possible d'envisager de supprimer cette utilisation des caractères minuscles et d'ajouter de nouvelles commandes à la place.

Mal documenté

Attention, tout l'intérêt du RESTORE Sédoric par rapport au RESTORE BASIC est de pouvoir indiquer un n° de ligne. Donc, si l'on ne veut pas courrir le risque d'oublier que RENUM ne met pas à jour la forme "restore" en minuscules, mieux vaut ne jamais l'utiliser!

Entrée de la commande !RESTORE/restore

```
7D9- 08          PHP          sauvegarde les indicateurs 6502 dont Z
7DA- A6 9A       LDX 9A       XY adresse de début du programme BASIC
7DC- A4 9B       LDY 9B       pour valeur par défaut (1er lien de ligne)
7DE- 28          PLP          récupère les indicateurs DONT Z
7DF- F0 0A       BEQ E7EB     continue en E7EB s'il n'y a pas de paramètre
7E1- 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 mis à jour sur l'octet suivant ce nombre
7E4- 20 9C D1    JSR D19C     JSR C6B3/ROM cherche l'adresse de cette ligne BASIC
7E7- A6 CE       LDX CE       XY pointe sur LL du lien de cette ligne
7E9- A4 CF       LDY CF       il faut pointer sur le #00 qui précède
7EB- 8A          TXA          teste si LL est nul
7EC- D0 01       BNE E7EF     sinon, se contente de décrémenter LL
7EE- 88          DEY          si oui, décrémente HH puis
7EF- CA          DEX          décrémente LL
7F0- 86 B0       STX B0       place l'adresse du #00 précédant le début de ligne
7F2- 84 B1       STY B1       dans le pointeur de DATA
7F4- 60          RTS
```

EXECUTION COMMANDE SEDORIC QUIT

Rappel de la syntaxe

QUIT tout court

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

Non documenté

Comment revenir à l'état initial sous Sédoric? 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 interessant de programmer une commande qui ferait le contraire de QUIT et qui pourrait s'appeler RETRIEVE par exemple.

```
7F5- D0 BE       BNE E7B5     "SYNTAX ERROR" (tout paramètre est interdit avec QUIT)
7F7- AD 3E 04    LDA 043E
7FA- AC 3F 04    LDY 043F     redirige le vecteur interpréteur F0/F1 sur ECB9
7FD- 85 F0       STA F0       au lieu de 0400 (la page 4 n'est plus utilisée)
7FF- 84 F1       STY F1
801- 08          PHP          sauve les indicateurs 6502 dont I
802- 78          SEI          interdit les interruptions
803- 2C 24 C0    BIT C024     teste ATMORI (#00 si V1.0 et #80 si V1.1)
806- 10 20       BPL E828     branche en E828 si Oric-1

                 Atmos
808- A9 22       LDA #22
80A- A0 EE       LDY #EE      redirige le vecteur IRQ 0245/0246
80C- 8D 45 02    STA 0245     sur EE22 au lieu de 0488
80F- 8C 46 02    STY 0246
812- A9 78       LDA #78
814- A0 EB       LDY #EB      redirige le s/p "Gerer le tampon touche" 023C/023D
816- 8D 3C 02    STA 023C     sur EB78 au lieu de 045B (les touches de
819- 8C 3D 02    STY 023D     fonctions ne sont plus accessibles)
81C- A9 B2       LDA #B2
81E- A0 F8       LDY #F8      redirige le vecteur NMI 0248/0249
820- 8D 48 02    STA 0248     sur F8B2 au lieu de 04C4
823- 8C 49 02    STY 0249
826- 28          PLP          récupère les indicateurs dont I
827- 60          RTS          et retourne sur Atmos

                 Oric 1
828- A9 03       LDA #03
82A- A0 EC       LDY #EC      redirige le vecteur IRQ 0229/022A
82C- 8D 29 02    STA 0229     sur EC03 au lieu de 0488
82F- 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 Sédoric (en CC27/CCF6). L'annexe 6 du manuel (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 ailleur dans la RAMOV, 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 s/p 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
```

EXECUTION COMMANDE SEDORIC STRUN

Rappel de la syntaxe
STRUN expression_alphanumérique

C'est une des commandes les plus géniales de Sédoric. Elle execute, en mode programme uniquement, l'expression alphanumérique indiquée qui doit correspondre à une ligne de commande BASIC et/ou Sédoric. Tout ce passe comme si la ligne corresp était exécutée en mode direct. Il est possible d'utiliser non seulement les mots-clés BASIC et Sédoric, 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 prédéfinies 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
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 Sédoric 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 LM il faudrait faire:

```
JSR 0472     (passage sur la RAMOVerlay)
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 LM
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 (voir Annexe F).

Saisie et vérification du paramètre

```
E853-  20 5C D2   JSR D25C     JSR D4D2/ROM interdire 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

```
860-  AA          TAX         longueur de la chaîne tokenisée
861-  A8          TAY         idem
862-  88          DEY         pour copier les A octets de la chaîne
863-  B1 91       LDA (91),Y  lit un octet de la chaîne
865-  99 35 00    STA 0035,Y  et le copie dans le TIB (tampon clavier)
868-  88          DEY         décrémente le compteur
869-  10 F8       BPL E863    reboucle en E863 s'il en reste à copier
86B-  C8          INY         qui passe à zéro au premier tour
86C-  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
86F-  95 35       STA 35,X    et qui constitue une fausse ligne de commande:
871-  E8          INX         CALL#467 avec #FF comme argument
872-  C0 0A       CPY #0A     teste si 11 octets sont copiés
874-  D0 F5       BNE E86B    sinon reboucle en E86B
876-  A5 E9       LDA E9
878-  A4 EA       LDY EA      sauve la valeur actuelle de TXTPTR en C013/C014
87A-  8D 13 C0    STA C013
87D-  8C 14 C0    STY C014
880-  A5 A8       LDA A8
882-  A4 A9       LDY A9      et celle du n° de ligne courante en C011/C012
884-  8D 11 C0    STA C011
887-  8C 12 C0    STY C012
88A-  A9 34       LDA #34
88C-  A0 00       LDY #00     reinitialise TXTPTR avec l'adresse du TIB (0034)
88E-  A2 3A       LDX #3A
890-  85 E9       STA E9
892-  84 EA       STY EA
894-  86 34       STX 34      place ":" en avant du TIB
896-  88          DEY
897-  84 A9       STY A9      place #FF (mode immédiat) dans le n° de ligne
899-  60          RTS
```

L'interpréteur continue à TXTPTR, c'est à dire dans le TIB, execute la chaîne, puis le CALL#467. Le s/p 0467 exécute la commande ! avec son argument #FF. Ce #FF est interprété comme le code d'une fonction: le s/p 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.

```
89A-  4C 77 E9    JMP E977    "STRING TOO LONG ERROR"
```

EXECUTION 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 Sédoric. 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 corresp. 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

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.

```
89D-  20 5C D2    JSR D25C    JSR D4D2/ROM interdit le mode direct
8A0-  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
8A3-  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
8A6-  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 corresp
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.

Mise à jour de la longueur et de l'adresse de la chaîne

```
E8D6- A0 02      LDY #02
E8D8- B9 D0 00   LDA 00D0,Y   copy 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
```

EXECUTION 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 corresp. 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
```

```
8EA-  85 F3      STA F3       sinon, sauve la longueur de la chaîne
8EC-  A2 00      LDX #00      initialise la longueur de la nouvelle chaîne
8EE-  A0 00      LDY #00      index de lecture
```

Parcourt la chaîne source à la recherche des tokens

```
8F0-  A9 E9      LDA #E9
8F2-  85 16      STA 16       16/17 = début de la table des mots-clés
8F4-  A9 C0      LDA #C0      (en C0EA/ROM)
8F6-  85 17      STA 17
8F8-  84 F2      STY F2       sauve l'index de lecture
8FA-  B1 91      LDA (91),Y   lit un octet de la chaîne source
8FC-  10 2B      BPL E929     ce n'est pas un token, suite en E929
```

Un token a été trouvé

```
8FE-  29 7F      AND #7F      en force le b7 à 0 ce qui donne son n° d'ordre
900-  F0 13      BEQ E915     continue en E915 si c'était #80 (n° d'ordre #00)
902-  85 26      STA 26       sinon sauve le n° d'ordre du mot-clé
904-  A0 00      LDY #00      initialise Y pour le s/p 0453 ci-après
```

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

```
906-  E6 16      INC 16
908-  D0 02      BNE E90C     incrémente le pointeur dans la table des mots-clés
90A-  E6 17      INC 17
90C-  20 53 04   JSR 0453     lit le caractère du mot-clé dans la table en ROM
90F-  10 F5      BPL E906     ce n'est pas la fin du mot-clé, reboucle en E906
911-  C6 26      DEC 26       fin du mot-clé atteinte, décrémente le compteur de mots-clés (n° d'ordre du mot-clé dans la
                              table)
913-  D0 F1      BNE E906     ce n'est pas le bon, reboucle en E906
```

Le bon mot-clé a été trouvé dans la table

```
915-  A0 01      LDY #01      c'est le bon, indexe le début du mot-clé
917-  E8         INX          longueur de la nouvelle chaîne (index d'écriture)
918-  F0 1E      BEQ E938     "STRING TOO LONG ERROR" si longueur > # 100 caractères
91A-  20 53 04   JSR 0453     lit un caractère du mot-clé dans la table en ROM
91D-  08         PHP          sauvegarde les indicateurs 6502 dont N
91E-  29 7F      AND #7F      en force le b7 à 0, même si ce n'est pas le dernier
920-  9D FF C0   STA C0FF,X   et le copie dans BUF1
923-  C8         INY          vise le caractère suivant du mot-clé
924-  28         PLP          récupère les indicateurs dont N
925-  10 F0      BPL E917     reboucle en E917 tant que la fin n'est pas atteinte
927-  30 06      BMI E92F     continue en E92F si le dernier caractère a été traité
```

Ce n'était pas un token

```
929-  E8         INX          longueur de la nouvelle chaîne
92A-  F0 0C      BEQ E938     "STRING TOO LONG ERROR" si longueur > # 100 caractères
92C-  9D FF C0   STA C0FF,X   copie cet octet "non token" dans BUF1
```

Octet suivant de la chaîne source

```
92F-  C6 F3      DEC F3       décrémente la longueur de chaîne source
931-  F0 08      BEQ E93B     suite en E93B si tous les octets ont été traités
933-  A4 F2      LDY F2       sinon, récupère l'index de lecture
935-  C8         INY          et l'incrémente
936-  D0 B8      BNE E8F0     reprise forcée en E8F0 tant que longueur < #100
938-  4C 77 E9   JMP E977     sinon, "STRING TOO LONG ERROR"
```

Tous les octets de la chaîne source ont été traités

```
93B-  8A         TXA          nouvelle longueur de la chaîne
93C-  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
93F-  A4 D0      LDY D0       longueur de la nouvelle chaîne
941-  88         DEY          index de copie
942-  B9 00 C1   LDA C100,Y   lit un octet de BUF1
945-  91 D1      STA (D1),Y   et le copie dans l'emplacement réservé en ROM
947-  98         TYA          teste si Y atteint 0 (terminé)
948-  D0 F7      BNE E941     reboucle en E941 si ce n'est pas le cas
94A-  4C D6 E8   JMP E8D6     suite en E8D6 si terminé (mise à jour de la variable)
```

Analyse de syntaxe, retourne C = 1 si SET ou C = 0 si OFF sinon "SYNTAX ERROR"

(appelé par les commandes ACCENT, ERR et KEY)

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 minMAJ)
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 minMAJ)
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"

EXECUTION COMMANDE SEDORIC ERR

Rappel de la syntaxe
ERR SET ou **ERR OFF**

Lorsqu'une erreur concernant Sédoric se produit (par exemple "INVALID FILE NAME ERROR"), le programme s'arrête: par défaut Sédoric 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 débrouiler 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 Sédoric 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 comande 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 Sédoric, soit "USER x ERROR" (x étant le n° de l'erreur utilisateur). Il est donc indispensable d'avoir un ERRGOTO après chaque ERRSET.

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	C = 1 si SET, C = 0 si OFF ou "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"
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	

EXECUTION 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 Sédoric, soit "USER x ERROR" (x étant le n° de l'erreur utilisateur). Il est donc indispensable d'avoir un ERRGOTO après chaque ERRSET.
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

999- 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)
99C- 8D 1C C0 STA C01C et l'écrit en C01B/C01C
99F- 8C 1B C0 STY C01B (n° de la ligne de gestion des erreurs)
9A2- 20 9C D1 JSR D19C JSR C6B3/ROM cherche l'adresse de cette ligne BASIC
9A5- A6 CF LDX CF
9A7- A4 CE LDY CE YX = adresse de cette ligne
9A9- D0 01 BNE E9AC calcule YX = YX - #01
9AB- CA DEX
9AC- 88 DEY
9AD- 4C 92 E9 JMP E992 copie YX dans C019/C01A (adresse où ajuster TXTPTR)

EXECUTION COMMANDE SEDORIC ERROR

Rappel de la syntaxe
ERROR n°_d'erreur

Lorsqu'elle est rencontrée, cette commande déclenche 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 (Sédoric et utilisateur, mais pas BASIC) dans le même s/p. 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

9B0- 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)
9B3- E0 32 CPX #32 teste si X < 50 (c'est à dire si erreur Sédoric)
9B5- 90 C5 BCC E97C si oui, "ILLEGAL QUANTITY ERROR" (50 à 255 SVP!)
9B7- CA DEX pour neutraliser l'incrémentation qui suit
9B8- 4C 7E D6 JMP D67E incrémente X et traite l'erreur n° X

EXECUTION COMMANDE SEDORIC RESUME

Rappel de la syntaxe
RESUME ou **RESUME NEXT**

A la fin du traitement de l'erreur par le s/p 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 s/p 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 Sédoric: 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

9BB- F0 06 BEQ E9C3 suite en E9C3 si pas de paramètre (avec Z = 1)
9BD- A9 90 LDA #90 A = token "NEXT"
9BF- 20 2E D2 JSR D22E JSR D067/ROM et D3A1/RAMOV demande un "NEXT" à TXTPTR, lit le caractère suivant et le convertit en MAJUSCULE
9C2- C8 INY Y a été mis à zéro dans le sous-programme ci-dessus et passe ici à #01 pour mettre Z à zéro ("NEXT" présent) (un peu tortueux!)

Re-initialise TXTPTR et n° de ligne

9C3- 08 PHP sauvegarde les indicateurs 6502 dont Z
9C4- 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
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'interprèteur
```

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 dangeureuse 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
```

EXECUTION COMMANDE SEDORIC EXT

Rappel de la syntaxe
EXT expression_alphanumérique ou **EXT ?** ou **EXT**

L'expression alphanumérique doit comporter 3 caractères. Ce peut être une chaîne ou une variable alphanumérique. Il est impossible de compléter avec des espaces (sauf bogue, voir ci-après). Si l'expression alphanumérique est remplacée par un "?" (EXT PRINT marche aussi!), l'extension courante est affichée. En absence de paramètre, l'extension courante est remise à sa valeur initiale "COM". Une chaîne vide est refusée, il faut mettre 3 espaces.

Non documenté
Une bogue fait que la validité du troisième caractère de l'extension n'est pas vérifiée. Il est donc possible de mettre n'importe quoi comme troisième caractère. Mais attention quand même, ce n'est pas sans risque: une extention à "CO?" est acceptée, mais les fichiers "*.CO?" ne le seront pas!
Enfin, petite curiosité, EXT PRINT est accepté, mais pas EXT print: toujours le même problème avec l'utilisation possible des minuscules pour entrer les commandes Sédoric: ça ne marche pas à tous les coups (bogue).

Analyse de la syntaxe et des paramètres

```
E9ED- D0 0C      BNE E9FB     continue en E9FB s'il y a des paramètres
```

Re-initialise l'extension courante avec "COM"

```
E9EF- A2 02      LDX #02      si pas de paramètre, remet COM par défaut
E9F1- BD FD CC   LDA CCFD,X   lit 3 caractères à partir de CCFD (COM = défaut)
E9F4- 9D F7 CC   STA CCF7,X   et les copie à partir de CCF7 (EXT courante)
E9F7- CA         DEX          caractère précédent
E9F8- 10 F7      BPL E9F1     reboucle en E9F1 tant qu'il en reste à copier
E9FA- 60         RTS          et retourne
```

Il y a un paramètre

```
E9FB- C9 BA      CMP #BA      le paramètre est-il un "?" (le token PRINT)
E9FD- D0 11      BNE EA10     sinon, poursuit l'analyse de syntaxe en EA10
```

Si c'est un "?", affiche l'extension courante

```
E9FF- A0 FD      LDY #FD      prépare l'index Y pour afficher les 3 caractères de l'extension courante (avec Y = #FD, #FE,
      #FF). D'aucuns trouvent que finalement cet artifice n'est pas si génial... mais ils ont tort et le tortue!
EA01- B9 FA CB   LDA CBFA,Y   lus à partir de CCF7
EA04- 20 2A D6   JSR D62A     XAFCAR affiche le caractère ASCII contenu dans A
EA07- C8         INY          caractère suivant
EA08- D0 F7      BNE EA01     jusqu'à ce que Y soit nul
EA0A- 20 06 D2   JSR D206     CBF0/ROM va à la ligne
EA0D- 4C 3A D3   JMP D33A     JSR 00E2/ROM incrémente TXTPTR, lit un caractère (CHRGET), les espaces sont sautés,
      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 et retourne au
      programme précédent
```

Analyse la validité de l'extension indiquée

A10- 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
A13- 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
A16- C9 03 CMP #03 cette chaîne a t-elle 3 caractères?
A18- D0 1E BNE EA38 sinon, "INVALID FILE NAME ERROR"
A1A- A0 02 LDY **#02** prépare pour index #01 en début de boucle (hélas!)
A1C- 88 DEY seuls Y = 1 et Y = 0 seront valides (bogue)
A1D- 30 0B BMI EA2A sortie de boucle en EA2A lorsqu'Y devient négatif
A1F- B1 91 LDA (91),Y lit un caractère de la chaîne (par la fin)
A21- 20 A1 D3 JSR D3A1 le convertit en MAJUSCULE si nécessaire
A24- 20 C7 D5 JSR D5C7 vérifie que caractère est valide (lettre ou chiffre)
A27- 4C 1C EA JMP EA1C reboucle obligatoirement en EA1C

Copie cette extension en RAMOV

A2A- A0 02 LDY #02 cette fois, indexe pour 3 caractères (Y = 2, 1, et 0)
A2C- B1 91 LDA (91),Y lit un caractère de la chaîne (par la fin)
A2E- 20 A1 D3 JSR D3A1 le convertit en MAJUSCULE si nécessaire
A31- 99 F7 CC STA CCF7,Y et l'écrit dans la zone CCF7/CCF9 (EXT courante)
A34- 88 DEY caractère précédent
A35- 10 F5 BPL EA2C reboucle tant que Y n'est pas négatif
A37- 60 RTS et termine sans vérification de la validité du 3ème caractère (bogue en EA1B: #03 aurait été préférable!)
A38- 4C AC D5 JMP D5AC "INVALID FILE NAME ERROR"

EXECUTION 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.

A3B- 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
A3E- 85 B8 STA B8
A40- 84 B9 STY B9 B8/B9 reçoit aussi cette adresse
A42- A5 28 LDA 28
A44- 48 PHA empile le type de la variable trouvée à TXTPTR
A45- A5 29 LDA 29
A47- 48 PHA
A48- 20 2C D2 JSR D22C D067/ROM exige une "," lit le caractère suivant et le convertit en MAJUSCULE
A4B- 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
A4E- 85 91 STA 91
A50- 84 92 STY 92 91/92 reçoit aussi cette adresse
A52- 68 PLA
A53- C5 29 CMP 29 compare le type des deux variables
A55- D0 20 BNE EA77 si différent, "TYPE MISMATCH ERROR"
A57- 68 PLA
A58- C5 28 CMP 28
A5A- D0 1B BNE EA77

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

A5C- A0 01 LDY #01 index par défaut pour copie si type "nombre entier"
A5E- 24 28 BIT 28 teste si b7 de 28 à 1 (type "chaîne")
A60- 30 06 BMI EA68 si oui, continue en EA68
A62- 24 29 BIT 29 teste si b7 de 29 à 1 (type "nombre entier")
A64- 30 03 BMI EA69 si oui, continue en EA69
A66- C8 INY Y = nombre d'octets à copier selon type de variable
A67- C8 INY 2 octets en plus pour un nombre réel
A68- C8 INY 1 octet en plus pour un nombre réel ou une chaîne
A69- B1 91 LDA (91),Y lit un octet de la variable 2
A6B- AA TAX et le place temporairement dans X
A6C- B1 B8 LDA (B8),Y lit un octet de la variable 1 et le copie à
A6E- 91 91 STA (91),Y l'emplacement homologue de la variable 2
A70- 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"
```

EXECUTION 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 est très peu utilisée, car le manuel n'explique pas assez clairement son mode d'emploi. Elle permet d'appeler des routines en LM 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 s/p LM au maximum avec ces commandes).

La 1ère forme permet de définir l'adresse d'exécution de la routine. Le paramètre ",O" sert à indiquer que cette routine se trouve en RAMOV ("O" pour Overlay).

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.

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 Sédoric 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). La seule différence avec CALL est 1) que l'on peut travailler directement en RAMOV 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 Sédoric. 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 Sédoric: 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 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" corresp
EA90- 85 F7     STA F7      à ce n° de routine
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

A9B-	20 9E D3	JSR D39E	XCRGOT relit le caractère à TXTPTR et le convertit en MAJUSCULE
A9E-	C9 2C	CMP #2C	le caractère à TXTPTR est-il une virgule?
AA0-	D0 46	BNE EAE8	sinon, termine en EAE8
AA2-	20 98 D3	JSR D398	XCRGET lit le caractère suivant à TXTPTR et le convertit en MAJUSCULE
AA5-	A0 04	LDY #04	doit être un des 5 cas suivants: A, X, Y, P ou DEF
AA7-	D9 83 CD	CMP CD83,Y	compare avec le contenu de la table CD83/CD87
AAA-	F0 05	BEQ EAB1	si trouvé, continue en EAB1 avec Y positionné
AAC-	88	DEY	sinon, vise l'octet précédent de la table
AAD-	10 F8	BPL EAA7	reboucle tant qu'il en reste à examiner
AAF-	30 37	BMI EAE8	rien trouvé, continue en EAE8 (le paramètre n'est ni une valeur de registre, ni DEF, mais un paramètre utilisateur)

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

AB1-	20 98 D3	JSR D398	XCRGET lit le caractère suivant à TXTPTR et le convertit en MAJUSCULE (en fait place TXTPTR sur le caractère suivant)
AB4-	C0 04	CPY #04	teste l'index Y: était-ce le token DEF?
AB6-	D0 22	BNE EADA	sinon, continue en EADA

Traite le token DEF

AB8-	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)
ABB-	A6 F6	LDX F6	récupère l'index sauvé en EA8A (valant de 0 à 9)
ABD-	9D 67 C0	STA C067,X	écrit HH du nombre évalué dans la table C066/C071
AC0-	98	TYA	idem avec LL (place l'adresse d'exécution de la
AC1-	9D 66 C0	STA C066,X	routine dans la table des adresses C066/C071)
AC4-	20 9E D3	JSR D39E	XCRGOT relit le caractère à TXTPTR et le convertit en MAJUSCULE
AC7-	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)
AC9-	20 2C D2	JSR D22C	D067/ROM exige une "," lit le caractère suivant et le convertit en MAJUSCULE
ACC-	A9 4F	LDA #4F	caractère "O" majuscule (toujours le même problème)
ACE-	20 2E D2	JSR D22E	JSR D067/ROM et D3A1/RAMOV demande un "O" à TXTPTR, lit le caractère suivant et le convertit éventuellement en MAJUSCULE
AD1-	A2 80	LDX #80	flag négatif qui indique que cette routine se
AD3-	8A	TXA	trouve en RAMOV (et/ou lower RAM)
AD4-	A6 F6	LDX F6	récupère l'index sauvé en EA8A (valant de 0 à 9)
AD6-	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
AD9-	60	RTS	

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

ADA-	98	TYA	index Y significatif du code trouvé (A, Y, X ou P)
ADB-	48	PHA	empile cet index
ADC-	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)
ADF-	68	PLA	récupère index
AE0-	A8	TAY	et le passe dans Y
AE1-	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
AE3-	20 9E D3	JSR D39E	XCRGOT relit le caractère à TXTPTR et le convertit en MAJUSCULE, en fait, ce JSR sert uniquement à forcer la reprise en EA9B et chose curieuse, en EA9B se trouve un autre JSR D39E!
AE6-	D0 B3	BNE EA9B	reboucle en EA9B pour traiter le paramètre suivant

Exécution de la routine

AE8-	A4 F4	LDY F4	valeur indiquée pour le registre Y
AEA-	A5 F5	LDA F5	valeur indiquée pour le registre P
AEC-	48	PHA	empile P
AED-	A6 F6	LDX F6	récupère l'index sauvé en EA8A (valant de 0 à 9)
AEF-	BD 66 C0	LDA C066,X	
AF2-	8D F0 04	STA 04F0	copie l'adresse d'exécution dans EXEVEC
AF5-	BD 67 C0	LDA C067,X	en page 4
AF8-	8D F1 04	STA 04F1	
AFB-	A5 F2	LDA F2	valeur indiquée pour le registre A
AFD-	A6 F3	LDX F3	valeur indiquée pour le registre X
AFF-	24 F7	BIT F7	teste si la routine se trouve en ROM
B01-	10 07	BPL EB0A	si oui, continue en EB0A

Exécution en RAMOV (et/ou RAM < C000)

B03-	28	PLP	sinon, récupère les indicateurs
B04-	20 22 EB	JSR EB22	appelle la routine en RAMOV selon EXEVEC
B07-	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 RAMOV (et/ou lower RAM)

EXECUTION COMMANDE SEDORIC NUM

Rappel de la syntaxe
NUM (n°_de_la_première_ligne)(,pas_de_la_renumérotation) ou **NUM END**

Permet de modifier en mémoire les paramètres de la renumérotation 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!

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

Régénère les valeurs à utiliser pour la renumérotation 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 et le convertit en MAJUSCULE
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 lit le caractère suivant à TXTPTR et le convertit en MAJUSCULE
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	

```
B49-  C8          INY
B4A-  B1 CE       LDA (CE),Y    XA reçoit le lien (adresse de la ligne suivante)
B4C-  F0 10       BEQ EB5E      si HH nul, fin du programme BASIC atteinte, continue en EB5E (seule sortie de cette
      boucle)
B4E-  48          PHA           sauvegarde provisoirement le registre A
B4F-  C8          INY
B50-  B1 CE       LDA (CE),Y
B52-  8D 42 C0    STA C042
B55-  C8          INY
B56-  B1 CE       LDA (CE),Y
B58-  8D 43 C0    STA C043      C042/C043 reçoit le n° de la ligne courante
B5B-  68          PLA           récupère A qui n'était pas nul
B5C-  D0 E2       BNE EB40      donc rebouclage obligatoire en EB40
```

Calcule le n° de la ligne suivante

```
B5E-  18          CLC
B5F-  AD 42 C0    LDA C042
B62-  6D 44 C0    ADC C044      n° de la dernière ligne trouvé et placé en
B65-  8D 42 C0    STA C042      C042/C043, calcule le n° de la ligne suivante en
B68-  AD 43 C0    LDA C043      ajoutant TRAVPAS (C044/C045) à TRAVNUM (C042/C043)
B6B-  6D 45 C0    ADC C045
B6E-  8D 43 C0    STA C043      C042/C043 = C042/C043 + C044/C045
B71-  60          RTS           et retourne
```

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

```
B72-  C9 2C       CMP #2C       le caractère à TXTPTR est-il une virgule?
B74-  F0 0E       BEQ EB84      si oui (1er paramètre absent), continue en EB84
B76-  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)
B79-  8D 43 C0    STA C043
B7C-  8C 42 C0    STY C042      place cette valeur dans C042/C043 (TRAVNUM)
B7F-  20 9E D3    JSR D39E      XCRGOT relit le caractère à TXTPTR et le convertit en MAJUSCULE
B82-  F0 0C       BEQ EB90      simple RTS en EB90 si fin d'instruction
B84-  20 2C D2    JSR D22C      D067/ROM exige une "," place TXTPTR sur l'octet suivant
B87-  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)
B8A-  8D 45 C0    STA C045
B8D-  8C 44 C0    STY C044      place cette valeur dans C044/C045 (TRAVPAS)
B90-  60          RTS           et retourne
```

EXECUTION 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 régénérés sont eux utilisables en mode HIRES, il faut donc effectuer la commande ACCENT avant de passer sous HIRES.

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

Non documenté

Il n'est pas possible d'obtenir directement le "ê" au clavier. Seul un CHR$(#7E) ou CHR$(126) peut être utilisé. Sinon, il faut redéfinir une touche de fonction, c'est à dire faire:

```
VUSER pour voir quelles définitions utilisateur sont actuellement en usage. Supposons que la n°6 soit libre
KEYDEF 6 puis appuyer sur FUNCT et sur ^
KEYUSE 6,CHR$(126)
```

KEYSAVE"ACCENT.KEY" (n'importe quel nom est valable)

Par la suite, il faut charger ACCENT.KEY, par exemple en ajoutant !LOAD"ACCENT.KEY" à l'INIST de la disquette. L'appui sur FUNCT et ^ affichera un ê.

EB91-	20 4D E9	JSR E94D	analyse de syntaxe (C = 1 si SET, C = 0 si OFF)
EB94-	20 DE DF	JSR DFDE	vérifie si bien mode TEXT (C n'est pas modifié)
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

Sélectionne le jeu de caractères correct

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 s/p F982/ROM: copie les caractères
EBAA-	4C 32 D3	JMP D332	normaux de la ROM dans la RAM et retourne

ACCENT SET: redéfinit de 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 tranférer (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 redéfinir
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 notre 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 redéfinir
EBDB-	D0 D6	BNE EBB3	reboucle tant qu'il en reste
EBDD-	60	RTS	

EXECUTION COMMANDE SEDORIC AZERTY

Rappel de la syntaxe
AZERTY

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 générés sont eux utilisables en mode HIRES, il faut donc effectuer la commande AZERTY avant de passer sous HIRES.

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

EXECUTION COMMANDE SEDORIC QWERTY

Rappel de la syntaxe
QWERTY

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

```
BE1- A9 00      LDA #00      masque 0000 0000 pour forcer les b5 et b7 à 0
BE3- 8D 3D C0   STA C03D     écrit le flag dans MODCLA. C'est la routine "Prendre un caractère au clavier" (D845) qui
teste MODCLA et affecte tel ou tel code ASCII selon la touche pressée.
BE6- 20 DE DF   JSR DFDE     vérifie si bien mode TEXT (c'est un peu tard)
BE9- 4C A3 EB   JMP EBA3     sélectionne le jeu de caractères correct
```

EXECUTION 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.

```
BEC- AD 69 02   LDA 0269     colonne du curseur TEXT
BEF- AC 68 02   LDY 0268     ligne du curseur TEXT
BF2- 4C FB EB   JMP EBFB     initialise les variables CX et CY
```

EXECUTION 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.

```
BF5- AD 19 02   LDA 0219     colonne du curseur HIRES
BF8- AC 1A 02   LDY 021A     ligne du curseur HIRES
```

Initialise les variables CX et CY

```
BFB- 48         PHA          empile le n° de colonne
BFC- 98         TYA          prend le n° de ligne
BFD- 20 E7 D7   JSR D7E7     et le copie dans la variable CY
C00- 68         PLA          récupère le n° de colonne
C01- 4C E4 D7   JMP D7E4     le copie dans la variable CX et retourne
```

EXECUTION 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).

```
C04- 08         PHP          sauvegarde les indicateurs 6502
C05- 48         PHA          sauvegarde le registre A
C06- AD F9 02   LDA 02F9
C09- AC FA 02   LDY 02FA     l'adresse présente au vecteur ] est
C0C- 8D F0 04   STA 04F0     copiée dans EXEVEC
C0F- 8C F1 04   STY 04F1
C12- 68         PLA          récupère le registre A
C13- 28         PLP          récupère les indicateurs
C14- 4C EC 04   JMP 04EC     exécute la routine indiquée à EXEVEC sur RAMOV (si ROM active) ou sur ROM (si
RAMOV active)
```

Initialise PAPER, INK et status console
Ce s/p arrive ici comme un cheveu sur la soupe: il est appelé par le s/p NMI Sédoric en 04C4, avant de passer au warmstart BASIC.

```
C17- 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    sélectionne le jeu de caractères correct
```

EXECUTION COMMANDE SEDORIC INSTR

Rappel de la syntaxe
INSTR expression_alphanumérique_n°1,expression_alphanumérique_n°2,position

Recherche la prochaine occurence 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 variabe IN la position de l'occurence trouvée (IN = 1 si la chaîne n°2 commence au premier caractère de la chaîne n°1).

Bogue dans le manuel concernant ce que retourne cette commande: 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.

```
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 1ère chaîne
EC5E-  A5 F2      LDA F2      longueur de la première chaîne
EC60-  F0 1C      BEQ EC7E    la 1ère chaîne est vide, termine en EC7E (IN = 0)
EC62-  A6 F3      LDX F3      longueur de la deuxième chaîne
EC64-  F0 18      BEQ EC7E    la 2ème 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 1ère 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 1ère chaîne
EC7C-  C8         INY         et l'ajuste (rang 0 correspond au 1er caractère)
EC7D-  2C A0 00   BIT 00A0    continue en EC80
```

```
C7E-  A0 00      LDY 00      recherche infructueuse (où chaîne vide)
C80-  98         TYA         dans tous les cas, A reçoit la position
C81-  4C DB D7   JMP D7DB    mise à jour de la variable IN avec la valeur A
```

Caractères différents: reprend la comparaison

```
C84-  E6 F6      INC F6      vise le caractère suivant de la 1ère chaîne
C86-  A5 F6      LDA F6
C88-  C5 F2      CMP F2      teste si la fin de la 1ère chaîne est atteinte
C8A-  F0 F2      BEQ EC7E    si oui, recherche infructueuse, termine en EC7E
C8C-  D0 D4      BNE EC62    sinon, reboucle en EC62

C8E-  4C 23 DE   JMP DE23    "SYNTAX ERROR"

C91-  4C 20 DE   JMP DE20    "ILLEGAL QUANTITY ERROR"
```

EXECUTION 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 (en théorie car cette fonction est boguée, ne pas l'utiliser, voir plus loin).
r, en fait 1er 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!
ng nombre de caractères à saisir (1 à 255), paramètre obligatoire.
/A variable alphanumérique où placer la chaîne 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 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.
Contrairement à ce qui est indiqué dans le manuel (**bogue**), interdit de sortir avec les flèches de déplacement (mode par défaut).
Sortie automatique lorsque le curseur atteint la fin de la fenêtre. Sinon, par défaut, le curseur revient au début de la fenêtre.
"Justifie" le texte entré à l'écran (mais pas la variable) en remplaçant les caractères de remplissage par des espaces.
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 s/p XLINPU, qui est en fait la partie principale de LINPUT et qui altère en sortie les adresses suivantes:

```
2/13  adresse de la ligne du curseur TEXT
F/20  calcul de l'adresse de la ligne du curseur TEXT
3     flag numérique/alphanumérique (#00 si numérique et #FF si chaîne)
 /92  adresse de la chaîne pointée par TXTPTR
6/B7  pointe sur le descripteur de chaîne
8/B9  pointe sur le descripteur de chaîne
0     longueur de la chaîne saisie
1/D2  adresse de la chaîne saisie
2     nombre de caractères à saisir
3     indique quels paramètres ont été sélectionnés (E, S, C, J et K)
4     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
5     nombre de colonnes actives selon 026A (indicateur état console),
      puis à la fin constitue une réplique de F2
268   n° de la ligne du curseur (ordonnée y, de 1 à 27)
269   et 02F8 le n° de la colonne du curseur (abscisse x, de 0 à 39)
075   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 présente un grave problème de gestion du curseur. Après un LINPUT, le positionnement du curseur est complètement faussé, rendant impossible l'utilisation de certaines autres commandes, par exemple un PRINT@ ou un autre LINPUT@. Ceci apparait aussi lorsque la longueur de la chaîne demandée dépasse 38 caractères. Les facéties du curseur sont quasi imprévisibles et rendent impossible l'utilisation, à coup sûr, du paramètre "@x,y,". Ceci vient du fait qu'il ne suffit pas d'initaliser correctement les variables impliquées dans la gestion du curseur (comme le fait très bien LINPUT), il faut les valider par un affichage réel.

Voici la solution la plus simple, valable non seulement pour LINPUT, mais aussi dans tous les cas où il y a un problème de ce type. Elle consiste à repositionner soit-même le curseur avec un PRINT@x,y;CHR$(18)**;** (ne pas oublier le ";"). CHR$(18) correspond à CTRL/R et n'est pas utilisé par ORIC (il faut se rappeler **R** comme **R**epositionner).

Il fallait, en effet, trouver quelque chose à afficher qui ne modifie pas l'affichage, l'idéal étant PRINT@x,y;"" qui hélas ne valide pas les variables impliquées dans la gestion du curseur. Par chance, le PRINT@x,y;CHR$(18)**;** effectue tout ce qu'il faut: c'est à dire une véritable validation des paramètres x et y sans perturber l'affichage pour autant.

Voici un petit exemple de ce qu'il faut faire:

```
10 PRINT@10,10;CHR$(18); :' Positionne le curseur au début de la fenêtre
20 LINPUT 40;A$          :' Pour saisir 40 caractères à la position 10,10
30 PRINT@10,20;CHR$(18); :' Repositionne le curseur pour le PRINT@ suivant
40 PRINT@10,20;A$        :' Seul moyen d'afficher avec PRINT@ après un LINPUT dont la longueur de saisie excède 38 caractères (bogue à coup sûr)
```

Début de l'analyse de la commande LINPUT

EC94-	AA	TAX	1er 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	1er car 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 lit le caractère suivant à TXTPTR et le convertit en MAJUSCULE, ceci afin de positionner convenablement TXTPTR pour la routine DA22/ROM
ECA0-	20 40 D7	JSR D740	"CURSEUR OFF" (curseur caché = 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	
ECB5-	20 3E D7	JSR D73E	"CURSEUR ON" (curseur visible = vidéo inverse)
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 s/p de gestion du caractère à afficher et continue au s/p 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

C**D7**- 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
CDA- 20 82 D2 JSR D282 JSR D8CB/ROM Prend un entier dans ACC1 et le retourne dans X (nombre de caractères à saisir)
CDD- 8A TXA teste ce nombre de caractères (Z = 1 si nul)
CDE- F0 4B BEQ ED2B si X = 0, "ILLEGAL QUANTITY ERROR"
CE0- 86 F2 STX F2 sauve le nombre de caractères à saisir dans F2

Demande la variable alphanumérique requise

CE2- A9 3B LDA #3B A = ";" (marqueur situé devant la variable)
CE4- 20 2E D2 JSR D22E JSR D067/ROM demande ";" à TXTPTR, lit le caractère suivant en continuant au s/p CHARGET en E2 (qui saute les espaces), puis à l'interpréteur Sédoric en 0400, puis au s/p ECB9/ROM et finalement convertit ce caractère en MAJUSCULE (s/p D3A1/RAMOV). Au cours de ce périple, le sort de Y est assez indéfini, mais semble finir avec Y = 0
CE7- 84 F3 STY F3 supposons que Y = #00 (sauf contre indication!)
CE9- 20 2E ED JSR ED2E prendre adresse de la variable à TXTPTR (B8/B9)
CEC- 20 1B D2 JSR D21B SEC et JSR CF09/ROM vérifie si alphanumérique
CEF- 20 9E D3 JSR D39E XCRGOT relit le caractère à TXTPTR et le convertit en MAJUSCULE
CF2- F0 25 BEQ ED19 Si nul, fin des paramètres, continue à XLINPU

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

C**F4**- 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
CF7- 20 A1 D3 JSR D3A1 re-convertit en MAJUSCULE (pour être bien sûr!)
CFA- A2 04 LDX #04 X = 4 pour lire 5 paramètres dans une table
CFC- 86 F4 STX F4 sauve cet index dans F4 (seul bit b2 est à 1)
C**FE**- 06 F4 ASL F4 décale vers la gauche le bit qui est à 1
D00- 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 = #04), **J** (en CDBC avec X = #04), **C** (en CDBB avec X = #04) ou **S** (en CDBA avec X = #04)

128	**64**	**32**	**16**	**8**	**4**	**2**	**1**
0	**0**	**0**	**0**	**0**	**1**	**0**	**0**
7	**6**	**5**	**4**	**3**	**2**	**1**	**0**
S	**C**	**J**	**K**	**E**			

D03- F0 05 BEQ ED0A si oui, branche en ED0A
D05- CA DEX sinon, indexe la lettre précédente dans table
D06- 10 F6 BPL ECFE et reboucle en ECFE (suite de la recherche)
D08- 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
D0A- 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**
D0C- 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
D0E- C5 F3 CMP F3 compare le résultat avec F3
D10- 90 16 BCC ED28 si A < F3 branche vers "SYNTAX ERROR", ce qui se produit si un paramètre figure en double dans la commande!
D12- 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
D14- 20 98 D3 JSR D398 XCRGET lit le caractère suivant à TXTPTR et le convertit en MAJUSCULE
D17- D0 DB BNE ECF4 reboucle s'il reste des paramètres, sinon...
D19- 20 36 ED JSR ED36 XLINPU: appel de la routine LINPUT proprement dite
D1C- 20 8E EE JSR EE8E au retour, met la longueur et l'adresse de la chaîne dans la variable alphanumérique indiquée dans la ligne de commande
D1F- 68 PLA
D20- 8D 6A 02 STA 026A restaure le mode console Oric-1/Atmos initial
D23- A5 F4 LDA F4 copie n° du mode de sortie
D25- 4C D8 D7 JMP D7D8 dans la variable OM et retourne

D28- 4C 23 DE JMP DE23 "SYNTAX ERROR"

D2B- 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 LINPUT proprement dite

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. XLINPU copie cette longueur et cette adresse dans la variable alphanumérique à l'adresse indiquée en B8/B9 (voir en ED1C).

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 s/p "CURSEUR OFF"
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
ED47- CA DEX bogue: ça ne marche pas si la fenêtre compte
ED48- D0 FA BNE ED44 plus de 38 caractères (voir ci-dessous)
ED4A- 20 40 D7 JSR D740 s/p "CURSEUR OFF"
ED4D- A6 F2 LDX F2 nombre de caractères à saisir
ED4F- 20 69 EE JSR EE69 curseur au début du masque de saisie: affiche X fois "flèche gauche".

Re-bogue: lorsque la fenêtre compte plus de 38 caractères, le curseur ne revient pas au début de la fenêtre, mais au début de la ligne corresp. Ces 2 phénomènes (remplissage inapproprié et position finale inadéquate) se produisent aussi bien en mode 38 qu'en mode 40 colonnes. Par la suite, la fenêtre de saisie aura le bon nombre de caractères, mais commencera en début de ligne et non au point xy demandé et selon les cas, les derniers caractères de remplissage resteront affichés (version Oric-1 non testée). Lorsque la fenêtre compte moins de 38 caractères, tout semble être bon, mais en sortie de LINPUT, tout nouvel affichage formaté (un PRINT@ ou un LINPUT@) est perturbé.

Met en F5 le nombre de colonnes actives selon 026A
ED52- 20 3E D7 JSR D73E s/p "CURSEUR ON"
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 s/p "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, affiche "flèche gauche"
ED77- AD 75 C0 LDA C075 reprend le caractère de remplissage fenêtre

```
D7A-  20 2A D6   JSR D62A    XAFCAR: affiche ce caractère (= efface à gauche)
D7D-  A9 08      LDA #08     reprend suite normale avec A = "flèche gauche"

      Suite "1" de l'analyse du caractère saisi

D7F-  C9 0E      CMP #0E     est-ce CTRL/N? (effacement de ligne en cours)
D81-  D0 05      BNE ED88    continue en ED88 si ce n'est pas le cas
D83-  20 69 EE   JSR EE69    si oui, affiche X fois "flèche gauche"
D86-  F0 B4      BEQ ED3C    et reprend au début de la saisie

      Suite "2" de l'analyse du caractère saisi

D88-  C9 04      CMP #04     est-ce CTRL/D? (double hauteur)
D8A-  F0 04      BEQ ED90    si oui, caractère valide, continue en ED90
D8C-  C9 1A      CMP #1A     est-ce CTRL/Z? (ESC pour attribut écran)
D8E-  D0 05      BNE ED95    continue en ED95 si ce n'est pas le cas

      Affiche un caractère de CTRL valide et reboucle en saisie
D90-  20 2A D6   JSR D62A    "affiche" le caractère de contrôle valide
D93-  D0 CF      BNE ED64    et reboucle en ED64 (sans incrémenter X)

      Suite "3" de l'analyse du caractère saisi

D95-  C9 20      CMP #20     est-ce un autre code de contrôle? (A < #20)
D97-  90 14      BCC EDAD    si oui, continue en EDAD, sinon...

      Affiche caractère valide et reboucle si nécessaire
D99-  20 2A D6   JSR D62A    XAFCAR: affiche A (et déplace curseur à droite)
D9C-  E8         INX         compteur de caractères saisis et affichés
D9D-  E4 F2      CPX F2      le nombre requis est-il atteint?
D9F-  D0 C3      BNE ED64    sinon, reprend en ED64 (saisie au clavier)
DA1-  24 F3      BIT F3      si oui, teste si b6 est à 1 (C, sortie auto)
DA3-  50 A5      BVC ED4A    sinon, reboucle (curseur en début de fenêtre)

      Sortie automatique à la fin de la fenêtre
DA5-  CA         DEX         si oui, décrémente le nombre de caractères
DA6-  20 73 EE   JSR EE73    et affiche "flèche gauche"
DA9-  A0 06      LDY #06     Y = 6 pour sortie automatique en fin de fenêtre
DAB-  D0 57      BNE EE04    qui n'est pas nul... continue en EE04

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

DAD-  A0 00      LDY #00     Y à zéro indiquera "RETURN"
DAF-  C9 0D      CMP #0D     est-ce RETURN?
DB1-  F0 49      BEQ EDFC    si oui, continue en EDFC avec Y = 0
DB3-  C8         INY         sinon, incrémente Y (qui passe à 1)
DB4-  C9 1B      CMP #1B     est-ce ESC?
DB6-  F0 44      BEQ EDFC    si oui, continue en EDFC avec Y = 1
DB8-  C8         INY         sinon, incrémente Y (qui passe à 2)
DB9-  C9 08      CMP #08     est-ce "flèche gauche"?
DBB-  D0 09      BNE EDC6    sinon, continue en EDC6

      Gestion flèche gauche

DBD-  8A         TXA         si oui, teste si X est nul (aucun caractère n'a été saisi)
DBE-  F0 3C      BEQ EDFC    si oui, continue en EDFC avec Y = 2
DC0-  CA         DEX         s'il y a déjà des caractères, décrémente X
DC1-  20 73 EE   JSR EE73    affiche "flèche gauche"
DC4-  D0 9E      BNE ED64    et reboucle en ED64 (saisie au clavier)

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

DC6-  C8         INY         incrémente Y (qui passe à 3)
DC7-  C9 09      CMP #09     est-ce "flèche droite"?
DC9-  D0 0E      BNE EDD9    sinon, continue en EDD9

      Gestion flèche droite
DCB-  E8         INX         si oui, incrémente le nombre de caractères
DCC-  E4 F2      CPX F2      le nombre requis est-il atteint?
DCE-  F0 05      BEQ EDD5    si oui, continue en EDD5 (saisie terminée)
DD0-  20 76 EE   JSR EE76    sinon, affiche "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    s/p "CURSEUR OFF"
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    affiche "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 (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
```

```
E2D- 20 73 EE   JSR EE73    affiche "flèche gauche" (recule le curseur)
E30- 28         PLP         récupère les indicateurs 6502 dont Z
E31- D0 E7      BNE EE1A    reboucle tant qu'il en reste à copier
```

Retourne à la fin de la fenêtre

```
E33- A6 F2      LDX F2      nombre de caractères requis
E35- 20 76 EE   JSR EE76    affiche "flèche droite"
E38- CA         DEX         décrémente le nombre de caractères requis
E39- D0 FA      BNE EE35    reboucle tant que la fin n'est pas atteinte
```

Remplace les caractères de remplissage par des espaces

```
E3B- 06 F3      ASL F3      les bits indiquant les paramètres sélectionnés
E3D- 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
E3F- A4 F2      LDY F2      nombre de caractères requis
E41- 88         DEY         que l'on diminue
E42- B1 D1      LDA (D1),Y  lit caractère de chaîne en commençant par la fin
E44- CD 75 C0   CMP C075    est-ce un caractère de remplissage?
E47- D0 18      BNE EE61    sinon, "justification" finie, continue en EE61
E49- A9 20      LDA #20     si oui, remplace le caractère lu par un espace
E4B- 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)
E4D- 10 02      BPL EE51    si J non demandé, saute l'instruction suivante
E4F- 91 D1      STA (D1),Y  écrit un espace à la place du caractère de remplissage
E51- 50 06      BVC EE59    si K non demandé, saute les 2 instructions suivantes
E53- 20 2A D6   JSR D62A    XAFCAR affiche cet espace (et déplace à droite)
E56- 20 73 EE   JSR EE73    puis affiche "flèche gauche" (revient en place)
E59- 20 73 EE   JSR EE73    affiche "flèche gauche" (caractère précédent)
E5C- 98         TYA         teste s'il reste des caractères à examiner
E5D- D0 E2      BNE EE41    si oui, reboucle en EE41
E5F- 24 C8      BIT C8      sinon, "justification" finie, continue en EE61
```

Repositionne en fin de fenêtre

```
E60- C8         INY         point de rebouclage: caractère suivant
E61- 20 76 EE   JSR EE76    affiche "flèche droite"
E64- C4 F2      CPY F2      tous les caractères ont-ils été examinés?
E66- D0 F8      BNE EE60    sinon, reboucle en EE60
E68- 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.
```

Affiche X fois "flèche gauche"

```
E69- 8A         TXA         teste le nombre de caractères à saisir
E6A- F0 06      BEQ EE72    simple RTS si 0
E6C- 20 73 EE   JSR EE73    affiche "flèche gauche"
E6F- CA         DEX         décrémente le nombre de caractères à saisir
E70- D0 F7      BNE EE69    reboucle en EE69 tant qu'il en reste à saisir
E72- 60         RTS
```

Affiche une "flèche gauche"

```
E73- A9 08      LDA #08     A = "flèche gauche"
E75- 2C A9 09   BIT 09A9    et continue en EE7A
```

Affiche une "flèche droite"

```
E76- A9 09      LDA #09     A = "flèche droite"
E78- 24 68      BIT 68      et continue en EE7A
```

Passe la marge en mode 38 colonnes

```
E79- 68         PLA         reprend dans A la flèche qui est sur pile
```

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

```
E7A- 48         PHA         empile "flèche gauche" ou "flèche droite"
E7B- 20 2A D6   JSR D62A    affiche "flèche gauche" ou "flèche droite"
E7E- AD 6A 02   LDA 026A    mode console (ici la flèche de A est perdue)
E81- 29 20      AND #20     teste b5 (à 0, si mode normal 38 colonnes)
E83- D0 07      BNE EE8C    continue en EE8C si mode 40 colonnes
E85- AD 69 02   LDA 0269    n° de colonne (abscisse x)
E88- 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
```

Copie la longueur et l'adresse de la chaîne
"dans" la variable BASIC à l'adresse pointée en B8/B9

```
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
EE93- 91 B8     STA (B8),Y  écriture dans la variable BASIC
EE95- 88        DEY         vise octet précédent
EE96- 10 F8     BPL EE90    reboucle tant que Y est supérieur ou égal à 0
EE98- 60        RTS
```

EXECUTION COMMANDE SEDORIC USING

Rappel de la syntaxe

USING expression_numérique,expression_alphanum_modèle(,variable_alphanum)

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):

```
C4      signe du nombre
C5/CD   partie entière (9 caractères au maximum)
CE/D6   partie décimale (9 caractères au maximum)
D7      signe de l'exposant
D8/D9   exposant (2 caractères au maximum)
```

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"
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)
```

Met le signe en place dans la chaîne formatée

EC0-	AD 00 01	LDA 0100	lit le signe du nombre à formater
EC3-	C9 2D	CMP #2D	est-ce le signe "-"?
EC5-	F0 02	BEQ EEC9	si oui, continue en EEC9
EC7-	A9 2B	LDA #2B	sinon, prend le signe "+"
EC9-	85 C4	STA C4	met le signe en place dans la chaîne formatée

Initialise pour la partie entière

ECB-	86 F4	STX F4	initialise F4 = #00 = index dans la chaîne modèle
ECD-	86 F5	STX F5	initialise F5 = #00 = index dans la chaîne formatée
ECF-	A9 20	LDA #20	"espace"
ED1-	85 F6	STA F6	met par défaut un espace comme caractère de tête
ED3-	A0 01	LDY #01	
ED5-	84 F2	STY F2	nombre de caractères dans la partie entière
ED7-	88	DEY	Y repasse à zéro
ED8-	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

ED9-	A2 09	LDX #09	début de la partie décimale de la chaîne formatée

S/p commun: élaboration de la partie entière ou décimale

EDB-	C8	INY	vise le caractère suivant
EDC-	B9 00 01	LDA 0100,Y	lit un caractère de la chaîne non formatée
EDF-	F0 25	BEQ EF06	continue en EF06 s'il n'y en a plus
EE1-	C9 2E	CMP #2E	est-ce le "." décimal?
EE3-	F0 F4	BEQ EED9	si oui, reprend en EED9 pour élaborer la partie décimale
EE5-	C9 45	CMP #45	est-ce un "E"? (début de l'exposant)
EE7-	F0 0B	BEQ EEF4	si oui, continue en EEF4 pour élaborer l'exposant
EE9-	95 C5	STA C5,X	sinon, copie le caractère dans la partie entière (ou décimale, selon X) de la chaîne formatée
EEB-	E0 09	CPX #09	teste si X vise dans la partie décimale
EED-	B0 02	BCS EEF1	si oui, saute l'instruction suivante
EEF-	84 F2	STY F2	met à jour le compte de caractères de la partie entière
EF1-	E8	INX	vise l'octet suivant de la chaîne formatée
EF2-	D0 E7	BNE EEDB	reprise forcée en EEDB

Elabore l'exposant

EF4-	B9 01 01	LDA 0101,Y	lit le signe de l'exposant
EF7-	85 D7	STA D7	et l'écrit en D7 dans la chaîne formatée
EF9-	B9 02 01	LDA 0102,Y	lit le premier chiffre décimal de l'exposant
EFC-	AA	TAX	et le passe dans X
EFD-	B9 03 01	LDA 0103,Y	lit le 2ème chiffre de l'exposant, le garde dans A
F00-	F0 02	BEQ EF04	s'il n'y en a pas, saute l'instruction suivante
F02-	85 D9	STA D9	met en place le 2ème chiffre de l'exposant
F04-	86 D8	STX D8	met en place le 1er chiffre de l'exposant

Justifie à droite la partie entière

F06-	A6 F2	LDX F2	longueur de la partie entière de la chaîne formatée (soit 8 au maximum)
F08-	A0 08	LDY #08	vise le dernier caractère de la partie entière de la chaîne formatée
F0A-	B5 C4	LDA C4,X	lit un caractère de la partie entière de la chaîne formatée
F0C-	CA	DEX	décrémente le nombre de caractères de la partie entière de la chaîne formatée
F0D-	10 02	BPL EF11	saute l'instruction suivante tant qu'il reste des caractères de la partie entière de la chaîne formatée à écrire
F0F-	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
F11-	99 C5 00	STA 00C5,Y	justifie à droite la partie entère
F14-	88	DEY	vise l'emplacement précédent
F15-	10 F3	BPL EF0A	reboucle en EF0A pour déplacer si possible tous les caractères à droite
F17-	2C 84 F5	BIT F584	continue en EF1A

Point de rebouclage lors du formatage

F18-	84 F5	STX F5	met à jour l'index de la chaîne produite

Y a t-il un exposant à formater?

F1A-	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 et le convertit en MAJUSCULE
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    copie l'adresse et la longueur de la chaîne dans les data de la variable BASIC 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

```
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 formattée finale
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
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

```
F7C- 20 A7 EF   JSR EFA7    lit le nombre de caractères de la partie décimale de la chaîne modèle et le met en F3
F7F- A2 09      LDX #09     pour 9 caractères
F81- D0 10      BNE EF93    suite forcée en EF93 pour élaborer la partie décimale formatée
```

Suite de l'analyse de la chaîne modèle

```
F83- C9 25      CMP #25     est-ce un "%"?
F85- D0 32      BNE EFB9    sinon, suite de l'analyse en EFB9
```

Détermine le nombre de caractères de la partie entière

```
F87- 20 A7 EF   JSR EFA7    lit le nombre de caractères de la partie entière de la chaîne modèle et le met en F3
F8A- C5 F2      CMP F2      est-ce aussi long que ce que nous avons déjà?
F8C- 90 25      BCC EFB3    sinon, "ILLEGAL QUANTITY ERROR"
F8E- A9 09      LDA #09     longueur maximum
F90- E5 F3      SBC F3      longueur que doit avoir la partie entière
F92- AA         TAX         index où commencer dans la chaîne non formatée
```

Elabore la partie décimale de la chaîne formatée

```
F93- C6 F3      DEC F3      décrémente le compteur
F95- 10 03      BPL EF9A    saute l'instruction suivante pour continuer à élaborer la chaîne formatée
F97- 4C 18 EF   JMP EF18    pas de partie entière
F9A- B5 C5      LDA C5,X    lit un chiffre de la partie entière
F9C- 29 7F      AND #7F     en force le b7 à zéro
F9E- 99 00 C1   STA C100,Y  l'écrit dans la chaîne formatée en cours
FA1- C8         INY         vise le suivant (cible)
FA2- F0 12      BEQ EFB6    "STRING TOO LONG ERROR" si > 256 caractères
FA4- E8         INX         vise le suivant (source)
FA5- 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

```
FA7- 20 2B F0   JSR F02B    lit un caractère dans la chaîne modèle
FAA- E9 30      SBC #30     convertit le code ASCII en chiffre de 0 à 9
FAC- 85 F3      STA F3      et sauve ce nombre en F3
FAE- C9 0A      CMP #0A     teste si < 10
FB0- B0 01      BCS EFB3    sinon, "ILLEGAL QUANTITY ERROR"
FB2- 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 maintemant 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	

EXECUTION COMMANDE SEDORIC LUSING

Rappel de la syntaxe

LUSING expression_numérique,expression_alphanum_modèle(,variable_alphanum)

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
F039-	20 99 EE	JSR EE99	USING
F03C-	4C D6 E7	JMP E7D6	PR OFF

Convertit AN en radians, résultat en BFE0/BFE4, place -PI/2 en BFEA/BFEE
(s/p appelé par LINE et BOX)

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	
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	
F06D-	20 C2 D2	JSR D2C2	JSR DEAD/ROM recopie les 5 octets de ACC1 de l'adresse XY à l'adresse XY + 4
F070-	AD 1F 02	LDA 021F	teste si mode HIRES (il est bien temps!)
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"

078- 60 RTS

EXECUTION 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 dégrés 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/RAMOV. Attention donc aux routines en LM qui pourraient utiliser cette zone!

079- 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
07C- 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)
07F- A2 E5 LDX #E5 pour mettre le résultat en BFE5/BFE9
081- A0 BF LDY #BF
083- 20 C2 D2 JSR D2C2 JSR DEAD/ROM recopie les 5 octets de ACC1 de l'adresse XY à l'adresse XY + 4
086- A2 00 LDX #00 index pour les paramètres DRAW
088- 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)
08A- A9 E0 LDA #E0
08C- A0 BF LDY #BF AY pointe sur la valeur de AN en radians
08E- 20 BA D2 JSR D2BA JSR DE7B/ROM place dans ACC1 (floating point accumulator) la valeur pointée par AY
091- A6 F2 LDX F2 index pour les paramètres DRAW
093- F0 09 BEQ F09E suite forcée en F09E si c'est le premier

Calcule le deuxième paramètre de DRAW

095- 20 F2 D2 JSR D2F2 JSR E392/ROM calcule SIN(AN) dans ACC1
098- 20 DA D2 JSR D2DA JSR E271/ROM changement de signe (echelle y inversée)
09B- 4C A1 F0 JMP F0A1 saute l'instruction suivante

Calcule le premier paramètre de DRAW

09E- 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

0A1- A9 E5 LDA #E5
0A3- A0 BF LDY #BF AY pointe sur la longueur de la ligne
0A5- 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
0A8- 20 8A D2 JSR D28A JSR D926/ROM convertit le nombre présent dans ACC1 en entier signé dans YA
0AB- 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 1ère 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 s/p 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 1er 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"
```

EXECUTION 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 1er coté selon la position courante du curseur HIRES, l'angle AN courant, la longueur indiquée pour le 1er coté, fait pivoter AN de -90°, trace le 2ème coté selon la longueur indiquée pour le 2ème coté, fait pivoter AN de -90°, trace le 3ème coté selon la longueur indiquée pour le 1er coté, fait pivoter AN de -90°, trace le dernier coté selon la longueur indiquée pour le 2ème 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/RAMOV. Attention donc aux routines en LM 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
```

```
0F4-  85 F6      STA F6      pour commencer avec le 1er paramètre de BOX
0F6-  A6 F6      LDX F6      copié dans X pour usage prochain
0F8-  8A         TXA
0F9-  49 01      EOR #01     bascule le b0 de F6 (flag pour l'autre paramètre)
0FB-  85 F6      STA F6      et le remet en place
0FD-  B4 F3      LDY F3,X    lit la longueur du 1er ou du 2ème coté
0FF-  A9 00      LDA #00     force à zéro le HH de cette longueur
101-  20 CA D2   JSR D2CA    JSR DF40/ROM transfère un nombre non signé YA dans ACC1 (floating point
                             accumulator)
104-  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°

```
107-  A9 E0      LDA #E0
109-  A0 BF      LDY #BF     valeur actuelle de AN
10B-  20 BA D2   JSR D2BA    JSR DE7B/ROM place dans ACC1 (floating point accumulator) la valeur pointée par AY
10E-  A9 EA      LDA #EA
110-  A0 BF      LDY #BF     valeur de -90°
112-  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
115-  A2 E0      LDX #E0
117-  A0 BF      LDY #BF     nouvelle valeur de AN
119-  20 C2 D2   JSR D2C2    JSR DEAD/ROM recopie les 5 octets de ACC1 de l'adresse XY à l'adresse XY + 4
11C-  C6 F5      DEC F5      compteur de cotés restant à dessiner
11E-  D0 D6      BNE F0F6    reprend en F0F6 s'il en reste à tracer
120-  60         RTS
```

COMMANDES SEDORIC FAISANT APPEL A UNE BANQUE EXTERNE

Les entrées des commandes Sédoric 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	ADR
BACKUP	2	#47	#03	C403
CHANGE	3	#4C	#06	C406
COPY	4	#51	#03	C403
DELETE	1	#42	#06	C406
DKEY	5	#56	#18	C418
DNAME	5	#56	#06	C406
DNUM	5	#56	#12	C412
DSYS	5	#56	#15	C415
DTRACK	5	#56	#09	C409
INIST	5	#56	#0F	C40F
INIT	6	#5B	(pas de valeur Y pour INIT)	
MERGE	3	#4C	#09	C409
MOVE	1	#42	#09	C409
RENUM	1	#42	#03	C403
SEEK	3	#4C	#03	C403
SYS	5	#56	#03	C403
TRACK	5	#56	#0C	C40C
VUSER	5	#56	#1B	C41B

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 "ADR" 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 1 commence donc au secteur #3D + 5 = #42, c'est à dire au 66ème 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 (5ème piste) car 66 = (16 x 5) + 2.

Sauf pour INIT, le programme se poursuit avec le sous-programme (s/p) 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 RAMOV de C400 à C7FF), puis suite en F16B.

EXECUTION 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

EXECUTION 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

EXECUTION 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

EXECUTION 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

EXECUTION 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

EXECUTION 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

EXECUTION COMMANDE SEDORIC MOVE

```
136- A2 42      LDX #42     position de la banque sur la disquette MASTER
 38- 2C A2 56   BIT 56A2    -> continue en #F13E
```

EXECUTION COMMANDE SEDORIC DTRACK

```
139- A2 56      LDX #56     position de la banque sur la disquette MASTER
 3B- 2C A2 4C   BIT 4CA2    -> continue en #F13E
```

EXECUTION COMMANDE SEDORIC MERGE

```
13C- A2 4C      LDX #4C     position de la banque sur la disquette MASTER
13E- A0 09      LDY #09     octet de poids faible de l'adresse d'exécution
 40- D0 1C      BNE F15E    -> branchement forcé en #F15E
```

EXECUTION COMMANDE SEDORIC DELETE

```
142- A2 42      LDX #42     position de la banque sur la disquette MASTER
 44- 2C A2 56   BIT 56A2    -> continue en #F14A
```

EXECUTION COMMANDE SEDORIC DNAME

```
145- A2 56      LDX #56     position de la banque sur la disquette MASTER
 47- 2C A2 4C   BIT 4CA2    -> continue en #F14A
```

EXECUTION COMMANDE SEDORIC CHANGE

```
148- A2 4C      LDX #4C     position de la banque sur la disquette MASTER
14A- A0 06      LDY #06     octet de poids faible de l'adresse d'exécution
 4C- D0 10      BNE F15E    -> branchement forcé en #F15E
```

EXECUTION COMMANDE SEDORIC RENUM

```
14E- A2 42      LDX #42     position de la banque sur la disquette MASTER
 50- 2C A2 47   BIT 47A2    -> continue en #F153
```

EXECUTION COMMANDE SEDORIC BACKUP

```
151- A2 47      LDX #47     position de la banque sur la disquette MASTER
 53- 2C A2 4C   BIT 4CA2    -> continue en #F15C
```

EXECUTION COMMANDE SEDORIC SEEK

```
154- A2 4C      LDX #4C     position de la banque sur la disquette MASTER
 56- 2C A2 51   BIT 51A2    -> continue en #F15C
```

EXECUTION COMMANDE SEDORIC COPY

```
157- A2 51      LDX #51     position de la banque sur la disquette MASTER
 59- 2C A2 56   BIT 56A2    -> continue en #F15C
```

EXECUTION COMMANDE SEDORIC SYS

```
15A- A2 56      LDX #56     position de la banque sur la disquette MASTER
 5C- A0 03      LDY #03     octet de poids faible de l'adresse d'exécution
```

Entrée commune des commandes sur banques externes (sauf INIT)

```
15E- A9 C4      LDA #C4     empile l'adresse d'entrée de la commande
 60- 48         PHA         dans la banque située de C400 à C7FF
 61- 98         TYA         cette adresse est composée de C4 dans tous les cas
 62- 48         PHA         et de Y (différent selon commande visée)
                            X contient toujours le numéro de banque de la commande (#42, #47 etc ..)
 63- EC 15 C0   CPX C015    compare banque voulue et n° de banque active
 66- F0 51      BEQ F1B9    si identiques, continue en F1B9
 68- 2C A2 5B   BIT 5BA2    si différentes, continue en F16B
```

EXECUTION COMMANDE SEDORIC INIT (sur banque externe 6)

```
169- A2 5B      LDX #5B     n° de banque pour INIT
```

Mise en place de la banque voulue

Le s/p F16B/F1B8 demande une disquette master, teste s'il faut abandonner (ESC) ou continuer (RETURN). Dans ce dernier cas, la bitmap est chargée, le s/p 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 corresp à la banque voulue.

```
16B- 8A         TXA         empile aussi X le n° de
 6C- 48         PHA         la banque qui sera chargée
 6D- A2 0C      LDX #0C     indexe "INSERT MASTER "
```

F16F-	20 6C D3	JSR D36C	affiche X+1ème 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 "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 secteur de bitmap, vérifie format
F192-	AD 07 C2	LDA C207	8ème 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	11ème 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 nème 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 1er 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
F1B2-	A9 C4	LDA #C4	du secteur Y de la piste indiquée en C001 PISTE
F1B4-	20 E5 F1	JSR F1E5	et les copie de C400 à C7FF (zone des banques)
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 s/p F1BA/F1E4 met à jour C016, flag "la banque a été chargée", initialise C00D/C00E (EXTER, adresse des messages d'erreur externe) 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 RAMOV 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 message erreur extern)
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 et le convertit en MAJUSCULE

Cas de INIT

Le s/p F1D2/F1E4 charge 95 secteurs pris à partir du secteur 1 de la piste 0 et les copie en RAM de 3000 à 8EFF puis effectue le fameux JMP C404 dont nous avons déjà parlé (entrée pour exécution de INIT dans la banque 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. Il faudrait modifier la syntaxe de INIT en ajoutant une option ",M" pour optimiser la recopie des secteurs de la disquette MASTER en mémoire (l'absence de ce ",M" conduisant par défaut à un formatage en SLAVE). De plus, il n'y a pas besoin de charger les 95 premiers secteurs, mais seulement les 94 premiers (bogue mineure).

```
1D2- A2 5F      LDX #5F     chargera 95 secteurs à partir de secteur 1 de piste 0
 D4- A9 30      LDA #30     et les copiera en RAM de l'adresse 3000 à l'adresse 8EFF
 D6- A0 00      LDY #00     la prochaine piste à copier sera
 D8- 8C 01 C0   STY C001    la première (piste n°0) et le
 DB- C8         INY         prochain secteur à copier sera le secteur n°1
 DC- 20 E5 F1   JSR F1E5    et hop! c'est parti mon kiki
 DF- 20 9E D3   JSR D39E    XCRGOT relit le caractère à TXTPTR et le convertit en MAJUSCULE
 E2- 4C 04 C4   JMP C404    entrée pour exécution de INIT dans la banque 6
```

Charge X secteurs pris à partir du secteur Y de la piste indiquée en C001 (PISTE) et les copie à partir de la page A

C'est bien beau, mais cela ne tient pas compte du fait qu'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 bien inutile. Il faudrait modifier la syntaxe de INIT en ajoutant ",M" pour optimiser la recopie des secteurs de la disquette Master en mémoire (l'absence de ce ",M" conduisant par défaut à un formatage en Slave).

```
1E5- 86 F5      STX F5      nombre de secteurs à charger
 E7- 8D 04 C0   STA C004    adresse de chargement du prochain secteur:
 EA- A9 00      LDA #00     (adresse RWBUF est formée de HH=A et LL=00)
 EC- 8D 03 C0   STA C003    soit #C400 pour une banque par exemple
 EF- 78         SEI         interdit les interruptions
 F0- 8C 02 C0   STY C002    SECTEUR (n° du prochain secteur à charger)
 F3- 20 73 DA   JSR DA73    XPRSEC lit un secteur selon DRIVE, PISTE, SECTEUR et RWBUF
 F6- EE 04 C0   INC C004    page suivante (adresse prochain chargement)
 F9- AC 02 C0   LDY C002    compare n° du secteur chargé avec
 FC- CC 4B C0   CPY C04B    nombre de secteurs par piste
 FF- 90 05      BCC F206    si pas encore égal augmente seulement n° de secteur
201- EE 01 C0   INC C001    si dernier lu augmente PISTE (n° de piste)
204- A0 00      LDY #00     et remet le n° de secteur à zéro
206- C8         INY         secteur suivant (n°1 si début de piste)
207- C6 F5      DEC F5      nombre de secteurs restant à charger
209- D0 E5      BNE F1F0    reboucle s'il en reste à charger
20B- 58         CLI         restaure les interruptions
20C- 60         RTS         et retourne

20D- 4C E0 E0   JMP E0E0    "FILE TYPE MISMATCH ERROR"
```

EXECUTION COMMANDE SEDORIC WINDOW

Rappel de la syntaxe

WINDOW (nom_de_fichier_non_ambigu)

La commande WINDOW affiche à l'écran le masque de saisie NF (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. Lorsque NF 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 Sédoric, 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 Sédoric, c'est très simple: la 1ère colonne porte le n°0 (#00) et la dernière le n°39 (#27); la 1ère 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 1ère 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 3ème. A ce moment là, le curseur bondira directement à la 3ème case.

Au contraire, avec l'Atmos, après un CTRL/L ou un retour à la ligne, le curseur se place en attente à la 3ème colonne (et 0269 contiendra la valeur #02), le caractère y sera affiché et le curseur passera à la 4ème 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 1ère colonne n'a pas de n°, la 2ème colonne a le n°0 (#00) et la dernière porte le n° 38 (#26); la 1ère 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 1ère case accessible avec le curseur sont donc (1,0). Curiosité: PLOT permet d'accéder à la 2ème case de la marge (PLOT 0,0,...), mais pas à la première!

Les coordonnées colonne/ligne BASIC de l'Atmos sont plus raisonnables: la 1ère colonne a le n°0 (#00) et la dernière le n° 39 (#27); la 1ère 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 1ère case accessible avec le curseur sont donc (2,0). PLOT permet enfin d'accéder à la 1ère 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 NF)
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	recherche "?" dans BUFNOM revient avec C = 1 si pas trouvé ou "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).

Ni la ligne service, ni les 1ère (n°1) et dernière (n°27) lignes de l'écran accessible ne sont utilisées par le masque

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'en apercevoir, la RAMOV 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

```
243-  08          PHP
```

Copie du masque dans l'écran TEXT

```
244-  20 DE DF    JSR DFDE      vérifie si bien mode TEXT
247-  A9 B8       LDA #B8
249-  A0 BB       LDY #BB       (#BBBB = #BBD0 - #18)
24B-  85 F2       STA F2        prépare un move de #03E8 octets
24D-  84 F3       STY F3        (4 fois #100 moins #18)
24F-  A9 E8       LDA #E8       de C400 à C7E7 (masque en RAMOV)
251-  A0 C3       LDY #C3       vers BBD0 à BFB7 (dans l'écran TEXT)
253-  85 F4       STA F4
255-  84 F5       STY F5
257-  A2 04       LDX #04       pour 4 pages (une page = une zone de 256 octets)
259-  A0 18       LDY #18       en commençant au 18ème octet de la 1ère page
25B-  B1 F4       LDA (F4),Y    lit octet dans le masque en RAMOV
25D-  91 F2       STA (F2),Y    écrit cet octet dans l'écran texte
25F-  C8          INY           octet suivant
260-  D0 F9       BNE F25B      reboucle tant que la page n'est pas finie (attend que Y = 0 et donc que Z = 1)
262-  E6 F3       INC F3        indexe page suivante (incrémente HH adresse écriture)
264-  E6 F5       INC F5        indexe page suivante (incrémente HH adresse lecture)
266-  CA          DEX           décrémente le nombre de page restant à copier
267-  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 1er 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.

```
269-  20 27 F3    JSR F327      remplit les champs avec les chaînes de WI$
26C-  20 09 F3    JSR F309      cherche la première case du premier champ présent dans le masque et sort de WINDOW si
ne trouve pas de champ
```

Saisie au clavier les données d'un champ (curseur visible)

```
26F-  20 3E D7    JSR D73E      champ trouvé, force l'affichage du curseur
272-  58          CLI           autorise les interruptions (pour saisir touche)
273-  20 45 D8    JSR D845      prendre un caractère au clavier
276-  10 FB       BPL F273      reboucle tant que b7 = 0 (attente touche)
278-  78          SEI           interdit les interruptions (touche saisie)
279-  C9 03       CMP #03       est-ce un CTRL/C? (CTRL/C = #03)
27B-  F0 68       BEQ F2E5      si oui, suite en F2E5 (sauvegarde des données)
27D-  C9 7F       CMP #7F       sinon, est-ce la touche DEL?
27F-  D0 15       BNE F296      si ce n'est pas le cas, continue en F296
281-  A9 08       LDA #08       si c'est DEL, A = CTRL/H (flèche gauche, sera "affichée" par la commande suivante pour
tester si ce DEL est possible)
283-  20 EC F2    JSR F2EC      Gestion déplacements du curseur dans le masque
286-  30 E7       BMI F26F      reprend saisie si "bute" contre début de masque
288-  20 CA F2    JSR F2CA      si ce n'est pas le cas, lit dans le masque l'octet corresp au curseur (donc à gauche de la
position initiale)
28B-  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
28D-  A9 09       LDA #09       si case de champ, A = CTRL/I (flèche droite, sera "affichée" par la commande suivante pour
revenir à la case initiale)
28F-  20 2A D6    JSR D62A      XAFCAR affiche ce caractère (retour case initiale)
292-  A9 7F       LDA #7F       A = DEL (reprend la valeur initiale de DEL)
294-  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

```
296-  C9 20       CMP #20       est-ce un code de CTRL? (A = code ASCII < #20)
298-  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)

```
29A-  20 2A D6    JSR D62A      XAFCAR affiche ce caractère ASCII
29D-  A9 08       LDA #08       affiche A = CTRL/H (flèche gauche, pour
29F-  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 déboguage. La détection d'un DEL illégal a entrainé 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 gènante. Pour contrecarrer cette anomalie, une flèche gauche a été intercallée 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 corresp 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 corresp 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 en C400 de l'octet corresp 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 s/p "CURSEUR OFF"
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) corresp 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 corresp 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.

```
2EE- 20 2A D6   JSR D62A    XAFCAR affiche le caractère ASCII contenu dans A
2F1- AC 68 02   LDY 0268    Y = n° de la ligne du curseur TEXT
2F4- C0 01      CPY #01     est-ce la ligne n°1? (donc hors masque)
2F6- F0 04      BEQ F2FC    si oui, continue en F2FC (avec C = 1)
2F8- C0 1B      CPY #1B     est-ce la ligne n°27? (donc hors masque)
2FA- D0 0A      BNE F306    sinon, continue en F306 (avec C = 0)
2FC- 24 F2      BIT F2      si hors masque (C = 1), teste flag "masque"
2FE- 30 08      BMI F308    si b7 à 1, simple RTS (rebouclage déjà fait)
300- 66 F2      ROR F2      sinon, C -> b7 donc flag F2 et N passent à 1
302- 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).
304- D0 E8      BNE F2EE    et rebouclage forcé pour mouvement inverse
306- 24 F2      BIT F2      positionne N selon b7 de F2 et retourne
308- 60         RTS
```

Cherche première case de champ présente dans le masque en C400/C7E7

```
309- A9 1E      LDA #1E     caractère de contrôle pour placer curseur en (0,1) c'est à dire au début de l'écran (1ère case de
1ère ligne) (fonction HOME)
30B- 20 2A D6   JSR D62A    XAFCAR "affiche" ce caractère ASCII (CTRL/^)
30E- 20 06 D2   JSR D206    JSR CBF0/ROM (CRLF) curseur au début du masque
311- 20 CA F2   JSR F2CA    lit dans le masque l'octet corresp au curseur
314- F0 0E      BEQ F324    simple RTS si c'est #7F (indicateur de champ)
316- A9 09      LDA #09     sinon, A = curseur vers la droite (CTRL/I)
318- 20 EC F2   JSR F2EC    gestion des déplacements du curseur dans le masque
31B- 10 F4      BPL F311    reboucle si curseur est toujours dans le masque
31D- 68         PLA         si fin de masque atteinte sans trouver de #7F,
31E- 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
31F- 28         PLP         récupère les indicateurs 6502
320- 68         PLA         récupère A
321- 8D 6A 02   STA 026A    et le remet en 026A (flags d'état console)
324- 60         RTS
```

Copie de WI$ dans SCRN et de SCRN dans WI$

Il s'agit de 2 routines complexes qui s'entremèlent 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 corresp). 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, nous vous conseillons 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 1er 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 1ère 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 1er 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 2ème 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 3ème 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 1er 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 Sédoric les chaînes ne peuvent avoir plus de 255 caractères or le 1er 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)

377-	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
379-	AC 69 02	LDY 0269	Y = n° de colonne où est le curseur TEXT
37C-	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.

37E-	B0 11	BCS **F391**	suite forcée en F391 car C = 1 (chaîne < champ)

SCRN->WI$: copie chaîne de l'écran dans BUF1 (avant d'aller dans WI$)

380-	AC 69 02	LDY 0269	Y = n° de colonne où est le curseur TEXT
383-	B1 12	LDA (12),Y	lit le caractère sous le curseur TEXT
385-	C9 7F	CMP #7F	est-ce #7F? (carré couleur INK utilisé par DEL)
387-	D0 02	BNE F38B	sinon, saute l'instruction suivante
389-	A9 20	LDA #20	remplace #7F par un espace
38B-	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$
38E-	E8	INX	vise le prochain caractère à copier
38F-	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.

391-	A9 09	LDA #09	A = curseur à droite (CTRL/I) (case suivante)

Suite commune pour tous (SCRN->WI$ et WI$->SCRN)

393-	20 2A D6	JSR D62A	XAFCAR affiche le caractère ASCII contenu dans A
396-	20 CA F2	JSR F2CA	lit dans le masque l'octet corresp au curseur, au retour Z = 1 si une case de champ a été trouvée
399-	F0 CC	BEQ F367	reboucle si #7F (lit caractère suivant de la chaîne)
39B-	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$)
39E-	30 1C	BMI F3BC	si ce n'est pas le cas, continue en F3BC

Suite pour SCRN->WI$: recopie de BUF1 dans WI$

3A0-	86 F2	STX F2	sauve X dans F2
3A2-	8A	TXA	et dans A (rappel: X est le pointeur de chaîne)
3A3-	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
3A6-	A0 00	LDY #00	prépare index pour copie de BUF1 -> chaîne
3A8-	B9 00 C1	LDA C100,Y	lit octet n°Y dans BUF1
3AB-	91 D1	STA (D1),Y	écrit cet octet dans la chaîne réservée
3AD-	C8	INY	octet suivant
3AE-	C4 F2	CPY F2	compare Y et valeur dans F2 (longueur chaîne)
3B0-	D0 F6	BNE F3A8	reboucle tant que pas fini
3B2-	A0 02	LDY #02	prépare Y pour copier 3 octets (longueur et adresse de la chaîne)
3B4-	B9 D0 00	LDA 00D0,Y	lit octet du descripteur de chaîne (longueur et adresse)
3B7-	91 B6	STA (B6),Y	et le copie en B6/B7/B8
3B9-	88	DEY	pour l'octet précédent
3BA-	10 F8	BPL F3B4	reboucle tant que Y n'est pas négatif

Suite pour tous (SCRN->WI$ et WI$->SCRN): recherche le champ suivant

3BC-	A9 09	LDA #09	A = curseur vers la droite (CTRL/I)
3BE-	20 EC F2	JSR F2EC	Gestion déplacements du curseur dans le masque
3C1-	30 0B	BMI F3CE	simple RTS si N = 1 (fin de masque atteinte)
3C3-	20 CA F2	JSR F2CA	lit dans le masque l'octet corresp au curseur
3C6-	D0 F4	BNE F3BC	reboucle en F3BC tant que Z = 0 (cherche champ)
3C8-	4C 3C F3	JMP F33C	champ trouvé, reboucle en F33C pour chercher l'adresse de la chaîne suivante dans le tableau WI$

3CB-	4C 77 E9	JMP E977	"STRING TOO LONG ERROR"

F3CE- 60 RTS

*** GESTION DE FICHIERS ***

Remarques préliminaires

Trois types de fichiers de data peuvent être utilisés sous Sédoric: 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):

- 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 1er 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 NF cible sans homologue dans le NF source longueur de l'enregistrement (nombre de caractères restant à afficher)
F2/F3	adresse de la paire d'octets corresp au NL dans la "Table NL" (cette paire d'octet est l'offset du début du "Channel Buffer" du fichier ouvert corresp, 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 Sédoric: 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 s/p 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	NFA "Source"	pour COPY* (drive, nom et extension)
C09D/C0A9	NFA "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

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:

"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. Nous parlons 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!

"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 corresp n'est pas ouvert. Cette paire d'octets représente l'offset nécessaire pour atteindre le début du "Channel Buffer" corresp à ce NL.

"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.

"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 corresp):

#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 1er 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 (NF 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 1er 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 (quel 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, corresp 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
les tableaux)	"**Entête**" 8 octets de #00 à #07: **FI** Lien du tableau suivant Offset du "Field Buffer" Nombre total de champs
+ #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)
ou + #88	"**General Buffer**" 512 octets de #88 à #287: 2 pages de tampon
ı + #288 ı ı	1er "**Channel Buffer**" de taille variable: pointeurs et flags (7 octets #00/#06) ligne de catalogue (16 octets #07/#16) "**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)
	2ème "Channel Buffer" (dans l'ordre des ouvertures)
	3ème "Channel Buffer" etc...
t longueur s au début de **FI**	"**Field Buffer**" 1 ou plusieurs blocs de 100 octets: 10 "entrées" de 10 octets chacune
ıdiqué par u début de **FI**	Suite de la zone des tableaux (tableaux proprement dits)

[1] 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:

/01 adresse réelle du début du "Channel Buffer" du fichier courant

/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**"

/05 idem + #117 = adresse du début du "Descriptor Buffer"

/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 1ère 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 2ème 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 1er 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 1er 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 Sédoric 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 (1ère 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 corresp à 0A dans la
"Table NL"puis calcule l'adresse AX et F2/F3 corresp à cet offset

(s/p F3CF-F3F2, appelé par la commande CLOSE et par les s/p F4A8 et F4E6)

Rappel: ce pointeur vise l'offset du début du "Channel Buffer" corresp 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 1er octet de la paire LLHH corresp au NL 63
F3D4- D0 0B BNE F3E1 suite forcée en F3E1 pour calculer l'adresse corresp, retournera finalement avec Y = #00 (voir OPEN pour explication)

Lit l'offset du début du "Field Buffer", puis calcule l'adresse corresp 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 corresp 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 corresp, retournera finalement avec Y = #00 (voir OPEN pour explication)

Calcule l'adresse AX et F2/F3 corresp 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 corresp 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
(s/p 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 1er 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")

```
414-  98          TYA
415-  91 9E       STA (9E),Y     force à zéro les 256 premiers octets de FI situés
417-  C8          INY            à partir de l'adresse présente en 9E/9F
418-  D0 FB       BNE F415       (entre autres, la "Table NL")
41A-  A0 05       LDY #05
41C-  B9 25 CD    LDA CD25,Y     copie les 6 octets présents en CD25/CD2A
41F-  91 9E       STA (9E),Y     à cette adresse, c'est à dire initialise le début
421-  88          DEY            de FI avec C6 C9 88 02 88 02
422-  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)
424-  60          RTS            et retourne
```

Extension de "**FI**" par insertion de AF2 octets au point d'offset XY
(s/p F425-F472, appelé par les s/p 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 (corresp 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 s/p 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 s/p 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.

```
425-  48          PHA            empile A (LL de l'augmentation de taille)
426-  84 F3       STY F3         garde Y dans F3 (HH de l'offset du point d'insertion)
428-  86 F9       STX F9         et X dans F9 (LL de l'offset du point d'insertion)
42A-  18          CLC            C = 0 représente une retenue pour une soustraction
42B-  A0 86       LDY #86        vise l'octet n°134, c'est à dire le 1er 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"
42D-  B1 9E       LDA (9E),Y     lors de la comparaison qui suit, C restera à 0
42F-  C5 F9       CMP F9         (retenue) si l'octet lu (1er 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)
431-  C8          INY            vise octet suivant, c'est à dire le 2ème d'une paire
432-  B1 9E       LDA (9E),Y     de même lors de cette soustraction, C restera à 0
434-  E5 F3       SBC F3         si l'octet lu (2ème octet d'une paire) est inférieur au HH de l'offset de la fin de la "Zone
Buffer"), sinon C passera à 1
436-  90 0F       BCC F447       continue en F447 si paire d'octet < XY (pas de mise à jour, la zone corresp est en dessous du
point d'insertion et ne sera pas déplacée)
438-  88          DEY            sinon, vise à nouveau le 1er octet de la paire
439-  18          CLC            prépare une addition
43A-  68          PLA            récupère une copie de A
43B-  48          PHA            (LL de l'augmentation de taille)
43C-  71 9E       ADC (9E),Y     calcule A = A + 1er octet de la paire
43E-  91 9E       STA (9E),Y     réécrit la nouvelle valeur dans FI
440-  C8          INY
441-  B1 9E       LDA (9E),Y     calcule A = F2 + 2ème octet de la paire
443-  65 F2       ADC F2
445-  91 9E       STA (9E),Y     réécrit la nouvelle valeur dans FI. Calcule ainsi: offset = offset + amplitude du décalage
vers le haut.
447-  88          DEY            vise à nouveau le 1er octet de la paire
448-  88          DEY
449-  88          DEY            vise le 1er octet de la paire précédente
44A-  D0 E1       BNE F42D       reboucle en F42D tant qu'il n'atteint pas la 1ère 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

```
44C-  8A          TXA            en entrée X = LL et
44D-  65 9E       ADC 9E         F3 = HH de l'offset du point d'insertion
44F-  AA          TAX            calcule XY (adresse de la zone à décaler)
450-  A5 F3       LDA F3         = XF3 (offset du point d'insertion)
452-  65 9F       ADC 9F         + adresse de début de FI
454-  A8          TAY
455-  68          PLA            récupère A (LL de l'augmentation de taille)
```

Décale à partir de l'adresse XY, de AF2 octets vers le haut
alcule 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 1er 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 1er 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 en A0/A1 (haut des tableaux))

Vérifie l'existence de **FI**, la validité du NL présent dans A et si le fichier est bien déjà ouvert
(s/p 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 corresp 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 (2ème 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 et le convertit en MAJUSCULE 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" corresp 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 (s/p F4A8-F4DB, appelé par les commandes &, >, BUID, FIELD, LSET, PUT, REWIND, RSET et TAKE).

4A8-	A9 88	LDA #88	AY = #0088 (136 décimal, soit
4AA-	A0 00	LDY #00	en-tête 8 + "Table NL" 64 x 2)
4AC-	20 E3 F3	JSR F3E3	calcule l'adresse AX (avec copie dans F2/F3) corresp à l'offset AY dans **FI** et retourne avec Y = #00
4AF-	85 06	STA 06	résultat qui est stocké en 06/07 (c'est l'adresse
4B1-	86 07	STX 07	du début du "General Buffer")

Calcule Y qui vise dans la "Table NL" la paire LLHH (offset du "Channel Buffer") corresp au NL

4B3-	A5 0A	LDA 0A	NL
4B5-	0A	ASL	multiplié par 2
4B6-	69 08	ADC #08	plus 8, le résultat est copié dans Y
4B8-	A8	TAY	

Place l'adresse du début du "Channel Buffer" corresp 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

4B9-	20 D8 F3	JSR F3D8	lit dans la "Table NL" l'offset visant le début du "Channel Buffer" corresp au NL, calcule l'adresse AX corresp à cet offset et retourne avec Y = #00
4BC-	85 00	STA 00	copie cette adresse en 00/01
4BE-	18	CLC	(début du "Channel Buffer")
4BF-	69 17	ADC #17	
4C1-	85 02	STA 02	copie cette adresse + #17 en 02/03
4C3-	85 04	STA 04	(début du buffer proprement dit
4C5-	8A	TXA	ou "Channel's own Data Buffer")
4C6-	85 01	STA 01	
4C8-	69 00	ADC #00	copie cette adresse + #117 en 04/05
4CA-	85 03	STA 03	(début du "Descriptor Buffer")
4CC-	69 01	ADC #01	
4CE-	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 1ère fois

4D0-	C8	INY	qui passe donc à 1
4D1-	B1 00	LDA (00),Y	lit l'octet de rang #01 dans le "Channel Buffer"
4D3-	8D 83 C0	STA C083	et le copie en C083 (longueur d'une fiche ou #00)
4D6-	88	DEY	qui repasse à 0
4D7-	B1 00	LDA (00),Y	lit l'octet de rang #00 dans le "Channel Buffer"
4D9-	85 0B	STA 0B	et le copie en 0B (flag "**S/R/D**")
4DB-	60	RTS	retourne avec Y = #00

Ajoute A au contenu de 02/03 (appelé de F5D9, F5F4 et FCAA)
(s/p F4DC-F4E5, appelé par les commandes >, LSET et RSET)

4DC-	18	CLC	prépare une addition
4DD-	65 02	ADC 02	
4DF-	85 02	STA 02	
4E1-	90 02	BCC F4E5	02/03 = 02/03 + A
4E3-	E6 03	INC 03	
4E5-	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)

(s/p F4E6-F5B9, appelés par les commandes >, CLOSE, FIELD, LSET, RSET et TAKE)

4E6-	A9 80	LDA #80	pour supprimer tous les noms de champs associés
4E8-	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)

4E9-	A9 00	LDA #00	pour localiser un nom de champ (erreur s'il
4EB-	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
F4EE- 2C A9 40 BIT 40A9 associé au fichier existe (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 corresp 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 corresp 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 corresp 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"

```
540-  99 76 C0     STA C076,Y    et le copie en C076/C07F (travaille par la fin)
543-  88           DEY           vise l'octet précédent
544-  10 F8        BPL F53E      reboucle en F53E tant qu'il en reste à copier
546-  30 E6        BMI F52E      fini: termine en F52E avec l'adresse de l'entrée corresp 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

548-  88           DEY           décrémente Y (passera successivement de #06 à #00)
549-  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"
54B-  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
54D-  D9 76 C0     CMP C076,Y    et le compare avec l'octet homologue en C076/C07C
550-  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
552-  A9 0A        LDA #0A       pour incrémenter le pointeur de 10 octets
554-  18           CLC           prépare une addition
555-  65 F2        ADC F2
557-  85 F2        STA F2
559-  90 AA        BCC F505      calcule F2/F2 = F2/F3 + #0A
55B-  E6 F3        INC F3
55D-  B0 A6        BCS F505      et reprend d'office en F505

      Tous les noms de champ ont été examinés
55F-  2C 82 C0     BIT C082      teste si le b6 de CO82 est à zéro (pas de réservation
562-  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

564-  A0 04        LDY #04       vise l'offset du début du "Field Buffer"
566-  B1 9E        LDA (9E),Y    lit l'octet de rang #04 de FI
568-  48           PHA           l'empile
569-  AA           TAX           et garde une copie dans X
56A-  C8           INY           vise l'octet suivant
56B-  B1 9E        LDA (9E),Y    lit l'octet de rang #05 de FI
56D-  48           PHA           l'empile
56E-  A8           TAY           et garde une copie dans Y
56F-  8A           TXA           récupère le 1er des 2 dans A
570-  20 E3 F3     JSR F3E3      calcule l'adresse F2/F3 corresp à l'offset AY dans FI, c'est à dire l'adresse du "Field Buffer"
      et retourne avec Y = #00
573-  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" corresp)
576-  68           PLA
577-  A8           TAY           récupère Y
578-  68           PLA
579-  AA           TAX           récupère X (XY, offset du début du "Field Buffer")
57A-  A9 00        LDA #00
57C-  85 F2        STA F2        AF2 = #0064 (10 "entrées" de 10 octets = 100)
57E-  A9 64        LDA #64
580-  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
ette partie corrige la "correction" effectuée par le s/p F425 qui a ajouté #64 à la paire d'octets 04/05 de FI alors qu'il ne fallait pas!

583-  38           SEC           prépare une soustraction
584-  A0 04        LDY #04
586-  B1 9E        LDA (9E),Y    lit l'octet de rang #04 de FI
588-  E9 64        SBC #64       en retire #64
58A-  91 9E        STA (9E),Y    et le remet en place
58C-  C8           INY
58D-  B1 9E        LDA (9E),Y    reporte la retenue sur l'octet de rang #05
58F-  E9 00        SBC #00
591-  91 9E        STA (9E),Y    et le remet en place

      Ajoute 10 au nombre total d'emplacements pour noms de champ
593-  A0 06        LDY #06       vise le nombre total d'emplacements possibles
595-  A9 09        LDA #09       C = 1 en entrée donc ajoute 10 en fait
597-  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

EXECUTION COMMANDE SEDORIC ">"
(Fichiers "**R**" et "**D**")
(s/p 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 s/p 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 et D3A1/RAMOV 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" corresp 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

```
5D6-  AD 7D C0   LDA C07D     lit l'index de début du champ dans la fiche
5D9-  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**"

```
5DC-  A6 0B      LDX 0B       flag "S/R/D"
5DE-  CA         DEX          teste si #01, c'est à dire si passe à zéro
5DF-  D0 08      BNE F5E9     sinon (pas type "D"), continue en F5E9
```

Fichier de type accès **D**isque

```
5E1-  AE 7F C0   LDX C07F     type de champ
5E4-  AC 7E C0   LDY C07E     longueur de champ
5E7-  D0 0E      BNE F5F7     suite forcée en F5F7
```

Fichier de type "**R**" (accès direct) ou **S**équentiel

```
5E9-  A0 00      LDY #00      vise 1er octet du "Channel's own Data Buffer" ("R") ou du "General Buffer" ("S")
5EB-  B1 02      LDA (02),Y   type de champ
5ED-  C8         INY
5EE-  AA         TAX
5EF-  B1 02      LDA (02),Y   longueur de champ
5F1-  A8         TAY
5F2-  A9 02      LDA #02      ajoute #02 au pointeur 02/03 du "Channel's own
5F4-  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

```
5F7-  84 F5      STY F5       longueur de champ
5F9-  8A         TXA          type de champ
5FA-  30 29      BMI F625     continue en F625 si champ de type "chaîne"
5FC-  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"

```
5FE-  A0 04      LDY #04      lit les 5 octets du nombre réel présents
600-  B1 02      LDA (02),Y   au pointeur 02/03 du "Channel's own Data Buffer"
602-  99 D0 00   STA 00D0,Y   et les copie dans ACC1
605-  88         DEY          reboucle tant qu'il en reste à copier
606-  10 F8      BPL F600     puis continue en F620 pour terminer la copie de ce
608-  30 16      BMI F620     nombre réel dans la variable BASIC
```

Sédoric V2.0 GB

La commande ">" (affecte un champ à une variable) a été modifiée, afin de permettre l'utilisation correcte d'un nombre réel à partir d'un fichier. 12 octets diffèrent entre les deux versions dans la zone F5FE/F609:

```
5FE-  A5 02      LDA #02      AY, adresse de la valeur du nombre réel
600-  A4 03      LDY 03       dans le "Channel's own Data Buffer"
602-  20 BA D2   JSR D2BA     JSR DE7B/ROM place dans ACC1 (floating point accumulator) la valeur pointée par AY
605-  4C 20 F6   JMP F620     reprise en F620 du cours normal de la commande ">"
608-  EA         NOP          pour terminer la copie de ce
609-  EA         NOP          nombre réel dans la variable BASIC
```

Champ de type "entier" ou simple octet

```
60A-  0A         ASL          teste le b6 du type de champ
60B-  0A         ASL          C = 0 si "entier" et C = 1 si simple octet
60C-  A0 00      LDY #00      index de lecture dans le "Channel's own Data Buffer"
60E-  B1 02      LDA (02),Y   lit le simple octet ou le LL d'un nombre "entier"
610-  A8         TAY          le copie dans Y si c'est un simple octet
611-  85 F2      STA F2       et dans F2 pour le cas où c'est le LL d'un "entier"
613-  A9 00      LDA #00      HH pour le cas où c'est un simple octet
615-  B0 06      BCS F61D     continue en F61D si c'est un simple octet
617-  A0 01      LDY #01      index l'octet suivant
619-  B1 02      LDA (02),Y   lit le HH d'un nombre "entier"
61B-  A4 F2      LDY F2       récupère le LL du nombre "entier"
61D-  20 54 D2   JSR D254     JSR D499/ROM convertit le nombre signé AY en nombre réel dans ACC1
```

Copie ACC1 dans la variable BASIC

```
620-  A5 29      LDA 29       lit le flag "entier"
622-  4C FE D1   JMP D1FE     JSR CB39/ROM affecter un nombre à une variable
```

Copie une chaîne dans la variable BASIC

```
625-  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 copie la longueur et l'adresse de la chaîne "dans" la variable BASIC à l'adresse pointée en B8/B9

Série de NOP en attente

F638- EA NOP
F639- EA NOP
F63A- EA NOP
F63B- EA NOP
F63C- EA NOP
F63D- EA NOP
F63E- EA NOP
F63F- EA NOP

Sédoric V2.0 GB

Un micro sous-programme a été ajouté dans la zone F638/F63D qui contenait auparavant des NOP. Ce sous-programme est appelé par la commande INIT en C5AE, afin d'insérer un appel au nouveau sous-programme FF4A qui permet de sauver le 2ème secteur de bitmap.

F638- **8E 15 31** STX 3115 pour remplacer le STX écrasé en C5AE/C5B0 (banque n°6)
F63B- **4C 4A FF** JMP FF4A autre micro sous-programme qui consiste en:

FF4A- **A0 03** LDY #03 pour 3ème secteur de la piste 20
FF4C- **4C 8B DC** JMP DC8B lui-même modifié en:

DC8B- **A9 14** LDA #14 pour la piste 20
DC8D- **4C 8E DA** JMP DA8E qui reprend le cours normal de XSMAP et sauve BUF2 sur la disquette dans le 3ème secteur de la piste 20, ouf!

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 s/p 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 et le convertit en MAJUSCULE
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 lit le caractère suivant à TXTPTR et le convertit en MAJUSCULE
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 s/p 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 s/p 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: Sédoric 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).

66C- 20 98 D3 JSR D398 XCRGET lit le caractère suivant à TXTPTR et le convertit en MAJUSCULE
66F- 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)
672- 8E 7B C0 STX C07B sauve X dans C07B (index de l'élément du tableau)
675- A9 29 LDA #29 code ASCII de ")"
677- 4C 2E D2 JMP D22E JSR D067/ROM et D3A1/RAMOV demande une ")" à TXTPTR, lit le caractère suivant, le convertit éventuellement en MAJUSCULE et retourne

Compare les types d'enregistrement et de variables
(ce s/p est aussi appelé par les commandes LSET et RSET)

67A- AD 7F C0 LDA C07F lit l'octet en C07F (type d'enregistrement)
(ce s/p est aussi appelé par les commandes PUT et TAKE)
67D- 8D 7F C0 STA C07F écrit l'octet en C07F (type d'enregistrement)
680- 0A ASL C reçoit le b7 de C07F (flag pour tester le type)
681- 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
(s/p 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 s/p charge dans le "General Buffer" les 2 secteurs contenant la fiche dont le n° est indiqué en 33/34.

684- A9 00 LDA #00 force F2 à zéro, ce sera l'octet de poids le plus
686- 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.
688- 8D 85 C0 STA C085 force C085 à zéro (ce sera le rang de l'octet)
68B- 85 08 STA 08
68D- 85 09 STA 09 force 08/09 à zéro (vise le secteur de rang n° #00)
68F- AD 83 C0 LDA C083 lit la longueur de fiche LF
692- A2 08 LDX #08 pour 8 rebouclages (nombre de bits de LF à traiter)
694- 85 F3 STA F3 c'est le multiplicateur LF
696- 46 F3 LSR F3 teste si le b0 de cet octet est nul
698- 90 15 BCC F6AF si oui, continue en F6AF, il n'y a pas d'addition à effectuer pour ce bit qui est nul
69A- 18 CLC sinon, prépare l'addition:
69B- A5 33 LDA 33 C085/08/09 = C085/08/09 + 33/34/F2
69D- 6D 85 C0 ADC C085
6A0- 8D 85 C0 STA C085 C085 = C085 + 33
6A3- A5 34 LDA 34
6A5- 65 08 ADC 08
6A7- 85 08 STA 08 08 = 08 + 34 + retenue précédente
6A9- A5 F2 LDA F2
6AB- 65 09 ADC 09
6AD- 85 09 STA 09 09 = 09 + F2 + retenue précédente
6AF- 06 33 ASL 33 calcule la valeur de la tranche suivante 33/34/F2
6B1- 26 34 ROL 34 qu'il faudra ajouter au totalisateur si le bit
6B3- 26 F2 ROL F2 corresp du multiplicateur est à 1. Ceci est réalisé par un décalage à gauche portant sur les trois octets 33/34/F2.
6B5- CA DEX décrémente le nombre de bit restant à traiter
6B6- D0 DE BNE F696 reboucle tant que X n'est pas nul
6B8- 20 CD F6 JSR F6CD charge 2 secteurs du fichier de la disquette dans le "General Buffer". Le 1er secteur contient le début de la fiche spécifiée. Lorsque le s/p est utilisé par la commande PUT, il alloue, si nécessaire, des secteurs supplémentaires sur la disquette à ce fichier de type "**R**".
6BB- 20 A8 F4 JSR F4A8 place l'adresse du début du "Channel Buffer" corresp 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
6BE- 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" corresp 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 s/p!

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 #05c'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 s/p 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 s/p 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 1er tour, c'est à dire pour le

premier descripteur. C'est donc en fait un compteur du nombre des descripteurs secondaires nécessaires.

707-	B0 F0	BCS F6F9	reboucle en F6F9 si le reste est supérieur à #7F
709-	C8	INY	nombre réel de secteurs répertoriés dans le dernier
70A-	98	TYA	(l'ancien nombre était un rang commençant à zéro)
70B-	0A	ASL	le multiplie par 2 (nombre d'octets requis)
70C-	8D 84 C0	STA C084	pointeur dans le dernier descripteur
70F-	85 F8	STA F8	idem
711-	8A	TXA	
712-	48	PHA	nombre des descripteurs secondaires nécessaires
713-	20 A8 F4	JSR F4A8	place l'adresse du début du "Channel Buffer" corresp 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
716-	68	PLA	récupère le rang R du dernier descripteur
717-	18	CLC	prépare une addition
718-	65 05	ADC 05	04/05 adresse du début du "Descriptor Buffer"
71A-	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
71C-	85 F7	STA F7	sauve également le résultat dans F7
71E-	AC 84 C0	LDY C084	pointeur dans le dernier descripteur
721-	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 1ère page du "General Buffer" de **FI** et incrémente le HH de ce pointeur, c'est à dire 07
724-	4C 36 F7	JMP F736	idem pour le secteur suivant dans la 2ème 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 2ème 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

727-	20 A8 F4	JSR F4A8	place l'adresse du début du "Channel Buffer" corresp 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
72A-	A5 F7	LDA F7	HH de l'adresse du descripteur courant remis en
72C-	85 05	STA 05	place (04/05, adresse du début du descripteur courant)
72E-	A4 F8	LDY F8	pointeur dans le descripteur courant
730-	20 33 F7	JSR F733	écrit sur la disquette un secteur de data de la 1ère page du "General Buffer" de **FI** pointé par 06/07 et incrémente 07 pour viser la 2ème page. Continue à la routine suivante qui fait de même pour la 2ème 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

733-	A2 A8	LDX #A8	commande "écriture" pour routine XRWTS
735-	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

736-	A2 88	LDX #88	commande "lecture" pour routine XRWTS
738-	B1 04	LDA (04),Y	lit le n° de piste
73A-	8D 01 C0	STA C001	et le copie dans PISTE
73D-	C8	INY	
73E-	B1 04	LDA (04),Y	lit le n° de secteur
740-	8D 02 C0	STA C002	et le copie dans SECTEUR
743-	A5 06	LDA 06	
745-	8D 03 C0	STA C003	
748-	A5 07	LDA 07	
74A-	8D 04 C0	STA C004	met à jour RWBUF avec 06/07
74D-	E6 07	INC 07	prépare HH suivant (page suivante du "General
74F-	C8	INY	Buffer") teste si fin du descripteur atteinte
750-	D0 04	BNE F756	sinon, saute les 2 instructions suivantes
752-	E6 05	INC 05	si oui, incrémente HH (descripteur suivant)
754-	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 secteur de bitmap dans BUF2
F768- 20 A8 F4 JSR F4A8 place l'adresse du début du "Channel Buffer" corresp 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 1ères 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 1ères coordonnées du 1er 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

7B1- D0 03 BNE F7B6
7B3- CE 59 C0 DEC C059 décrémente le nombre de secteurs supplémentaires
7B6- CE 58 C0 DEC C058 à allouer
7B9- 8C 5F C0 STY C05F index du dernier secteur alloué dans le descripteur
7BC- 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
7BF- 84 F2 STY F2 sauve l'indication de secteur libre dans F2
7C1- AC 5F C0 LDY C05F récupère l'index dans le descripteur
7C4- 91 04 STA (04),Y sauve l'indication de piste à la fin de la liste
7C6- C8 INY de coordonnées dans le descripteur
7C7- A5 F2 LDA F2 récupère l'indication de secteur
7C9- 91 04 STA (04),Y sauve l'indication de secteur à la suite
7CB- C8 INY position suivante dans le descripteur
7CC- D0 D5 BNE F7A3 reboucle en F7A3 tant que le descripteur n'est pas fini

Génère un autre descripteur, car le précédant est plein

7CE- AD 58 C0 LDA C058 teste si le nombre de secteurs supplémentaires
7D1- 0D 59 C0 ORA C059 requis est nul
7D4- 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

7D6- 20 4C F8 JSR F84C incrémente le nombre de secteurs totaux du fichier et cherche un secteur libre AY sur la bitmap
7D9- 85 F5 STA F5
7DB- 84 F6 STY F6 sauve les coordonnées piste/secteur en F5/F6
7DD- A0 00 LDY #00
7DF- 91 04 STA (04),Y copie l'indication de piste du nouveau descripteur au début du descripteur courant (lien
7E1- C8 INY indiquant les coordonnées dudescripteur suivant) récupère
7E2- A5 F6 LDA F6 l'indication de secteur du nouveau descripteur
7E4- 91 04 STA (04),Y sauve l'indication de secteur à la suite
7E6- 20 A4 DA JSR DAA4 XSVSEC écrit le descripteur courant qui était plein sur la disquette, selon DRIVE, PISTE, SECTEUR et RWBUF
7E9- A5 F5 LDA F5
7EB- A4 F6 LDY F6 récupère les coordonnées piste/secteur
7ED- 8D 01 C0 STA C001 et copie dans PISTE
7F0- 8C 02 C0 STY C002 et dans SECTEUR pour le prochain descripteur
7F3- 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
7F6- A9 00 LDA #00
7F8- A8 TAY
7F9- 91 04 STA (04),Y forcer à zéro toute la page
7FB- C8 INY de ce nouveau descripteur
7FC- D0 FB BNE F7F9
7FE- A0 02 LDY #02 index des premières coordonnées dans le nouveau descripteur
800- 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.

802- 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 s/p, 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.

805- 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)
807- B1 00 LDA (00),Y lit un octet selon l'adresse en 00/01 + Y
809- 99 22 C0 STA C022,Y et le copie dans BUFNOM
80C- C8 INY
80D- C0 17 CPY #17
80F- D0 F6 BNE F807 reboucle en F807 tant qu'il en reste à copier
811- 20 30 DB JSR DB30 XTVNM cherche le fichier BUFNOM revient avec POSNMX POSNMP et POSNMS ou Z = 1 si rien trouvé
814- 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 XSMAP sauve le secteur de bitmap sur la disquette
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" corresp au NL en 00/01, celle du début du 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 corresp
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
(s/p 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 s/p)

```
375-  A5 04      LDA 04      AY est l'adresse de début du nouveau
377-  A4 05      LDY 05      descripteur
379-  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")
37C-  A9 01      LDA #01
37E-  85 F2      STA F2      AF2 = #0100
380-  A9 00      LDA #00     (nombre d'octets à insérer à l'adresse 04/05)
382-  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
385-  38         SEC         prépare une soustraction
386-  E5 9E      SBC 9E
388-  AA         TAX
389-  98         TYA
38A-  E5 9F      SBC 9F      calcule XY = AY - 9E/9F
38C-  A8         TAY
38D-  60         RTS
```

COMMANDE &()
(Fichiers "**S**" et "**R**")
(s/p 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 s/p 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. Sédoric 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)
88E-  20 4C D2   JSR D24C    JSR D2A9/ROM nombre en ACC1 -> #D4-#D3 (signé)
891-  A5 D4      LDA D4
893-  A6 D3      LDX D3      A = LL et X = HH
895-  10 0C      BPL F8A3    continue en F8A3 si c'est un nombre entier positif

      C'est un nombre entier négatif
897-  49 FF      EOR #FF     inverse tous les bits du LL et ajoute 1
899-  18         CLC         (calcule le complément à 2 du LL,
89A-  69 01      ADC #01     c'est à dire le NL du fichier à tester)
89C-  E0 FF      CPX #FF     teste si HH est bien égal à #FF (D3 garde le signe)
89E-  F0 07      BEQ F8A7    si oui, tout va bien, continue en F8A7
8A0-  4C 20 DE   JMP DE20    sinon "ILLEGAL QUANTITY ERROR"

      C'est un nombre entier positif
8A3-  E0 00      CPX #00     teste si HH est bien égal à #00 (D3 garde le signe)
8A5-  D0 F9      BNE F8A0    si ce n'est pas le cas, "ILLEGAL QUANTITY ERROR"

      Ce nombre (NL présent dans A) est un octet signé
8A7-  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" corresp 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 s/p 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
```

EXECUTION COMMANDE SEDORIC TAKE
(Fichiers "**S**","**R**"et "**D**")
(s/p 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 s/p F3CF.

Variables utilisées

En voir la liste et les fonctions dans les généralités sur la gestion de fichiers de data situées devant le s/p F3CF.

Analyse de la syntaxe et saisie des paramètres

8DF- 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**"

8E2- D0 06 BNE F8EA continue en F8EA si ce n'est pas un fichier "**D**"

TAKE pour un fichier d'accès **D**isque

8E4- 20 6B F9 JSR F96B lit à TXTPTR piste, secteur et (si indiqué) drive, initialise RWBUF au début du "Channel's own Data Buffer"

8E7- 4C 73 DA JMP DA73 XPRSEC lit ce secteur selon DRIVE, PISTE, SECTEUR et RWBUF dans le "Channel's own Data Buffer" corresp au NL indiqué et retourne

Suite de l'analyse de la syntaxe et saisie des paramètres

8EA- 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 1er 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".

8EC- 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")

8EF- 08 PHP sauvegarde les indicateurs 6502

8F0- 78 SEI interdit les interruptions

8F1- 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 1er secteur) et retourne avec Y = 0

8F4- B1 06 LDA (06),Y la fiche proprement dite (en fait un bloc de 256

8F6- 91 02 STA (02),Y octets situés à partir du début de la fiche, ce

8F8- 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

8F9- D0 F9 BNE F8F4 reboucle en F8F4 tant qu'il en reste à copier

8FB- 28 PLP récupère les indicateurs

8FC- 60 RTS et retourne

TAKE pour un fichier **S**équentiel

8FD- 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 s/p appartient à la commande LINPUT)

900- 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

903- 8A TXA type du data

904- 20 7D F6 JSR F67D vérifie que type de data = type de variable

907- A5 06 LDA 06

909- A4 07 LDY 07 AY, pointeur dans l'enregistrement

90B- 85 02 STA 02

90D- 84 03 STY 03 02/03, pointeur dans le "General Buffer"

90F- 20 DC F5 JSR F5DC place le data (la donnée) dans la variable BASIC

912- 20 9E D3 JSR D39E XCRGOT relit le caractère à TXTPTR et le convertit en MAJUSCULE

915- F0 E5 BEQ F8FC RTS en F8FC s'il n'y a plus de variable à pourvoir

917- 20 2C D2 JSR D22C D067/ROM exige une "," place TXTPTR sur l'octet suivant

91A- 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

(s/p appelé par la commande PUT)

91D- 18 CLC point d'entrée pour la commande PUT

91E- 24 38 BIT 38 continue en F920

Lit le n° de fiche indiqué à TXTPTR et vérifie si la fiche existe

91F- 38 SEC point d'entrée pour la commande TAKE

920- 08 PHP sauvegarde les indicateurs 6502 dont C

921- 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)

924- A0 04 LDY #04 index visant le nombre de fiches dans le descripteur

926- B1 04 LDA (04),Y

928- C5 33 CMP 33 la fiche demandée existe t-elle?

92A- 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
(s/p 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**"
(s/p 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" corresp 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"
(s/p 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" corresp 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
```

98F- 60 RTS et retourne

EXECUTION COMMANDE SEDORIC PMAP
(Fichiers "**D**")
(s/p 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 s/p F3CF.
Analyse de syntaxe et saisie du paramètre

990-	20 0D E6	JSR E60D	valide drive si indiqué, sinon valide DRVDEF
993-	4C 4C DA	JMP DA4C	XPMAP prend le secteur de bitmap dans BUF2

EXECUTION COMMANDE SEDORIC SMAP
(Fichiers "**D**")
(s/p 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 s/p F3CF.

Analyse de syntaxe et saisie du paramètre

996-	20 0D E6	JSR E60D	valide drive si indiqué, sinon valide DRVDEF
999-	4C 8A DA	JMP DA8A	XSMAP sauve le secteur de bitmap présent dans BUF2

EXECUTION COMMANDE SEDORIC FRSEC
(Fichiers "**D**")
(s/p 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 s/p F3CF.

Analyse de syntaxe et saisie du paramètre

99C-	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)
99F-	8A	TXA	
9A0-	48	PHA	empile ce n° de piste
9A1-	29 7F	AND #7F	force à zéro le n° de face
9A3-	CD 06 C2	CMP C206	vérifie que le n° de piste indiqué est valide
9A6-	B0 20	BCS F9C8	sinon, "ILLEGAL QUANTITY ERROR"
9A8-	20 2C D2	JSR D22C	D067/ROM exige une "," place TXTPTR sur l'octet suivant
9AB-	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)
9AE-	8A	TXA	garde ce n° de secteur dans A
9AF-	CA	DEX	vérifie que ce n° n'est pas #00
9B0-	30 16	BMI F9C8	sinon "ILLEGAL QUANTITY ERROR"
9B2-	EC 07 C2	CPX C207	vérifie que le n° de piste indiqué est valide
9B5-	B0 11	BCS F9C8	sinon "ILLEGAL QUANTITY ERROR"
9B7-	A8	TAY	
9B8-	68	PLA	AY coordonnées secteur Y de la piste A
9B9-	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 ERROR" ou une "READ FAULT ERROR"). Pour cela, le masque obtenu en DD15 doit être inversé et le ORA en DD18 remplacé par un AND.

EXECUTION COMMANDE SEDORIC CRESEC
(Fichiers "**D**")
(s/p 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 s/p 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"

EXECUTION COMMANDE SEDORIC PUT
(Fichiers "**S**", "**R**" et "**D**")
(s/p 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 1er 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 s/p F3CF.

Variables utilisées

En voir la liste et les fonctions dans les généralités sur la gestion de fichiers de data situées devant le s/p 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

9CB- 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**"

9CE- D0 06 BNE F9D6 continue en F9D6 si c'est un fichier "**R**" ou "**S**"

Commande PUT pour fichier de type "**D**":

9D0- 20 6B F9 JSR F96B lit les coordonnées indiquées (secteur, piste et éventuellement drive), initialise RWBUF au début du "Channel's own Data Buffer"

9D3- 4C A4 DA JMP DAA4 XSVSEC écrit un secteur selon DRIVE, PISTE, SECTEUR et RWBUF

Suite commune pour fichiers "**R**" ou "**S**":

9D6- 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

9D8- 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.

9DB- 08 PHP sauvegarde les indicateurs 6502

9DC- 78 SEI interdit les interruptions

9DD- 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 1er secteur) et retourne avec Y = 0

9E0- B1 02 LDA (02),Y lit un octet dans le "Channel's own Data Buffer"

9E2- 91 06 STA (06),Y et l'écrit dans le "General Buffer"

9E4- C8 INY vise le suivant

9E5- CC 83 C0 CPY C083 compare avec la longueur de la fiche

9E8- D0 F6 BNE F9E0 et reboucle s'il en reste à copier

9EA- 20 27 F7 JSR F727 écrit sur la disquette les 2 secteurs de data ("General Buffer") contenant la fiche spécifiée

9ED- 28 PLP récupère les indicateurs

9EE- 60 RTS

Commande PUT pour fichier de type "**S**":

9EF- 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

9F2- 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)

9F5- 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

9F7- A2 05 LDX #05 longueur d'enregistrement par défaut (nombre réel)

9F9- A0 00 LDY #00 type d'enregistrement par défaut (nombre réel)

9FB- 24 28 BIT 28 teste la variable indiquée ("chaîne" ou "nombre")

9FD- 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

9FF- A5 D3 LDA D3

A01- A6 D4 LDX D4

A03- 85 91 STA 91 adresse de cette chaîne

A05- 86 92 STX 92

A07- B1 91 LDA (91),Y lit la longueur de la chaîne

A09- AA TAX et garde cette information dans X

A0A- 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 et le convertit en MAJUSCULE
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" corresp 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.

EXECUTION COMMANDE SEDORIC OPEN
(Fichiers "**S**", "**R**" et "**D**")
(s/p 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,NF,NL(,LF ,NR) pour ouvrir un fichier à accès direct (**R**andom) de nom **NF**, 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 1ère 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 NF est égale au nombre d'octets de data, soit NR x LF / 256 secteurs.

OPEN S,NF,NL pour ouvrir un fichier **S**équentiel de nom **NF** 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 **NF** à un numéro logique **NL**. **NF** est toujours un nom de fichier non ambigu 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 s/p 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 (1ère 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 1er 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 NF 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 NF 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 1er octet de l'enregistrement dans le "Channel's own Data Buffer".

Organigramme de la commande OPEN **R** pour un nouveau fichier

FA50 analyse du 1er 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 NF 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 NF 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 s/p F684, alloue autant de secteurs que nécessaire pour toutes ces fiches et termine en chargeant dans le "General Buffer" les deux secteurs corresp 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 1er 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 lit le caractère suivant (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 revalide 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"

A74- A0 06 LDY #06 index pour écriture
A76- AD 00 C0 LDA C000 copie DRIVE (lecteur à utiliser)
A79- 91 00 STA (00),Y dans l'octet de rang #06 du "Channel Buffer"
A7B- 60 RTS

Suite de l'analyse des paramètres

A7C- C9 52 CMP #52 est-ce un "R"?
A7E- D0 12 BNE FA92 sinon, poursuit l'analyse en FA92

OPEN R

A80- A9 00 LDA #00 si oui, initialise A avec #00 (flag "**R**")
A82- A0 08 LDY #08 et initialise Y avec #08 (FTYPE "fichier direct")
A84- 20 08 FB JSR FB08 ce long sous-programme initialise 0B (flag "**R**") et F9 (FTYPE), saisit et vérifie NF 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 NF 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).

A87- D0 F2 BNE FA7B simple RTS si le fichier existait déjà
A89- 20 2C D2 JSR D22C D067/ROM exige une "," place TXTPTR sur l'octet suivant
A8C- 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.

A8F- 4C 23 DE JMP DE23 "SYNTAX ERROR"

Suite de l'analyse des paramètres

A92- C9 53 CMP #53 est-ce un "S"? (dernière possibilité)
A94- D0 F9 BNE FA8F sinon, "SYNTAX ERROR"

OPEN S

A96- A9 80 LDA #80 si oui, A = #80 (flag "**S**" pour initialiser 0B)
A98- A0 10 LDY #10 et Y = #10 (FTYPE "séquentiel" pour initialiser F9)
A9A- 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 NF 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 (sino "FILE ALREADY OPEN ERROR"), cherche le fichier NF 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 1er secteur du fichier),
sinon écrit #FF au début du "Channel's own Data Buffer"
et continue en FD66 (sauve 1er secteur du fichier)

A9D- 08 PHP sauvegarde Z (à zéro si le fichier existait déjà)
A9E- A0 03 LDY #03 index pour écriture dans le "Channel Buffer"
AA0- A9 0C LDA #0C écrit #0C (index de lecture dans le 1er descripteur)
AA2- 91 00 STA (00),Y dans l'octet de rang #03 du "Channel Buffer"
AA4- C8 INY
AA5- A9 00 LDA #00 écrit #00 dans l'octet de rang #04 du
AA7- 91 00 STA (00),Y "Channel Buffer" (n° du descripteur courant)
AA9- C8 INY écrit #00 dans l'octet de rang #05 du
AAA- 91 00 STA (00),Y "Channel Buffer" (index dans le tampon)
AAC- 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 1er 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 1er octet.

EXECUTION COMMANDE SEDORIC REWIND
(Fichiers "**S**")
(s/p 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 s/p F3CF.

Analyse de la syntaxe et saisie du paramètre

ABB- 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**").
ABE- 30 DD BMI FA9D suite forcée en FA9D, car au retour du s/p 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 s/p FAC0-FACA, est aussi appelé par les commandes APPEND, JUMP, LTYPE et TYPE)

AC0- 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
AC3- 20 A8 F4 JSR F4A8 place l'adresse du début du "Channel Buffer" corresp 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
AC6- 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.
AC8- 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)
(s/p 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".

ACB- 48 PHA sauvegarde A puis Y sur la pile (rappel: A est la
ACC- 98 TYA longueur de fiche si "**R**" sinon A = #00; Y = #00
ACD- 48 PHA si "**R**" ou #80 si "**S**" ou #01 si "**D**")
ACE- A0 05 LDY #05 index pour lecture dans en-tête de **FI**
AD0- B1 9E LDA (9E),Y lit HH de l'offset du début du "Field Buffer"
AD2- 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**).
AD4- A0 03 LDY #03 index pour lecture dans l'en-tête de **FI**
AD6- 88 DEY pointe le précédent (donc octet de rang #04 ou #02)
AD7- B1 9E LDA (9E),Y à l'issue de cette partie, XY (et sa copie sur
AD9- AA TAX la pile) contient une valeur d'offset qui se
ADA- 48 PHA trouvait soit dans les octets de rang #04/05
ADB- C8 INY et qui représente le début du "Field Buffer"
ADC- B1 9E LDA (9E),Y soit dans ceux de rang #02/03,
ADE- 48 PHA qui représentent la fin actuelle de **FI**
ADF- A8 TAY XY (et sa copie sur la pile) pointe sur le nouveau
AE0- A9 01 LDA #01 "Channel Buffer", c'est aussi l'adresse de base
AE2- 85 F2 STA F2 du bloc à déplacer vers le haut de AF2 octets
AE4- A9 21 LDA #21 AF2 = #0121
AE6- 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"
AE9- 20 CF F3 JSR F3CF calcule l'adresse F2/F3 de la paire d'octets corresp 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 corresp au NL)
FAF4- 20 A8 F4 JSR F4A8 place l'adresse du début du "Channel Buffer" corresp 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 NF et n° logique NL,
crée si besoin et ouvre le fichier corresp,
en charge le ou les descripteurs dans le "Descriptor Buffer"
(s/p 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 NF 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 NF 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 NF 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 cherche "?" dans BUFNOM ("WILDCARD(S) NOT ALLOWED")
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 NF 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 Sédoric, 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)

32A- 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!

32C- 90 D7 BCC FB05 si oui, "ILLEGAL QUANTITY ERROR"

B2E- A9 00 LDA #00

330- 8D 52 C0 STA C052 force DESALO à #0000

333- 8D 53 C0 STA C053 (adresse fictive de début de fichier)

336- A8 TAY force Y à zéro et copie X dans A

337- 8A TXA (c'est à dire la longueur de fiche ou #00)

338- A6 F9 LDX F9 type de fichier (#08 si "**R**", #10 si "**S**")

33A- 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).

33D- 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)

B40- BD 0C C3 LDA C30C,X coordonnées A = piste et Y = secteur du

343- BC 0D C3 LDY C30D,X descripteur principal de ce fichier

346- 20 5D DA JSR DA5D XPBUF1 charge dans BUF1 le descripteur AY

349- AD 03 C1 LDA C103 type de fichier

34C- C5 F9 CMP F9 teste si différent de celui indiqué par F9

34E- D0 B2 BNE FB02 si oui, "FILE TYPE MISMATCH ERROR"

350- AD 06 C1 LDA C106 longueur d'une fiche si fichier à accès direct

353- A4 0B LDY 0B #00 si "**R**" et #80 si "**S**"

355- 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")

358- A0 07 LDY #07 index pour écriture (ira de #07 à #16)

35A- AE 27 C0 LDX C027 POSNMX, position dans le secteur de catalogue

B5D- BD 00 C3 LDA C300,X lit un octet d'une "entrée" du catalogue

360- 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...)

362- E8 INX suivant en lecture

363- C8 INY suivant en écriture

364- C0 17 CPY #17 teste si 16 octets (une ligne) ont été copiés

366- D0 F5 BNE FB5D sinon, reboucle en FB5D

368- A9 FF LDA #FF qui passera à #00 au début de la routine suivante

36A- A0 02 LDY #02 copie un #FF dans l'octet de rang #02

36C- 91 00 STA (00),Y du "Channel Buffer"

36E- C6 05 DEC 05 décrémente 05. Le s/p 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

B70- 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

373- 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"

376- 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 corresp au descripteur principal

379- A0 00 LDY #00 pour viser le 1er octet du descripteur chargé

37B- B1 04 LDA (04),Y lit cet octet (piste du descripteur suivant)

37D- 8D 01 C0 STA C001 et le copie dans PISTE

380- C8 INY pour viser l'octet suivant du descripteur chargé

381- B1 04 LDA (04),Y lit cet octet (secteur du descripteur suivant)

383- 8D 02 C0 STA C002 et le copie dans SECTEUR

386- 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.

388- 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)

EXECUTION COMMANDE SEDORIC CLOSE
(Fichiers "**S**", "**R**" et "**D**")
(s/p FB8D-FBBE)

Rappel de la syntaxe
CLOSE (NL)(,NL)(,NL) etc...

Libère le ou les n° logique(s) indiqués (tous les n° logiques en absence d'indication), ferme le ou les fichier(s) corresp 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: 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 s/p F3CF.

Principe de la commande CLOSE

Pour chaque NL devant être "fermé", l'octet de poids fort de l'offset corresp à ce NL dans la "Table NL" est remis à zéro pour indiquer que ce NL est désormais libre et que le fichier corresp est fermé.
Chaque champ relatif au NL devant être "fermé" est libéré en plaçant un zéro au début de l'entrée corresp 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, le "Channel Buffer" corresp 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 le Sédoric 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 corresp 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 et le convertit en MAJUSCULE
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 corresp 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 corresp
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 corresp au NL dans la "Table NL" et revient avec Y = #00

3B2- 98 TYA force A à zéro
3B3- C8 INY et Y à 1
3B4- 91 F2 STA (F2),Y force à zéro le 2ème octet de la paire d'octets corresp au NL (HH de l'offset qui ne peut jamais être nul)
3B6- 4C E6 F4 JMP F4E6 suite forcée en F4E6 pour supprimer tous les noms de champ spécifiés pour ce fichier. En fait les emplacements corresp sont rendus disponibles pour définir de nouveaux champs, mais la mémoire occupée n'est pas libérée pour un autre usage!

3B9- 4C E0 E0 JMP E0E0 "FILE TYPE MISMATCH ERROR"

3BC- 4C 23 DE JMP DE23 "SYNTAX ERROR"

EXECUTION COMMANDE SEDORIC FIELD
(Fichiers "**R**" et "**D**")
(s/p 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 (redéfinition de la fiche modèle).

Non ou mal documenté

1) Attention le "TO" de la commande FIELD, comme celui des autres commandes Sédoric doit être en majuscule, sinon il n'est pas reconnu. Il y a un peu de flottement dans l'usage des minuscules pour les commandes Sédoric (bogue)!

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 notre 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 il serait peut-être mieux de reprogrammer cette partie de manière à prévenir ("DUPLICATE FIELD").

5) Il est également possible d'avoir le même "nom_de_champ(index)" dans plusieurs fichiers. Cependant, peut-être à cause

d'une bogue du Sédoric, lors d'un transfert de ou vers un champ, grâce aux commandes LSET ou RSET, seul le 1er "nom_de_champ(index)" est accessible. Ceci est dû au fait que Sédoric 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 Sédoric: 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 Sédoric 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 (Sédoric 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 s/p 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" corresp 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 et D3A1/RAMOV 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 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

```
C03-  F0 4F      BEQ FC54    "ILLEGAL QUANTITY" si longueur nulle
C05-  A0 80      LDY #80     code pour type alphanumérique

C07-  8C 7F C0   STY C07F    sauve le type de champ
C0A-  8E 7E C0   STX C07E    sauve la longueur du champ
C0D-  AD 81 C0   LDA C081    longueur totale des champs jusqu'ici
C10-  A4 0A      LDY 0A      NL du fichier courant
C12-  8D 7D C0   STA C07D    longueur totale des champs jusqu'ici
C15-  8C 7C C0   STY C07C    NL du fichier courant
C18-  18         CLC         prépare les additions en FC1D et FC22
C19-  A6 0B      LDX 0B      type "S/R/D" du fichier courant
C1B-  D0 05      BNE FC22    saute les 2 instructions suivantes si fichier de type "D"
```

Cas d'un fichier de type "**R**" à accès direct

```
C1D-  69 02      ADC #02     ajoute déjà les 2 octets séparateurs
C1F-  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

```
C22-  6D 7E C0   ADC C07E    ajoute la longueur du champ
C25-  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
C28-  8D 81 C0   STA C081    sauve la longueur totale des champs jusqu'ici
```

Met à jour l'emplacement corresp du "Field Buffer"

```
C2B-  20 EC F4   JSR F4EC    vérifie si le nom de champ existe déjà pour ce fichier
C2E-  B0 03      BCS FC33    si oui, saute l'instruction suivante (utilise le même emplacement)
C30-  20 EF F4   JSR F4EF    sinon, réserve une place disponible pour un nouveau nom de champ associé au fichier
      courant
C33-  A0 09      LDY #09     pour copier les #10 octets de nom de champ etc...
C35-  B9 76 C0   LDA C076,Y  du "General Field Buffer"
C38-  91 F4      STA (F4),Y  dans le "Field Buffer"
C3A-  88         DEY         en travaillant par la fin
C3B-  10 F8      BPL FC35    reboucle en FC35 tant qu'il en reste à copier
```

Y a t-il encore des paramètres?

```
C3D-  A2 00      LDX #00     pour éventuelle mise à zéro de la longueur totale
C3F-  20 9E D3   JSR D39E    XCRGOT relit le caractère à TXTPTR et le convertit en MAJUSCULE
C42-  D0 04      BNE FC48    saute les 2 instructions suivantes si pas fini
C44-  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
C47-  60         RTS         commande FIELD et retourne
C48-  20 2E D2   JSR D22E    JSR D067/ROM et D3A1/RAMOV 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...
C4B-  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

```
C4D-  A5 0A      LDA 0A      NL du fichier courant
C4F-  8D 80 C0   STA C080    sauvegarde du NL de la dernière commande FIELD
C52-  18         CLC         force C = 0
C53-  60         RTS         et retourne
C54-  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!

```
C57-  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")
C59-  B0 13      BCS FC6E    si le résultat de l'addition précédente dépasse 255 pour un fichier "R", génère une "FIELD
      OVERFLOW ERROR"
C5B-  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
```

EXECUTION COMMANDE SEDORIC LSET
(Fichiers "**R**" et "**D**")
(s/p 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 justifiée 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 s/p F3CF.

Entrée de la commande LSET

```
FC73-  18     CLC      flag pour LSET
FC74-  24 38  BIT 38   continue en FC76
```

EXECUTION COMMANDE SEDORIC RSET
(Fichiers "**R**" et "**D**")
(s/p 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 justifiée 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 s/p F3CF.

Entrée de la commande RSET

```
FC75-  38     SEC      flag pour RSET
```

Suite commune pour les commandes LSET et RSET

C**76**- 08 PHP sauvegarde C (flag "LSET/RSET")
C77- 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 !!!)
C7A- 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
C7D- A9 D5 LDA #D5 token de "<"
C7F- 20 2E D2 JSR D22E JSR D067/ROM et D3A1/RAMOV demande "<" à TXTPTR, lit le caractère suivant et le convertit éventuellement en MAJUSCULE
C82- 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
C85- 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" corresp dans le "General Field Buffer"
C88- 20 7A F6 JSR F67A compare le type du champ et celui de l'expression
C8B- 20 A8 F4 JSR F4A8 place l'adresse du début du "Channel Buffer" corresp 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
C8E- AD 7C C0 LDA C07C NL du champ courant
C91- 85 0A STA 0A n°logique courant
C93- AC 7D C0 LDY C07D index du début du champ dans l'enregistrement
C96- AE 7E C0 LDX C07E longueur du champ
C99- A5 0B LDA 0B type "**R/D**" du fichier
C9B- D0 0C BNE FCA9 si fichier de type "**D**", continue en FCA9
C9D- 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 s/p 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 s/p 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.

C**9E**- 08 PHP sauvegarde C (LSET /RSET) pour usage ultérieur
C9F- AD 7F C0 LDA C07F type d'enregistrement ou de champ
CA2- 91 02 STA (02),Y stocké au début du tampon
CA4- C8 INY Y = #01
CA5- 8A TXA longueur de l'enregistrement ou du champ
CA6- 91 02 STA (02),Y stocké à la suite
CA8- C8 INY Y = #02

Entrée pour fichier de type "**D**":
C**A9**- 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
CAA- 20 DC F4 JSR F4DC ajoute A au contenu de 02/03 qui vise donc directement les data impliqués
CAD- 28 PLP récupère C (flag LSET /RSET pour chaîne seulement)
CAE- A0 00 LDY #00 nouvel index visant le début des data
CB0- AD 7F C0 LDA C07F teste le type d'enregistrement ou de champ
CB3- 30 22 BMI FCD7 continue en FCD7 si c'est une chaîne
CB5- 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"
CB7- 20 4C D2 JSR D24C JSR D2A9/ROM nombre en ACC1 -> #D4-#D3 (entier)
CBA- A0 00 LDY #00 encore! (index visant le début des data)
CBC- A5 D4 LDA D4 LL du nombre entier ou "octet"
CBE- 91 02 STA (02),Y stocké dans le tampon
CC0- 2C 7F C0 BIT C07F teste le b6 du type d'enregistrement ou de champ
CC3- 50 05 BVC FCCA continue en FCCA si c'est un "entier"
CC5- A5 D3 LDA D3 c'est un "octet" dont le HH doit être nul
CC7- D0 8B BNE FC54 sinon "ILLEGAL QUANTITY ERROR"
CC9- 60 RTS tout fini bien pour le type "octet"
CCA- C8 INY vise l'octet suivant du tampon
CCB- A5 D3 LDA D3 HH du nombre entier
CCD- 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!!!

      Reinitialise 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)
      (s/p FD0E-FD29, appelé par les commandes &, JUMP, LTYPE, PUT et TYPE)
      (pour fichiers de type Séquentiel seulement)

FD0E- 20 A8 F4    JSR F4A8     place l'adresse du début du "Channel Buffer" corresp 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
```

Reinitialise 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
(s/p FD2A-FD72, appelé par les commandes BUILD et PUT)

D2A- 20 A8 F4 JSR F4A8 place l'adresse du début du "Channel Buffer" corresp 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
D2D- 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)
D2F- AD 88 C0 LDA C088 lit la valeur antérieure de cet index
D32- 91 00 STA (00),Y et la remet en place
D34- 88 DEY vise l'octet de rang #04 du "Channel Buffer" (n° du descripteur courant)
D35- AD 87 C0 LDA C087 lit la valeur antérieure de ce n° de descripteur
D38- 91 00 STA (00),Y et la remet en place
D3A- 88 DEY vise l'octet de rang #04 du "Channel Buffer" (index de lecture dans la liste des coordonnées du descripteur courant)
D3B- AD 86 C0 LDA C086 lit la valeur antérieure de cet index et la compare
D3E- D1 00 CMP (00),Y avant écriture avec l'octet qu'il écrasera dans le
D40- 91 00 STA (00),Y "Channel Buffer"
D42- 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"

D44- 18 CLC C = 0 pour charger un secteur de la disquette
D45- 24 38 BIT 38 continue en FD47

Sauve sur la disquette le secteur qui est présent dans le "Channel's own Data Buffer"

D46- 38 SEC C = 1 pour sauver un secteur sur la disquette
D47- 08 PHP sauvegarde les indicateurs 6502 dont C
D48- 20 A8 F4 JSR F4A8 place l'adresse du début du "Channel Buffer" corresp 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
D4B- A5 02 LDA 02 AY = adresse de début du "Channel's own Data Buffer"
D4D- A4 03 LDY 03 corresp au NL traité
D4F- 8D 03 C0 STA C003 mise à jour de RWBUF avec
D52- 8C 04 C0 STY C004 cette adresse
D55- A0 04 LDY #04 vise l'octet de rang #04 du "Channel Buffer"
D57- B1 00 LDA (00),Y lit le n° du descripteur courant
D59- 18 CLC prépare une addition
D5A- 65 05 ADC 05 calcule HH de l'adresse du début du descripteur
D5C- 85 05 STA 05 = HH de l'adresse du début du "Descriptor Buffer"
D5E- 88 DEY + n° du descripteur courant
D5F- B1 00 LDA (00),Y lit l'index de lecture dans le descripteur courant
D61- A8 TAY pointe dans la liste des coordonnées piste/secteur
D62- B1 04 LDA (04),Y lit un n° de PISTE dans cette liste de coordonnées
D64- 48 PHA et l'empile
D65- C8 INY index octet suivant
D66- B1 04 LDA (04),Y lit maintenant le n° de SECTEUR qui suit
D68- A8 TAY et le garde dans Y
D69- 68 PLA récupère le premier (AY = cordonnées piste/secteur)
D6A- 28 PLP récupère les indicateurs dont C initial
D6B- 90 03 BCC FD70 saute l'instruction suivante si C = 0
D6D- 4C 9E DA JMP DA9E XSAY sauve secteur RWBUF à la piste A, secteur Y
D70- 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)
(s/p FD73-FDD8, appelé par les commandes BUILD et PUT, et par le s/p FE38)

D73- 20 CC FD JSR FDCC Y = index dans le "Channel's own Data Buffer"
D76- 91 02 STA (02),Y écrit A dans le "Channel's own Data Buffer"
D78- 38 SEC C = 1 flag "PUT" et
D79- 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 (s/p 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" corresp 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"

```
DD6- 4C 7E D6   JMP D67E     incrémente X et traite l'erreur n° X
```

Lit l'enregistrement suivant du fichier
(s/p 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.

```
DD9- 20 0E FD   JSR FD0E     reinitialise 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)
DDC- F0 F6      BEQ FDD4     si oui, "END OF FILE ERROR"
DDE- A0 00      LDY #00      sinon, copie cet octet (type de variable)
DE0- 91 06      STA (06),Y   dans le "General Buffer"
DE2- 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)
DE5- A0 01      LDY #01      copie cet octet à la suite
DE7- 91 06      STA (06),Y   dans le "General Buffer"
DE9- C8         INY          qui passe donc à #02
DEA- 84 F5      STY F5       et le copie en F5 (index dans le "General Buffer")
DEC- 85 F6      STA F6       copie aussi le dernier octet lu en F6 (longueur)
DEE- E6 F6      INC F6       prépare un compteur d'octets à copier
DF0- 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)
DF3- A4 F5      LDY F5       récupère l'index d'écriture en cours
DF5- E6 F5      INC F5       et incrémente F5 pour le tour suivant
DF7- 91 06      STA (06),Y   copie dans le "General Buffer" l'octet lu
DF9- C6 F6      DEC F6       décompte F6
DFB- D0 F3      BNE FDF0     reboucle en FDF0 tant qu'il en reste à copier
DFD- A0 00      LDY #00      vise le début du "General Buffer"
DFF- B1 06      LDA (06),Y   retourne avec les deux premiers octets du
E01- AA         TAX          "General Buffer" dans XA
E02- C8         INY          X = type de variable
E03- B1 06      LDA (06),Y   A = longueur de la variable
E05- 60         RTS
E06- EA         NOP
```

EXECUTION COMMANDE SEDORIC APPEND
(Fichiers "**S**")
(s/p 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 corresp 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 s/p F3CF.

Analyse de la syntaxe et saisie du paramètre

```
E07- 20 C0 FA   JSR FAC0     vérifie l'existence de FI, la validité du NL et si le fichier est déjà ouvert, reinitialise 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 Séquentiel)
E0A- A9 FF      LDA #FF      fin de fichier
E0C- 85 33      STA 33       place #FF en 33/34
E0E- 85 34      STA 34
E10- 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!
```

EXECUTION COMMANDE SEDORIC JUMP
(Fichiers **S**")
(s/p 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 s/p 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, reinitialise 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 reinitialise 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)
(s/p 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 s/p 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)

E5A- 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.
E5C- A2 0C LDX #0C pour copier 12 octets (nom et extension)
E5E- CA DEX indexés de 11 à 0
E5F- 30 22 BMI FE83 si fini, continue en FE83
E61- BD 91 C0 LDA C091,X lit un octet du nom de fichier source
E64- 9D 29 C0 STA C029,X écrit cet octet dans BUFNOM
E67- BC 9E C0 LDY C09E,X lit l'octet corresp du fichier cible
E6A- C9 3F CMP #3F l'octet source est-il un "?"?
E6C- F0 08 BEQ FE76 si oui, continue en FE76 avec C = 1
E6E- C0 3F CPY #3F l'octet cible est-il un "?"?
E70- D0 EC BNE FE5E sinon, reprend en FE5E (ni dans source, ni dans cible)
E72- 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)
E74- D0 E8 BNE FE5E reboucle en FE5E, (forcé car F2 non nul)
E76- 66 F4 ROR F4 force le b7 de F4 à 1 (donc dans le cas ou un "?" a été trouvé dans la source)
E78- 24 16 BIT 16 teste si b6 de 16 est à 1 (COPYM)
E7A- 70 E2 BVS FE5E si oui, reboucle en FE5E ("?" autorisés dans source pour COPYM)
E7C- C0 3F CPY #3F sinon, l'octet cible est-il aussi un "?"?
E7E- F0 DE BEQ FE5E si oui, reboucle en FE5E ("?" homologue requis est présent)
E80- 4C AC D5 JMP D5AC sinon, "INVALID FILE NAME ERROR"
E83- 24 F2 BIT F2 teste si le b7 de F2 est à 0 (pas de "?" présent dans la cible sans homologue dans la source)
E85- 10 0C BPL FE93 si oui, termine en FE93, sinon...
E87- A2 0C LDX #0C pour tester 12 octets qui doivent tous être des "?"
E89- BD 9D C0 LDA C09D,X lit un octet du nom de fichier cible
E8C- C9 3F CMP #3F est-ce un "?"?
E8E- D0 F0 BNE FE80 sinon, "INVALID FILE NAME ERROR"
E90- CA DEX octet précédent
E91- D0 F6 BNE FE89 reboucle en FE89 s'il en reste
E93- 58 CLI autorise les interruptions
E94- 60 RTS

EXECUTION COMMANDE SEDORIC LTYPE
(Fichiers "**S**")
(s/p 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 s/p F3CF.

Entrée de la commande LTYPE
E95- 20 C5 E7 JSR E7C5 PR SET et enchaîne avec la commande TYPE

EXECUTION COMMANDE SEDORIC TYPE
(Fichiers "**S**")
(s/p FE98-FEDF)

Rappel de la syntaxe
TYPE NL

Permet d'afficher, intégralrment 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 s/p 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, reinitialise 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 corresp, 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 pause jusqu'à ce qu'une touche soit pressée
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 reinitialise 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 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

EXECUTION COMMANDE SEDORIC BUILD
(Fichiers "**S**")
(s/p 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 s/p 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

EE0- 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**")
EE3- 20 00 FF JSR FF00 initialise un premier tampon de 218 octets forcés à #00 dans le "General Buffer"
EE6- 20 3D FF JSR FF3D s/p D845 "Prendre un caractère au clavier"
EE9- A4 F2 LDY F2 récupère Y, pointeur dans le tampon
EEB- C9 03 CMP #03 le caractère saisi est-il un CTRL/C?
EED- 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
EEF- C9 0D CMP #0D le caractère saisi est-il un RETURN?
EF1- D0 05 BNE FEF8 sinon, saute les deux instructions suivantes
EF3- 20 1B FF JSR FF1B si oui, met CR dans tampon, sauve celui-ci si plein
EF6- A9 0A LDA #0A prend un LF dans A pour l'ajouter à la suite du CR
EF8- 20 1B FF JSR FF1B met le caractère dans le tampon, sauve celui-ci si plein
EFB- 84 F2 STY F2 sauvegarde le pointeur Y dans F2
EFD- 4C E6 FE JMP FEE6 et reboucle en FEE6

Initialise un premier enregistrement de 216 octets forcés à #00 dans le "General Buffer"

F00- 20 A8 F4 JSR F4A8 place l'adresse du début du "Channel Buffer" corresp 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
F03- A9 80 LDA #80 type d'enregistrement "chaîne alphanumérique"
F05- 91 06 STA (06),Y place #80 dans le premier octet du tampon
F07- C8 INY
F08- A9 D8 LDA #D8 longueur de l'enregistrement (216 octets)
F0A- 91 06 STA (06),Y place cette valeur en 2ème position du tampon
F0C- A9 00 LDA #00 pour remise à zéro de la chaîne alphanumérique
F0E- C8 INY vise le 1er octet de la chaîne
F0F- 84 F2 STY F2 garde Y dans F2 index dans la chaîne
F11- 91 06 STA (06),Y force à 0 les 216 octets suivants
F13- C8 INY
F14- C0 DA CPY #DA teste si Y atteint #DA (218 = 216 + les 2 premiers)
F16- D0 F9 BNE FF11 reboucle tant qu'il en reste
F18- A0 02 LDY #02 vise le début de la chaîne
F1A- 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

F1B- 91 06 STA (06),Y place dans le tampon le caractère saisi
F1D- 20 2A D6 JSR D62A XAFCAR affiche ce caractère
F20- C8 INY vise la place suivante dans le tampon
F21- C0 DA CPY #DA la fin du tampon est-elle atteinte?
F23- 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

F25- 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 prédéfinie (218 octets).
F28- A9 FF LDA #FF marqueur de fin de fichier
F2A- 20 CC FD JSR FDCC lit l'index Y du "Channel's own Data Buffer"
F2D- 91 02 STA (02),Y place #FF à l'adresse 02/03 + Y
F2F- 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
F32- A0 02 LDY #02 reprend Y = 2 (vise le début du tampon)
F34- 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

F37- 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

Saisit un caractère au clavier et revient avec ce caractère dans A
(ce s/p est aussi appelé par la commande TYPE)

FF3D- 20 45 D8 JSR D845 prendre un caractère au clavier
FF40- 10 FB BPL FF3D reboucle tant qu'aucune touche n'a été pressée
FF42- 60 RTS

*** TABLE DES VECTEURS SYSTEME ***
(#FF43-#FFC6)

FF43- 4C 36 ED JMP **ED36** **XLINPU** appel à la routine LINPUT. TRAV0 contient la longueur de la chaîne à saisir, au retour TRAV2 contient le mode de sortie et #D0-#D1 donne l'adresse de début de la chaîne

FF46- 4C 98 D3 JMP **D398** **XCRGET** lit le caractère suivant à TXTPTR et le convertit en MAJUSCULE (identique à CHRGET du BASIC, mais en plus convertit les minuscules en MAJUSCULES)

FF49- 4C 9E D3 JMP **D39E** **XCRGOT** relit le caractère à TXTPTR et le convertit en MAJUSCULE (identique au CHRGOT du BASIC, mais en plus convertit les minuscules en MAJUSCULES)

FF4C- 4C 4F D4 JMP **D44F** **XNF** lit 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+1ème 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" corresp 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 fichier** selon X = POSNMX, POSNMP et POSNMS, VSALO0, VSALO1, DESALO
FF73- 4C E5 E0 JMP **E0E5** **XLOADA** charge le programme 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, piste A sur le bitmap courant

F88- 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")

F8B- 4C C0 DB JMP **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 1er secteur descripteur dans PSDESC (C05C/5D), le nombre de descripteurs utilisés dans NSDESC (C05E) et 1er descripteur en place

F8E- 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

F91- 4C A5 DB JMP **DBA5** **cherche** POSNMX de **1ère place libre** dans le directory
F94- 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)

F97- 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
F9A- 4C 2D DB JMP **DB2D** vérifie que la disquette en place est bien une disquette Sédoric, 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

F9D- 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

FA0- 4C FE DA JMP **DAFE** Charge dans BUF3 le sect pointé par POSNMP et POSNMS puis XCABU
FA3- 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

FA6- 4C CE DA JMP **DACE** **XVBUF1** rempli de 0 le BUFFER1
FA9- 4C A4 DA JMP **DAA4** **XSVSEC** écrit un secteur selon DRIVE, PISTE, SECTEUR et RWBUF
FAC- 4C 9E DA JMP **DA9E** **XSAY** sauve le secteur visé par RWBUF au secteur Y de la piste A
FAF- 4C 91 DA JMP **DA91** **XSBUF1** sauve BUF1 à la piste A et le secteur Y
FB2- 4C 82 DA JMP **DA82** **XSCAT** sauve le secteur de catalogue contenu dans BUF3, selon POSNMP et POSNMS
FB5- 4C 8A DA JMP **DA8A** **XSMAP** sauve le secteur de bitmap sur la disquette
FB8- 4C 73 DA JMP **DA73** **XPRSEC** lit un secteur selon DRIVE, PISTE, SECTEUR et RWBUF
FBB- 4C 6D DA JMP **DA6D** **XPAY** charge dans RWBUF le secteur Y de piste A
FBE- 4C 5D DA JMP **DA5D** **XPBUF1** charge dans le BUFFER1 le secteur dont la piste est contenue dans A et le secteur dans Y
FC1- 4C 4C DA JMP **DA4C** **XPMAP** prend secteur de bitmap dans BUF2, vérifie le format
FC4- 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

FC7-	53 45 44 4F 52 49 43 20 31 2E 30 20 70 61 72 20	SEDORIC 1.0 par
FD7-	46 2E 42 52 4F 43 48 45 20 65 74 20	F.BROCHE et
FE3-	44 2E 53 45 42 42 41 47	D.SEBBAG
FEB-	28 63 29 20 31 39 38 35 20 45 55 52 45 4B 41	(c) 1985 EUREKA
FFA-	**21 D1**	Vecteur NMI en D121
FFC-	**10 23**	Vecteur RESET en 2310 (non valide)
FFE-	**A5 D0**	Vecteur IRQ en D0A5

Sédoric V2.0 GB

Toute la table des vecteurs système, qui était utilisée par certains programmes écrits en langage machine, ainsi que la plus grande partie du copyright final ont été supprimées (soit 183 octets) et remplacées par une série de sous-programmes utilisés pour réaliser diverses adaptations:

Adaptation de XPMAP (en DA4C, prend la bitmap dans BUF2)
Initialise pour première page de bitmap

F43-	**A9 14**	LDA #14	piste où se trouve la bitmap
F45-	**A0 02**	LDY #02	secteur de la première page de bitmap
F47-	**84 2F**	STX 2F	b7 = 0 flag "première page de bitmap active"
F49-	**60**	RTS	

Adaptation de la commande INIT (en C482 dans la banque n°6)
La commande INIT appelle un micro sous-programme, ajouté dans la zone F638/F63D qui contenait auparavant des NOP, afin d'insérer un appel au nouveau sous-programme FF4A qui permet de sauver le 2ème secteur de bitmap.

638-	**8E 15 31**	STX 3115	pour remplacer STX écrasé en C5AE/C5B0 banque n°6
63B-	**4C 4A FF**	JMP FF4A	autre micro sous-programme qui consiste à écrire la deuxième page de bitmap située dans

BUF2 sur la disquette:

FF4A-	**A0 03**	LDY #03	pour 3ème secteur de la piste 20
FF4C-	**4C 8B DC**	JMP DC8B	lui-même modifié en:
DC8B-	**A9 14**	LDA #14	pour la piste 20
DC8D-	**4C 8E DA**	JMP DA8E	qui reprend le cours normal de XSMAP et sauve BUF2 sur la disquette dans le 3ème secteur de la piste 20

Adaptation de XSMAP (en DA8A, sauve la bitmap sur la disquette)

Un JMP FF4F placé au début de la routine XSMAP entraine le remplacement de celle-ci par la routine suivante, qui écrit la 2ème page de bitmap sur la disquette et charge ensuite la 1ère page dans BUF2:

FF4F-	**18**	CLC	flag entrée de "Sauve la bitmap sur disquette"
FF50-	**24 38**	BIT 38	continue en FF52
FF51-	**38**	SEC	flag entrée de "Sauve 1ère et charge 2ème"
FF52-	**48**	PHA	
FF53-	**98**	TYA	
FF54-	**48**	PHA	empile A et Y
FF55-	**AD 01 C0**	LDA C001	
FF58-	**48**	PHA	
FF59-	**AD 02 C0**	LDA COO2	
FF5C-	**48**	PHA	empile PISTE et SECTEUR
FF5D-	**A2 06**	LDX #06	
FF5F-	**BD 02 C2**	LDA C202,X	empile les 7 octets du BUF2 de C202 à C208
FF62-	**48**	PHA	(nombre de secteurs libres, nombre de fichiers,
FF63-	**CA**	DEX	nombre de pistes par face, nombre de secteurs par
FF64-	**10 F9**	BPL FF5F	piste et nombre de secteurs de directory).

Les informations correctes de la page active (1ère ou 2ème) seront toujours copiées sur l'autre page de la bitmap (2ème ou 1ère). En effet, ces informations sont mises à jour continuellement.

FF66-	**B0 08**	BCS FF70	si entré en FF51, continue en FF70 pour écrire la 1ère page de bitmap sur la disquette et charger la seconde

Ecrit la 2ème page de bitmap sur la disquette et charge la 1ère

FF68-	**20 4A FF**	JSR FF4A	sinon sauve BUF2 dans le 3ème secteur de la piste 20
FF6B-	**20 4C DA**	JSR DA4C	force le b7 de 2F à zéro (flag "1er secteur de bitmap en place dans BUF2") et prend le 2ème secteur de la piste 20 dans BUF2
FF6E-	**F0 0C**	BEQ FF7C	suite forcée en FF7C car Z = 1 après XPMAP

Ecrit la 1ère page de bitmap sur la disquette et charge la 2ème

FF70-	**86 2F**	STX 2F	force le b7 de 2F à 1 car X = FF (2ème page active)
FF72-	**20 89 DC**	JSR DC89	sauve BUF2 dans le 2ème secteur de la piste 20
FF75-	**A9 14**	LDA #14	indexe la piste n°20
FF77-	**A0 03**	LDY #03	et le secteur n°3
FF79-	**20 50 DA**	JSR DA50	prend dans BUF2 le 2ème secteur de bitmap

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.

FF7C-	**A2 00**	LDX #00	
FF7E-	**68**	PLA	récupère les 7 octets précédemment empilés
FF7F-	**9D 02 C2**	STA C202,X	et les copie dans BUF2 de C202 à C208
FF82-	**E8**	INX	(nombre de secteurs libres, nombre de fichiers,
FF83-	**E0 07**	CPX #07	nombre de pistes par face, nombre de secteurs par
FF85-	**90 F7**	BCC FF7E	piste et nombre de secteurs de directory)
FF87-	**68**	PLA	
FF88-	**8D 02 C0**	STA C002	
FF8B-	**68**	PLA	
FF8C-	**8D 01 C0**	STA C001	restaure SECTEUR et PISTE
FF8F-	**68**	PLA	
FF90-	**A8**	TAY	
FF91-	**68**	PLA	restaure Y et A
FF92-	**38**	SEC	force C à 1
FF93-	**60**	RTS	et retourne

Adaptation du sous-programme DC7D "Cherche un secteur libre"

La routine DC7D a été remplacée par le nouveau sous-programme FF94:

FF94-	**A2 02**	LDX #00	X pointe au début de la liste des secteurs
FF96-	**BD 10 C2**	LDA C210,X	teste l'octet visé par X dans la liste des secteurs

F99- **D0 11** BNE FFAC si présence de secteurs libres, branche en FFAC
F9B- **E8** INX sinon, vise l'octet suivant
F9C- **E0 F0** CPX #F0 la valeur maximale de X est-elle atteinte?
F9E- **D0 F6** BNE FF96 sinon, reboucle en FF96 (même page de bitmap)

La fin de la page courante est atteinte

FA0- **24 2F** BIT 2F si oui, teste si 1er secteur de bitmap est présent
FA2- **10 03** BPL FFA7 si présent, saute l'instruction suivante
FA4- **4C 78 DC** JMP DC78 sinon (les 2 pages ont été examinées), "DISK FULL"

La 1ère page était active, on la sauve, on charge la 2ème et on l'examine

FA7- **20 51 FF** JSR FF51 sauve le 1er secteur de bitmap et charge le 2ème
FAA- **B0 E8** BCS FF94 reprise forcée en FF94 car revient de FF51 avec C = 1
Un secteur libre a été trouvé, on le reserve

FAC- **AD 02 C2** LDA C202 un octet "mappant" un secteur libre a été trouvé
FAF- **D0 03** BNE FFB4
FB1- **CE 03 C2** DEC C203
FB4- **CE 03 C2** DEC C202 décrémente le nombre de secteurs libres
FB7- **24 2F** BIT 2F teste si on est sur le 2ème secteur de bitmap
FB9- **30 03** BMI FFBE si oui, saute l'instruction suivante
FBB- **4C 90 DC** JMP DC90 si le 1er secteur de bitmap est présent dans BUF2, reprend le cours normal de la routine "Cherche un secteur libre" en DC90

S/p spécial pour réserver un secteur dans la 2ème page de bitmap

FBE- **A9 60** LDA #60 sinon, place un RTS en DCA8 à la fin du sous-programme "Inverse
FC0- **8D A8 DC** STA DCA8 le bit corresp et met à jour la bitmap, c'est à dire inhibe le passage au sous-programme suivant "Calcule le nombre de secteurs du début de la disquette jusqu'au secteur trouvé"
FC3- **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
FC6- **A9 A9** LDA #A9 restaure la valeur originale de l'octet DCA8
FC8- **8D A8 DC** STA DCA8 qui avait été écrasée par un RTS
FCB- **8A** TXA nombre d'octets situés avant l'octet modifié
FCC- **A2 00** LDX #00 HH de ce nombre d'octets
FCE- **18** CLC on ajoute les #F0 octets déjà utilisés
FCF- **69 F0** ADC #F0 dans le 1er secteur de la bitmap
FD1- **90 01** BCC FFD4 si pas de retenue, continue en FFD4
FD3- **E8** INX sinon reporte cette retenue
FD4- **86 F3** STX F3 dans F3 (HH du nombre d'octets)
FD6- **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", puis des n° de piste A et de secteur Y corresp (s/p DCBD)

Adaptation du sous-programme DCD6
"A quel bit de la bitmap correspond le secteur AY à libérer"

Le sous-programme DCD6 comporte une série de petites routines. Trois instructions (ROR, TAX et SEC) de la fin d'une de ces routines ("Calcule la position de l'octet à modifier dans la bitmap" en DD0B/DD0D) a été écrasée par un JMP FFD9. Voici le listing de cette routine complémentaire:

FD9- **6A** ROR remplace le ROR écrasé en DD0B A est l'octet de la bitmap qu'il faut modifier
FDA- **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
FDC- **D0 04** BNE FFE2 si c'est le cas, saute les 2 instructions suivantes
FDE- **C9 F0** CMP #F0 teste si A < #F0 c'est à dire si l'octet à modifier est sur le 1er secteur de bitmap
FE0- **90 0F** BCC FFF1 si oui, continue en FFF1

Le secteur libre est dans la 2ème page de bitmap

FE2- **24 2F** BIT 2F teste si le b7 de 2F est à 1, c'est à dire si le 2ème secteur de bitmap est présent dans BUF2
FE4- **30 03** BMI FFE9 si oui, saute l'instruction suivante
FE6- **20 51 FF** JSR FF51 sauve le 1er en place dans BUF2 et charge le 2ème
FE9- **38** SEC pour faire une soustraction
FEA- **E9 F0** SBC #F0 calcule l'octet du 2ème qui sera modifié
FEC- **AA** TAX remplace le TAX écrasé en DD0C
est l'octet du 2ème secteur de bitmap qu'il faut modifier
FED- **38** SEC remplace le SEC écrasé en DD0E
FEE- **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 1ère page de bitmap

FFF1-	**24 2F**	BIT 2F	teste si le 1er secteur de bitmap est présent dans BUF2
FFF3-	**10 F7**	BPL FFEC	si oui, reprend en FFEC
FFF5-	**20 4F FF**	JSR FF4F	sinon, sauve le 2ème secteur de bitmap et charge le 1er
FFF8-	**B0 F2**	BCS FFEC	et reprend en FFEC (C mis à 1 par s/p FF4F)

Le reste de la dernière page du Sédoric (FFFA à FFFF) n'a pas été mofifié.

*** ANNEXES ***

ANNEXE A

Rappel de la structure des disquettes sédoric

Le Sédoric occupe 102 secteurs sur une disquette MASTER en deux groupes. Le premier groupe se trouve au début de la disquette et occupe 94 secteurs à partir du secteur n°1 de la piste n°0. Le deuxième groupe se trouve à la piste n°20 et occupe les secteurs n°1, 2, 3, 4, 7, 10, 13 et 16.

Sur une disquette SLAVE, le Sédoric occupe 8 secteurs en deux groupes. Le premier groupe se trouve au début de la disquette et occupe 8 secteurs à partir du secteur n°1 de la piste n°0. Ces huit secteurs ont identiques aux secteurs corresp d'une disquette MASTER, sauf le 23ème octet du 2ème secteur qui contient #00 (Master) ou #01 (Slave). Le deuxième groupe se trouve à la piste n°20 et occupe les secteurs n°1, 2, 3, 4, 7, 10, 13 et 16. Ces huit secteurs ont identiques aux secteurs corresp d'une disquette MASTER, sauf le secteur de bitmap (2ème secteur de la piste 20) dont les octets n°#02/#03 indiquent un nombre de secteurs libres différents et l'octet n°#0A qui contient #00 (Master) ou #01 (Slave). La carte des secteurs occupés (bitmap) est bien sûr également différente! Rappel: l'utilisation d'une disquette Slave nécessite la présence du Sédoric en RAMOV. De plus, ce type de disquette ne permet pas d'utiliser des commandes nécessitant le chargement d'une banque interchangeable. Sinon, il n'y a pas de différence.

Lors d'un INIT, les 95 (!) premiers secteurs de la disquette master sont chargés en RAM (de #3000 à 8F00) et ceci sans considération pour la syntaxe de INIT, ce qui représente une perte de temps quand il s'agit de formater un disque SLAVE où seuls les 8 premiers secteurs sont utilisés. Après certains ajustements, 94 (Master) ou 8 (Slave) secteurs sont recopiés sur la nouvelle disquette. Il est donc prudent de reprendre la disquette Master d'origine si on veut éviter l'accumulation des erreurs.

Les 3 premiers secteurs contiennent n° de version, boot et copyright et sont listés ci-après (que de perte de place!).

Les 61 suivants sont structurés comme un fichier: 1 secteur de descripteur suivi de 60 secteurs de code qui, lors du boot, sont copiés en RAM (de #1400 à #4FFF), puis en RAMOV (de C400 à FFFF). Ce fichier n'apparaît pas au directory.

Les 30 secteurs suivants représentent les 6 banques interchangeables et sont structurés en 6 fichiers de 5 secteurs: 1 secteur de descripteur suivi de 4 secteurs de code qui sont copiés en RAMOV (de C400 à C7FF) lors de l'appel de certaines commandes. Ces fichiers n'apparaissent pas au directory.

Pour récupérer ces fichiers cachés, il suffit de partir d'une disquette master vierge formatée en 16 secteurs par piste, de créer une serie de vrais fichiers à l'aide des commandes suivantes: SAVE"NOYAU.SED",A#1400,E#4FFF SAVE"BANQUE1.SED",A#C400,E#C7FF etc idem pour BANQUE2.SED, BANQUE3.SED, BANQUE4.SED, BANQUE5.SED ET BANQUE6.SED. Puis à l'aide d'un éditeur de secteur (BDDISK par exemple), il faut remplacer les coordonnées des descripteurs de ces fichiers dans le secteur 4 de la piste 20 (1ersecteur de catalogue) par les coordonnées des descripteurs des fichiers cachés. Les coordonnées des descripteurs se trouvent aux 13ème et 14ème octets de chaque ligne de catalogue. Il faut y écrire les coordonnées suivantes: 0004, 0401, 0406, 040B, 0410, 0505 et 050A pour une disquette master formatée en 16 secteurs par piste (voir plus loin le tableau décrivant l'emplacement de Sédoric sur une disquette master formatée en 16 secteurs par piste). Si l'on voulait pouvoir utiliser normalement cette disquette, il faudrait poursuivre les mises à jour (directory et bitmap), mais en l'état, il est déjà possible de copier ces 7 fichiers sur une autre disquette non triturée et de les exploiter à souhait.

Dump du premier secteur de disquette Master ou Slave

```
     0  1  2  3  4  5  6  7  8  9  A  B  C  D  E  F  0123456789ABCDEF
000- 01 00 00 00 00 00 00 00 20 20 20 20 20 20 20 20  ........
010- 00 00 03 00 00 00 01 00 53 45 44 4F 52 49 43 20  ........ SEDORIC
020- 20 20 20 20 20 20 20 20 20 20 20 20 20 20 20 20
030- 20 20 20 20 20 20 20 20 20 20 20 20 20 20 20 20
040- 53 45 44 4F 52 49 43 20 56 31 2E 30 30 36 20 30  SEDORIC V1.006 0
050- 31 2F 30 31 2F 38 36 20 20 20 20 20 20 20 20 20  1/01/86
060- 20 20 20 20 20 20 20 20 20 20 20 20 20 20 20 20
070- 20 20 20 20 20 20 20 20 20 20 20 20 20 20 20 20
080 à 00FF idem uniquement des #20... ce n'est pas rentable!
```

Le secteur n°1 de la piste n°0 du Sédoric version 2.0 GB comporte 66 octets différents dûs au message de copyright qui devient:

SEDORIC V2.0 08/11/91CRLF
Upgraded by Ray McLaughlin to allowCRLF
80 track double sided drives.

Dump du deuxième secteur de disquette Master ou Slave

```
     0  1  2  3  4  5  6  7  8  9  A  B  C  D  E  F  0123456789ABCDEF
000- 00 00 FF 00 D0 9F D0 9F 02 B9 01 00 FF 00 00 B9
010- E4 B9 00 00 E6 12 00 78 A9 7F 8D 0E 03 A9 10 A0       01 si Slave
020- 07 8D 6B 02 8C 6C 02 A9 86 8D 14 03 A9 BA A0 B9
030- 20 1A 00 A9 84 8D 14 03 A2 02 BD FD CC 9D F7 CC
040- CA 10 F7 A2 37 A0 80 A9 00 18 79 00 C9 C8 D0 F9
050- EE 37 B9 CA D0 F3 A2 04 A8 F0 08 AD 01 B9 A8 D0
060- 02 A2 3C 84 00 A9 7B A0 B9 8D FE FF 8C FF FF A9
070- 05 8D 12 03 A9 85 8D 14 03 A9 88 8D 10 03 A0 00
080- 58 AD 18 03 30 FB AD 13 03 99 00 C4 C8 4C 6C B9
090- A9 84 8D 14 03 68 68 68 AD 10 03 29 1C D0 D5 EE
0A0- 76 B9 EE 12 03 CA F0 1F AD 12 03 CD 00 B9 D0 C1
0B0- A9 58 8D 10 03 A0 03 88 D0 FD AD 10 03 4A B0 FA
0C0- A9 01 8D 12 03 D0 AA A9 C0 8D 0E 03 4C 00 C4 0C
0D0- 11 53 45 44 4F 52 49 43 20 56 31 2E 30 0A 0D 60  .SEDORIC V1.0..`
0E0- 20 31 39 38 35 20 4F 52 49 43 20 49 4E 54 45 52   1985 ORIC INTER
0F0- 4E 41 54 49 4F 4E 41 4C 0D 0A 00 00 00 00 00 00  NATIONAL........
```

Le secteur n°2 de la piste n°0 du Sédoric version 2.0 GB comporte 1 octet différent dûs au message de copyright qui devient:

SEDORIC V2.0

Désassemblage du deuxième secteur de disquette master

Cette routine est probablement mise en jeu lors du BOOT. Elle semble charger le Sédoric en RAMOV. Il faudrait connnaître la signification des registres d'I/O du contrôleur de disquette pour pouvoir en comprendre les détails.

017-	78	SEI	interdit les interruptions
018-	A9 7F	LDA #7F	A = 0111 1111
01A-	8D 0E 03	STA 030E	b7 de 030E à 0 pour interdire les interruptions
01D-	A9 10	LDA #10	
01F-	A0 07	LDY #07	
021-	8D 6B 02	STA 026B	PAPER = #10 (noir)
024-	8C 6C 02	STY 026C	INK = 07 (blanche)
027-	A9 86	LDA #86	
029-	8D 14 03	STA 0314	#86 I/O contrôleur de disquette
02C-	A9 BA	LDA #BA	
02E-	A0 B9	LDY #B9	AY = #BAB9
030-	20 1A 00	JSR 001A	afficher la chaîne pointée par AY
033-	A9 84	LDA #84	
035-	8D 14 03	STA 0314	#84 I/O contrôleur de disquette
038-	A2 02	LDX #02	pour copie de 3 octets de CCFD/CCFF en CCF7/CCF9
03A-	BD FD CC	LDA CCFD,X	lit un caractère de "COM" (extension par défaut)
03D-	9D F7 CC	STA CCF7,X	et le copie comme extension courante
040-	CA	DEX	caractère précédant
041-	10 F7	BPL 003A	reboucle en 003A tant qu'il y en a à copier
	Temporise?		
043-	A2 37	LDX #37	
045-	A0 80	LDY #80	

```
0047-  A9 00       LDA #00
0049-  18          CLC
004A-  79 00 C9    ADC C900,Y     calcule A = A + contenu de C900 + Y
004D-  C8          INY            indexe le suivant
004E-  D0 F9       BNE 0049       et reboucle en 0049 tant que Y n'est pas nul
0050-  EE 37 B9    INC B937       incrémente B937 lorsque Y passe par zéro
0053-  CA          DEX            décrémente l'index X
0054-  D0 F3       BNE 0049       et reboucle en 0049 tant que X n'est pas nul
0056-  A2 04       LDX #04
0058-  A8          TAY            teste si A est nul
0059-  F0 08       BEQ 0063       si oui, continue en 0063 avec Y = #00
005B-  AD 01 B9    LDA B901       sinon, A = B901
005E-  A8          TAY            teste si A est différent de zéro
005F-  D0 02       BNE 0063       si oui, continue en 0063 avec Y <> #00
0061-  A2 3C       LDX #3C
0063-  84 00       STY 00         écrit Y en 00
0065-  A9 7B       LDA #7B
0067-  A0 B9       LDY #B9        AY = #B97B
0069-  8D FE FF    STA FFFE
006C-  8C FF FF    STY FFFF       FFFE/FFFF = #B97B
006F-  A9 05       LDA #05
0071-  8D 12 03    STA 0312       #05 I/O contrôleur de disquette
0074-  A9 85       LDA #85
0076-  8D 14 03    STA 0314       #85 I/O contrôleur de disquette
0079-  A9 88       LDA #88
007B-  8D 10 03    STA 0310       #88 I/O contrôleur de disquette
007E-  A0 00       LDY #00        index pour écriture
0080-  58          CLI            autorise les interruptions
0081-  AD 18 03    LDA 0318       teste Ready du contrôleur de disquette
0084-  30 FB       BMI 0081       reboucle en 0081 tant que b7 n'est pas à 1
0086-  AD 13 03    LDA 0313       lecture du registre data contrôleur de disquette
0089-  99 00 C4    STA C400,Y     écriture à partir de C400
008C-  C8          INY            indexe la position suivante
008D-  4C 6C B9    JMP B96C       suite en B96C
0090-  A9 84       LDA #84
0092-  8D 14 03    STA 0314       #84 I/O contrôleur de disquette
0095-  68          PLA
0096-  68          PLA            élimine 3 octets de la pile
0097-  68          PLA
0098-  AD 10 03    LDA 0310       lit octet en 0310 commande contrôleur de disquette
009B-  29 1C       AND #1C        0001 1100 force à 0 tous les bits sauf b2 b3 b4
009D-  D0 D5       BNE 0074       reboucle en 0074 si le résultat n'est pas nul
009F-  EE 76 B9    INC B976       incrémente B976
00A2-  EE 12 03    INC 0312       incrémente 0312 contrôleur de disquette
00A5-  CA          DEX            décrément X
00A6-  F0 1F       BEQ 00C7       continue en 00C7 lorsque X devient nul
00A8-  AD 12 03    LDA 0312       lit octet en 0312 contrôleur de disquette
00AB-  CD 00 B9    CMP B900       teste s'il est différent du contenu de B900
00AE-  D0 C1       BNE 0071       si oui, reboucle en 0071
00B0-  A9 58       LDA #58
00B2-  8D 10 03    STA 0310       #58 I/O commande contrôleur de disquette
00B5-  A0 03       LDY #03        pour temporisation
00B7-  88          DEY            décrémente Y
00B8-  D0 FD       BNE 00B7       et reboucle en 00B7 jusqu'à ce qu'il soit nul
00BA-  AD 10 03    LDA 0310       lit octet en 0310 commande contrôleur de disquette
00BD-  4A          LSR            teste si le b0 de l'octet lu en 0310 est à 1
00BE-  B0 FA       BCS 00BA       si oui, reboucle jusqu'à ce qu'il passe à 0
00C0-  A9 01       LDA #01
00C2-  8D 12 03    STA 0312       #01 I/O contrôleur de disquette
00C5-  D0 AA       BNE 0071       reprise forcée en 0071
00C7-  A9 C0       LDA #C0        1100 0000
00C9-  8D 0E 03    STA 030E       autorise les interruptions T1
00CC-  4C 00 C4    JMP C400       et continue en C400 (initialisation Sédoric)
```

Dump du troisième secteur de disquette master

```
      0  1  2  3  4  5  6  7  8  9  A  B  C  D  E  F  0123456789ABCDEF
0000- 00 00 02 53 59 53 54 45 4D 44 4F 53 01 00 02 00  ...SYSTEMDOS....
0010- 02 00 00 42 4F 4F 54 55 50 43 4F 4D 00 00 00 00  ...BOOTUPCOM....
0020- 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
0030- 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
```

```
040- 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
050- 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
060- 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
070- 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
080 à 00FF idem uniquement des zéros... ce n'est pas rentable!
```

Exemple de secteur 1 de la piste #14 (20):

```
     0  1  2  3  4  5  6  7  8  9  A  B  C  D  E  F  0123456789ABCDEF
100- 2A 2A 2A 00 40 64 00 0A 00 94 41 6E 64 72 7B 20  ***.@d....André
110- 43 68 7B 72 61 6D 79 20 7A 7A 7A 7A 20 90 50 52  Chéramy zzzz .PR
120- 49 4E 54 22 42 6F 6E 6A 6F 75 72 2C 20 76 6F 69  INT"Bonjour, voi
130- 63 69 20 6C 65 20 53 7B 64 6F 72 69 63 20 6E 6F  ci le Sédoric no
140- 75 76 65 61 75 21 22 3A 50 49 4E 47 3A 50 52 49  uveau!":PING:PRI
150- 4E 54 22 53 61 6C 75 74 21 22 00 00 00 00 00 00  NT"Salut!"......
160 à C1FF uniquement des zéros...
```

Le Secteur Système (secteur 1 de la piste 20) est structuré ainsi:

100-	octets n°00/03	table des drives (contient le nombre de pistes)
104-	octets n°04	type de clavier (b6=1 si ACCENT SET et b7=1 si AZERTY)
105-	octets n°05/06	départ de RENUM (ici #0064 = 100)
107-	octets n°07/08	"pas" de RENUM (ici #0000A = 10)
109-	octets n°09/1D	nom de la disquette (ici #94 papier bleu, #90 noir)
11E-	octets n°1E/59	INIST, instructions exécutées au demarrage (60 octets)
15A-	octets n°5A/FF	non utilisés (#00) (l'INIST aurait pû être plus long)

Exemple de secteur 2 de la piste #14 (20):

```
     0  1  2  3  4  5  6  7  8  9  A  B  C  D  E  F
200- FF 00 6A 04 0C 00 2A 10 01 AA 00 00 00 00 00 00  ................
210- 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
220- 00 00 00 00 00 00 00 00 00 C0 FF FF FF FF FF FF  ................
230- FF FF FF FF FF FF FF FF B0 6D FF FF FF FF FF FF  ................
240- FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF  ................
250- FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF  ................
260- FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF  ................
270- FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF  ................
280- FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF  ................
290- FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF  ................
2A0- FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF  ................
2B0- FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF  ................
2C0- FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF  ................
2D0- FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF  ................
2E0- FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF  ................
2F0- FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF  ................
```

Le Secteur de Bitmap (secteur 2 de la piste 20) est structuré ainsi:

200-	octets n°00	contient toujours #FF
201-	octets n°01	contient toujours #00
202-	octets n°02/03	nombre de secteurs libres (ici #046A = 1130)
204-	octets n°04/05	nombre de fichiers (ici #000C = 12)
206-	octets n°06	nombre de pistes/face (ici #2A = 42)
207-	octets n°07	nombre de secteurs/piste (ici #10 = 16)
208-	octets n°08	nombre de secteurs de directory (ici #01 = 1)
209-	octets n°0	copie de l'octet n°06 dont on a ajusté le b7 à 0 si simple face ou à 1 si double face (en principe, car hélas c'est "la" bogue)
20A-	octets n°0A	si Master, #01 si Slave ou #47 ("G") si Games
20B-	octets n°0B/0F	contient toujours #00 (non utilisés)
210-	octets n°10/FF	Bitmap: chaque bit représente un secteur. Ce secteur est libre si le bit corresp est à 1 ou occupé s'il est à 0. Les bits de chaque octet sont lus de droite à gauche (sens b0 -> b7), mais les octets sont lus de gauche à droite (sens n°#10 -> n°#FF).

Par exemple, on voit ici que la plupart des secteurs sont libres (octets à #FF, c'est à dire à 1111 1111). Mais les octets n°38 et 39 indiquent #B0 et #6D. Il s'agit des 16 secteurs de la piste n°20. Les 8 premiers secteurs sont répertoriés par l'octet n°38 et on a #B0 = 1011 0000 qui indique que les secteurs n°1 à 4 sont occupés (les bits b0 à b3 sont à 0), les secteurs n°5 et 6 sont libres (les bits b4 et b5 sont à 1) le secteur n°7 est occupé (bit b6 à 0) et le secteur n°8 est libre (bit b7 à 1). Les 8 derniers octets sont répertoriés par l'octet n°39 où l'on a #6D = 0110 1101 qui indique que les secteurs n°10, 13 et 16 sont occupés, alors que les octets n°9, 11, 12, 14 et 15 sont libres. Cet exemple était simplifié par le fait que cette disquette étant formatée à 16 secteurs par piste, chaque piste est représentée par une paire d'octets.

Certains secteurs sont déjà occupés au départ, ce sont:

-Les secteurs situés au début de la disquette et dont le nombre varie selon qu'il s'agit d'une disquette Master (94), Slave (8) ou de type "G" (17) (Shortsed initialisé par GAMEINIT)
-Le Secteur Système (secteur 1 de la piste 20) voir ci-dessus
-Le Secteur Bitmap (secteur 2 de la piste 20) voir ci-dessus
-Le secteur 3 de la piste 20 qui ne contient que des #00 (non utilisés)
-Les secteurs de catalogues (secteurs 4, 7, 10, 13 et 16 de la piste 20) qui sont marqués "occupés" (réservés) mais contiennent des #00 tant qu'ils ne sont pas réellement utilisés

Exemple de Directory pour l'étude des différentes sortes de déscripteurs

```
Drive A (Master)              XX/XX/XX

E         .DOC 259   F         .DOC 514
D         .DOC 255   A         .DOC   3
B         .DOC   4   C         .DOC   2
G         .DOC   9   PETIFICHA.DSC   2
PETIFICHB.DSC   2   PETIFICHC.DSC   2
GROSFICHD.DSC   4   GROSFICHE.DSC   4
PETIFICHM.DSC   4   GROSFICHM.DSC   7

*255 sectors free (D/42/17), 14 Files
```

Le fichier G.DOC est formé des fichiers A, B et C mergés. Le fichier F.DOC est formé des fichiers D et E mergés. Dans les 2 cas, la taille résultante est la somme des tailles.

Ne pas tenir compte des fichiers "*.DSC". La page de catalogue corresp (piste #04 du secteur #14) est la suivante (les coordonnées du premier descripteur de chacun des fichiers qui nous intéressent sont en gras):

```
      0  1  2  3  4  5  6  7  8  9  A  B  C  D  E  F  0123456789ABCDEF
C300- 00 00 F0 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
C310- 45 20 20 20 20 20 20 20 20 44 4F 43 26 0A 03 41  E        DOC...@
C320- 46 20 20 20 20 20 20 20 20 44 4F 43 8B 0E 02 42  F        DOC...@
C330- 44 20 20 20 20 20 20 20 20 44 4F 43 0D 0E FF 40  D        DOC...@
C340- 41 20 20 20 20 20 20 20 20 44 4F 43 05 0A 03 40  A        DOC...@
C350- 42 20 20 20 20 20 20 20 20 44 4F 43 05 0D 04 40  B        DOC...@
C360- 43 20 20 20 20 20 20 20 20 44 4F 43 05 11 02 40  C        DOC...@
C370- 47 20 20 20 20 20 20 20 20 44 4F 43 06 02 09 40  G        DOC...@
C380- 50 45 54 49 46 49 43 48 41 44 53 43 06 0B 02 40  PETIFICHADSC...@
C390- 50 45 54 49 46 49 43 48 42 44 53 43 06 0D 02 40  PETIFICHBDSC...@
C3A0- 50 45 54 49 46 49 43 48 43 44 53 43 06 0F 02 40  PETIFICHCDSC...@
C3B0- 47 52 4F 53 46 49 43 48 44 44 53 43 07 04 04 40  GROSFICHDDSC...@
C3C0- 47 52 4F 53 46 49 43 48 45 44 53 43 07 08 04 40  GROSFICHEDSC...@
C3D0- 50 45 54 49 46 49 43 48 47 44 53 43 06 11 04 40  PETIFICHGDSC...@
C3E0- 47 52 4F 53 46 49 43 48 46 44 53 43 07 0C 07 40  GROSFICHFDSC...@
C3F0- 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
```

Un Secteur de Catalogue est structuré ainsi:

C300- octets n°00/01 coordonnées (piste/secteur) du catalogue suivant (ici un seul secteur de catalogue est en service, il n'y a donc pas de suivant)
C302- octets n°02 n° de l'octet de la 1ère entrée libre (#00 si plein)
C303- octets n°03/0F contient toujours #00 (inutilisés)
C310- octets n°10/FF 15 "entrées" de catalogue (une ligne de 16 octets par entrée)

Chaque "entrée" de catalogue est structurée ainsi:

Octets n°00 à 08 nom complété à droite par des espaces (#20)
octets n°09 à 0B extension (idem)
octet n°0C piste du descripteur
octet n°0D secteur du descripteur
octet n°0E nombre de secteurs du fichier (y compris le(s) descripteurs)
octet n°0F attribut de protection (b6=1, PROT si b7=1, UNPROT si b7=0) (#40 = 0100 0000 pour UNPROT et #C0 = 1100 0000 pour PROT). b0 à b5 = HH du nombre de secteurs = rarement utlisés, sauf pour les très gros fichiers mergés (comme ci-dessus F.DOC).

Exemples de descripteurs:

Le 1er 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 1er secteur descripteur)
(Le pointeur X est positionné sur ce #FF, et permet de lire la suite)
C103- octet n°03 (C101+X) type de fichier (voir manuel page 100)
C104- octets n°04/05 (C102+X et C103+X) adresse (normale) de début
(ou nombre de fiches pour un fichier à accès direct)

106- octets n°06/07 (C104+X et C105+X) adresse (normale) de fin
(ou longueur d'une fiche pour un fichier à accès direct)
108- octets n°08/09 (C106+X et C107+X) = adresse d'exécution si AUTO
10A- octets n°0A/0B (C108+X et C109+X) = nombre de secteurs à charger
10C- octets n°0C/FF (C100+Y et C101+Y) liste des coordonnées piste/secteur des secteurs à charger soit 122 paires de 2 octets. Si le nombre de secteurs à charger dépasse 122, lorsque Y atteint #00 (fin BUF1), il faut charger le descripteur suivant dont la structure est simplifiée:
100- octets n°00/01 "lien" (coordonnées du descripteur suivant)
102- octets n°02/FF (C100+Y et C101+Y) liste des coord piste/secteur des secteurs à charger (Y de #02 à #EF maxi), 127 paires de 2 octets. Si le nombre de secteurs à charger dépasse 122 + 127 = 249, il faut charger le descripteur suivant etc...

Examinons maintenant les descripteurs de chacun des fichiers qui nous intéressent:

Petit Fichier A (1 seul descripteur):

```
     0  1  2  3  4  5  6  7  8  9  A  B  C  D  E  F
100- 00 00 FF 40 00 10 FF 11 00 00 02 00 05 0B 05 0C  ................
110- 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
120 à C1FF idem uniquement des zéros...
```

Petit Fichier B (1 seul descripteur):

```
     0  1  2  3  4  5  6  7  8  9  A  B  C  D  E  F
100- 00 00 FF 40 00 20 FF 22 00 00 03 00 05 0E 05 0F  ................
110- 05 10 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
120 à C1FF idem uniquement des zéros...
```

Petit Fichier C (1 seul descripteur):

```
     0  1  2  3  4  5  6  7  8  9  A  B  C  D  E  F
100- 00 00 FF 40 00 30 FF 30 00 00 01 00 06 01 00 00  ................
110- 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
120 à C1FF idem uniquement des zéros...
```

Gros Fichier D (3 descripteurs) premier descripteur (les 2 premiers octets indiquent les coordonnées du descripteur suivant, soit le 9ème secteur de la piste #19):

```
     0  1  2  3  4  5  6  7  8  9  A  B  C  D  E  F
100- 15 09 FF 40 00 04 FF FF 00 00 FC 00 0D 0F 0D 10  ................
110- 0D 11 0E 01 0E 02 0E 03 0E 04 0E 05 0E 06 0E 07  ................
120- 0E 08 0E 09 0E 0A 0E 0B 0E 0C 0E 0D 0E 0E 0E 0F  ................
130- 0E 10 0E 11 0F 01 0F 02 0F 03 0F 04 0F 05 0F 06  ................
140- 0F 07 0F 08 0F 09 0F 0A 0F 0B 0F 0C 0F 0D 0F 0E  ................
150- 0F 0F 0F 10 0F 11 10 01 10 02 10 03 10 04 10 05  ................
160- 10 06 10 07 10 08 10 09 10 0A 10 0B 10 0C 10 0D  ................
170- 10 0E 10 0F 10 10 10 11 11 01 11 02 11 03 11 04  ................
180- 11 05 11 06 11 07 11 08 11 09 11 0A 11 0B 11 0C  ................
190- 11 0D 11 0E 11 0F 11 10 11 11 12 01 12 02 12 03  ................
1A0- 12 04 12 05 12 06 12 07 12 08 12 09 12 0A 12 0B  ................
1B0- 12 0C 12 0D 12 0E 12 0F 12 10 12 11 13 01 13 02  ................
1C0- 13 03 13 04 13 05 13 06 13 07 13 08 13 09 13 0A  ................
1D0- 13 0B 13 0C 13 0D 13 0E 13 0F 13 10 13 11 14 05  ................
1E0- 14 06 14 08 14 09 14 0B 14 0C 14 0E 14 0F 14 11  ................
1F0- 15 01 15 02 15 03 15 04 15 05 15 06 15 07 15 08  ................
```

Deuxième descripteur (situé dans le secteur #09 de la piste #15, les 2 premiers octets indiquent les coordonnées du descripteur suivant, soit 1er secteur de la piste #1D):

```
     0  1  2  3  4  5  6  7  8  9  A  B  C  D  E  F
100- 1D 01 15 0A 15 0B 15 0C 15 0D 15 0E 15 0F 15 10  ................
110- 15 11 16 01 16 02 16 03 16 04 16 05 16 06 16 07  ................
120- 16 08 16 09 16 0A 16 0B 16 0C 16 0D 16 0E 16 0F  ................
130- 16 10 16 11 17 01 17 02 17 03 17 04 17 05 17 06  ................
140- 17 07 17 08 17 09 17 0A 17 0B 17 0C 17 0D 17 0E  ................
150- 17 0F 17 10 17 11 18 01 18 02 18 03 18 04 18 05  ................
160- 18 06 18 07 18 08 18 09 18 0A 18 0B 18 0C 18 0D  ................
170- 18 0E 18 0F 18 10 18 11 19 01 19 02 19 03 19 04  ................
180- 19 05 19 06 19 07 19 08 19 09 19 0A 19 0B 19 0C  ................
190- 19 0D 19 0E 19 0F 19 10 19 11 1A 01 1A 02 1A 03  ................
1A0- 1A 04 1A 05 1A 06 1A 07 1A 08 1A 09 1A 0A 1A 0B  ................
1B0- 1A 0C 1A 0D 1A 0E 1A 0F 1A 10 1A 11 1B 01 1B 02  ................
1C0- 1B 03 1B 04 1B 05 1B 06 1B 07 1B 08 1B 09 1B 0A  ................
1D0- 1B 0B 1B 0C 1B 0D 1B 0E 1B 0F 1B 10 1B 11 1C 01  ................
1E0- 1C 02 1C 03 1C 04 1C 05 1C 06 1C 07 1C 08 1C 09  ................
1F0- 1C 0A 1C 0B 1C 0C 1C 0D 1C 0E 1C 0F 1C 10 1C 11  ................
```

Troisième et dernier descripteur (situé dans le secteur #01 de la piste #1D, les 2 premiers octets indiquent #0000, c'est à dire qu'il n'y a pas de descripteur suivant):

```
       0  1  2  3  4  5  6  7  8  9  A  B  C  D  E  F
C100- 00 00 1D 02 1D 03 1D 04 00 00 00 00 00 00 00 00  ................
C110- 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
C120 à C1FF idem uniquement des zéros...
```

Gros Fichier E (3 descripteurs) premier descripteur (les 2 premiers octets indiquent les coordonnées du descripteur suivant, soit le secteur n°#0E de la piste #03 de la 2ème face):

```
C100- 83 0E FF 40 00 00 FF FF 00 00 00 01 26 0B 26 0C  ................
C110- 26 0D 26 0E 26 0F 26 10 26 11 27 01 27 02 27 03  ................
C120- 27 04 27 05 27 06 27 07 27 08 27 09 27 0A 27 0B  ................
C130- 27 0C 27 0D 27 0E 27 0F 27 10 27 11 28 01 28 02  ................
C140- 28 03 28 04 28 05 28 06 28 07 28 08 28 09 28 0A  ................
C150- 28 0B 28 0C 28 0D 28 0E 28 0F 28 10 28 11 29 01  ................
C160- 29 02 29 03 29 04 29 05 29 06 29 07 29 08 29 09  ................
C170- 29 0A 29 0B 29 0C 29 0D 29 0E 29 0F 29 10 29 11  ................
C180- 80 01 80 02 80 03 80 04 80 05 80 06 80 07 80 08  ................
C190- 80 09 80 0A 80 0B 80 0C 80 0D 80 0E 80 0F 80 10  ................
C1A0- 80 11 81 01 81 02 81 03 81 04 81 05 81 06 81 07  ................
C1B0- 81 08 81 09 81 0A 81 0B 81 0C 81 0D 81 0E 81 0F  ................
C1C0- 81 10 81 11 82 01 82 02 82 03 82 04 82 05 82 06  ................
C1D0- 82 07 82 08 82 09 82 0A 82 0B 82 0C 82 0D 82 0E  ................
C1E0- 82 0F 82 10 82 11 83 01 83 02 83 03 83 04 83 05  ................
C1F0- 83 06 83 07 83 08 83 09 83 0A 83 0B 83 0C 83 0D  ................
```

2ème descripteur (situé dans le secteur #0E de la piste #03 de la 2ème face, les premiers octets indiquent que le descripteur suivant se trouve au 6ème secteur de la piste #0B de la 2ème face):

```
       0  1  2  3  4  5  6  7  8  9  A  B  C  D  E  F
C100- 8B 06 83 0F 83 10 83 11 84 01 84 02 84 03 84 04  ................
C110- 84 05 84 06 84 07 84 08 84 09 84 0A 84 0B 84 0C  ................
C120- 84 0D 84 0E 84 0F 84 10 84 11 85 01 85 02 85 03  ................
C130- 85 04 85 05 85 06 85 07 85 08 85 09 85 0A 85 0B  ................
C140- 85 0C 85 0D 85 0E 85 0F 85 10 85 11 86 01 86 02  ................
C150- 86 03 86 04 86 05 86 06 86 07 86 08 86 09 86 0A  ................
C160- 86 0B 86 0C 86 0D 86 0E 86 0F 86 10 86 11 87 01  ................
C170- 87 02 87 03 87 04 87 05 87 06 87 07 87 08 87 09  ................
C180- 87 0A 87 0B 87 0C 87 0D 87 0E 87 0F 87 10 87 11  ................
C190- 88 01 88 02 88 03 88 04 88 05 88 06 88 07 88 08  ................
C1A0- 88 09 88 0A 88 0B 88 0C 88 0D 88 0E 88 0F 88 10  ................
C1B0- 88 11 89 01 89 02 89 03 89 04 89 05 89 06 89 07  ................
C1C0- 89 08 89 09 89 0A 89 0B 89 0C 89 0D 89 0E 89 0F  ................
C1D0- 89 10 89 11 8A 01 8A 02 8A 03 8A 04 8A 05 8A 06  ................
C1E0- 8A 07 8A 08 8A 09 8A 0A 8A 0B 8A 0C 8A 0D 8A 0E  ................
C1F0- 8A 0F 8A 10 8A 11 8B 01 8B 02 8B 03 8B 04 8B 05  ................
```

Troisième et dernier descripteur (situé dans le 6ème secteur de la piste #0B de la 2ème face, les 2 premiers octets indiquent #0000, c'est à dire qu'il n'y a pas de descripteur suivant):

```
       0  1  2  3  4  5  6  7  8  9  A  B  C  D  E  F
C100- 00 00 8B 07 8B 08 8B 09 8B 0A 8B 0B 8B 0C 8B 0D  ................
C110- 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
C120 à C1FF idem uniquement des zéros...
```

Descripteurs des fichiers mergés

Pour les fichiers "mergés", le "lien" du dernier descripteur de chaque fichier indique les coordonnées du 1er 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 "mergé". 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.

Exemples de descripteurs de fichiers mergés

Petit Fichier G (3 descripteurs) (formé des 3 petits fichiers A, B et C mergés)

1er descripteur (fichier A), avec les coordonnées du descripteur du fichier B):

```
       0  1  2  3  4  5  6  7  8  9  A  B  C  D  E  F
C100- 06 05 FF 40 00 10 FF 11 00 00 02 00 06 03 06 04  ................
C110- 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
C120 à C1FF idem uniquement des zéros...
```

2ème descripteur (fichier B), avec les coordonnées du descripteur du fichier C):

```
     0  1  2  3  4  5  6  7  8  9  A  B  C  D  E  F
100- 06 09 FF 40 00 20 FF 22 00 00 03 00 06 06 06 07  ................
110- 06 08 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
120 à C1FF idem uniquement des zéros...
```

Troisième et dernier descripteur (fichier C):

```
     0  1  2  3  4  5  6  7  8  9  A  B  C  D  E  F
100- 00 00 FF 40 00 30 FF 30 00 00 01 00 06 0A 00 00  ................
110- 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
120 à C1FF idem uniquement des zéros...
```

Gros Fichier F (6 descripteurs) (formé des gros fichiers D et E mergés)

1er descripteur (fichier D), avec les coordonnées du 2ème descripteur de D):

```
     0  1  2  3  4  5  6  7  8  9  A  B  C  D  E  F
100- 93 01 FF 40 00 00 FF FF 00 00 00 01 8B 0F 8B 10  ................
110- 8B 11 8C 01 8C 02 8C 03 8C 04 8C 05 8C 06 8C 07  ................
120- 8C 08 8C 09 8C 0A 8C 0B 8C 0C 8C 0D 8C 0E 8C 0F  ................
130- 8C 10 8C 11 8D 01 8D 02 8D 03 8D 04 8D 05 8D 06  ................
140- 8D 07 8D 08 8D 09 8D 0A 8D 0B 8D 0C 8D 0D 8D 0E  ................
150- 8D 0F 8D 10 8D 11 8E 01 8E 02 8E 03 8E 04 8E 05  ................
160- 8E 06 8E 07 8E 08 8E 09 8E 0A 8E 0B 8E 0C 8E 0D  ................
170- 8E 0E 8E 0F 8E 10 8E 11 8F 01 8F 02 8F 03 8F 04  ................
180- 8F 05 8F 06 8F 07 8F 08 8F 09 8F 0A 8F 0B 8F 0C  ................
190- 8F 0D 8F 0E 8F 0F 8F 10 8F 11 90 01 90 02 90 03  ................
1A0- 90 04 90 05 90 06 90 07 90 08 90 09 90 0A 90 0B  ................
1B0- 90 0C 90 0D 90 0E 90 0F 90 10 90 11 91 01 91 02  ................
1C0- 91 03 91 04 91 05 91 06 91 07 91 08 91 09 91 0A  ................
1D0- 91 0B 91 0C 91 0D 91 0E 91 0F 91 10 91 11 92 01  ................
1E0- 92 02 92 03 92 04 92 05 92 06 92 07 92 08 92 09  ................
1F0- 92 0A 92 0B 92 0C 92 0D 92 0E 92 0F 92 10 92 11  ................
```

2ème descripteur (fichier D), avec les coordonnées du 3ème descripteur de D):

```
     0  1  2  3  4  5  6  7  8  9  A  B  C  D  E  F
100- 9A 0A 93 02 93 03 93 04 93 05 93 06 93 07 93 08  ................
110- 93 09 93 0A 93 0B 93 0C 93 0D 93 0E 93 0F 93 10  ................
120- 93 11 94 01 94 02 94 03 94 04 94 05 94 06 94 07  ................
130- 94 08 94 09 94 0A 94 0B 94 0C 94 0D 94 0E 94 0F  ................
140- 94 10 94 11 95 01 95 02 95 03 95 04 95 05 95 06  ................
150- 95 07 95 08 95 09 95 0A 95 0B 95 0C 95 0D 95 0E  ................
160- 95 0F 95 10 95 11 96 01 96 02 96 03 96 04 96 05  ................
170- 96 06 96 07 96 08 96 09 96 0A 96 0B 96 0C 96 0D  ................
180- 96 0E 96 0F 96 10 96 11 97 01 97 02 97 03 97 04  ................
190- 97 05 97 06 97 07 97 08 97 09 97 0A 97 0B 97 0C  ................
1A0- 97 0D 97 0E 97 0F 97 10 97 11 98 01 98 02 98 03  ................
1B0- 98 04 98 05 98 06 98 07 98 08 98 09 98 0A 98 0B  ................
1C0- 98 0C 98 0D 98 0E 98 0F 98 10 98 11 99 01 99 02  ................
1D0- 99 03 99 04 99 05 99 06 99 07 99 08 99 09 99 0A  ................
1E0- 99 0B 99 0C 99 0D 99 0E 99 0F 99 10 99 11 9A 01  ................
1F0- 9A 02 9A 03 9A 04 9A 05 9A 06 9A 07 9A 08 9A 09  ................
```

3ème descripteur (fichier D), avec les coordonnées du 1er du fichier E):

```
     0  1  2  3  4  5  6  7  8  9  A  B  C  D  E  F
100- 9B 01 9A 0B 9A 0C 9A 0D 9A 0E 9A 0F 9A 10 9A 11  ................
110- 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
120 à C1FF idem uniquement des zéros...
```

4ème descripteur (fichier E), avec les coordonnées du 2ème descripteur de E):

```
     0  1  2  3  4  5  6  7  8  9  A  B  C  D  E  F
100- A2 05 FF 40 00 04 FF FF 00 00 FC 00 9B 02 9B 03  ................
110- 9B 04 9B 05 9B 06 9B 07 9B 08 9B 09 9B 0A 9B 0B  ................
120- 9B 0C 9B 0D 9B 0E 9B 0F 9B 10 9B 11 9C 01 9C 02  ................
130- 9C 03 9C 04 9C 05 9C 06 9C 07 9C 08 9C 09 9C 0A  ................
140- 9C 0B 9C 0C 9C 0D 9C 0E 9C 0F 9C 10 9C 11 9D 01  ................
150- 9D 02 9D 03 9D 04 9D 05 9D 06 9D 07 9D 08 9D 09  ................
160- 9D 0A 9D 0B 9D 0C 9D 0D 9D 0E 9D 0F 9D 10 9D 11  ................
170- 9E 01 9E 02 9E 03 9E 04 9E 05 9E 06 9E 07 9E 08  ................
180- 9E 09 9E 0A 9E 0B 9E 0C 9E 0D 9E 0E 9E 0F 9E 10  ................
190- 9E 11 9F 01 9F 02 9F 03 9F 04 9F 05 9F 06 9F 07  ................
1A0- 9F 08 9F 09 9F 0A 9F 0B 9F 0C 9F 0D 9F 0E 9F 0F  ................
1B0- 9F 10 9F 11 A0 01 A0 02 A0 03 A0 04 A0 05 A0 06  ................
1C0- A0 07 A0 08 A0 09 A0 0A A0 0B A0 0C A0 0D A0 0E  ................
1D0- A0 0F A0 10 A0 11 A1 01 A1 02 A1 03 A1 04 A1 05  ................
1E0- A1 06 A1 07 A1 08 A1 09 A1 0A A1 0B A1 0C A1 0D  ................
1F0- A1 0E A1 0F A1 10 A1 11 A2 01 A2 02 A2 03 A2 04  ................
```

5ème descripteur (fichier E), avec les coordonnées du 3ème descripteur de E):

```
      0  1  2  3  4  5  6  7  8  9  A  B  C  D  E  F
C100- A9 0E A2 06 A2 07 A2 08 A2 09 A2 0A A2 0B A2 0C  ................
C110- A2 0D A2 0E A2 0F A2 10 A2 11 A3 01 A3 02 A3 03  ................
C120- A3 04 A3 05 A3 06 A3 07 A3 08 A3 09 A3 0A A3 0B  ................
C130- A3 0C A3 0D A3 0E A3 0F A3 10 A3 11 A4 01 A4 02  ................
C140- A4 03 A4 04 A4 05 A4 06 A4 07 A4 08 A4 09 A4 0A  ................
C150- A4 0B A4 0C A4 0D A4 0E A4 0F A4 10 A4 11 A5 01  ................
C160- A5 02 A5 03 A5 04 A5 05 A5 06 A5 07 A5 08 A5 09  ................
C170- A5 0A A5 0B A5 0C A5 0D A5 0E A5 0F A5 10 A5 11  ................
C180- A6 01 A6 02 A6 03 A6 04 A6 05 A6 06 A6 07 A6 08  ................
C190- A6 09 A6 0A A6 0B A6 0C A6 0D A6 0E A6 0F A6 10  ................
C1A0- A6 11 A7 01 A7 02 A7 03 A7 04 A7 05 A7 06 A7 07  ................
C1B0- A7 08 A7 09 A7 0A A7 0B A7 0C A7 0D A7 0E A7 0F  ................
C1C0- A7 10 A7 11 A8 01 A8 02 A8 03 A8 04 A8 05 A8 06  ................
C1D0- A8 07 A8 08 A8 09 A8 0A A8 0B A8 0C A8 0D A8 0E  ................
C1E0- A8 0F A8 10 A8 11 A9 01 A9 02 A9 03 A9 04 A9 05  ................
C1F0- A9 06 A9 07 A9 08 A9 09 A9 0A A9 0B A9 0C A9 0D  ................
```

6ème et dernier descripteur (fichier N, c'est le 3ème du fichier E):

```
      0  1  2  3  4  5  6  7  8  9  A  B  C  D  E  F
C100- 00 00 A9 0F A9 10 A9 11 00 00 00 00 00 00 00 00  ................
C110- 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
C120 à C1FF idem uniquement des zéros...
```

ANNEXE B

Sédoric V2.0 GB

Cette version est due à Ray McLaughlin. D'une part elle corrige certaines bogues, d'autre part elle permet d'utiliser des disquettes 3"1/2 double densité au maximum de leurs possibilités soit 720 kilo octets (Double face, 80 pistes de 18 secteurs) et même un peu plus (747 kilo octets avec 83 pistes de 18 secteurs). Mais, comme avec la version 1.006, le formatage en 19 secteurs par piste n'est pas fiable. Il faudrait essayer avec un lecteur de type HD (1.44 Mo), mais les disquettes sont chères pour un gain de capacité minime.

La grande nouveauté consiste à disposer d'un 2ème secteur de bitmap que Ray a placé dans le 3ème secteur de la piste 2O, qui était déjà réservé, mais inutilisé. On peut tirer un grand coup de chapeau à notre ami Ray, car il a fait du beau travail!

A l'usage, cette version 2.0 GB se révèle très pratique, même si l'on s'en tient à l'utilisation des lecteurs 3" et 5"1/4 (bogue double face corrigée). De plus, les lecteurs 3"1/2 étant bon marché (350F) et d'une bien meilleure qualité que les anciens lecteurs 3", il est recommandé de passer à ce format (disquettes à 3F au lieu de 21F) et de disposer enfin de 720 kilo octets sur Sédoric. La gestion des fichiers est très fiable et sans problème, jusqu'à 83 pistes de 18 secteurs. La commande DEL fonctionne sans bogue et nous l'avons triturée dans tous les sens!

Voici la liste des commandes et sous-programmes qui ont été modifiés:

1) TRACK (en C446 sur la banque 5) Inutile modification de la vérification du nombre total de secteurs par disquette qui doit être < 3840, mais ne peut jamais dépasser 3762!

2) INIT (en C404 sur la banque 6) Même inutile vérification, mise en place d'un 2ème secteur de bitmap et correction de la bogue de double face.

3) XPMAP (en DA4C, "Prend le secteur de bitmap dans BUF2") Adaptation pour fonctionner avec 2 secteurs de bitmap, nouveau s/p FF43 appelé en DA4C.

4) XSMAP (en DA8A, "Sauve le secteur de bitmap sur la disquette") Adaptation pour la même raison, remplacé par s/p DC80 placé au début de XSMAP. Ce s/p DC80 a été implanté dans un vide laissé par la modification du s/p DC7D (voir plus loin). Si le b7 de 2F est à 1, le 2ème secteur de bitmap présent dans BUF2 est sauvegardé par un JSR FF4F.

5) Le s/p DC7D "Cherche un secteur libre" a été remplacé par le s/p FF94.

6) La fin du s/p DCD6 "Calcule à quel bit et à quel octet de la bitmap correspond le secteur AY à libérer" a été remplacé par le s/p FFD9 tenant compte des 2 bitmaps.

7) La commande Sédoric ">" (en F5BA, "Affecte un champ à une variable") a aussi été modifiée en F5FE/F609.

8) Une petite série de NOP (F638/F63D) a été remplacée par un s/p utilisé par INIT pour insérer un appel au nouveau s/p FF4A qui permet de sauver la 2ème bitmap.

9) La table des vecteurs système (FF43/FFC6, qui n'était en fait pas utilisée) et une partie du copyright final (FFC7/FFF9) ont été supprimées et remplacées par des sous- programmes utilisés pour réaliser les adaptations indiquées ci-dessus:

- s/p FF43 utilisé pour XPMAP

- s/p FF4A utilisé pour INIT

- s/p FF4F utilisé pour XSMAP

- s/p FF51 utilisé pour XSMAP

- s/p FF94 utilisé par s/p "Cherche un secteur libre"

- s/p FFD9 utilisé par le s/p "A quel bit des bitmaps correspond le secteur AY à libérer"

Au total 595 bits diffèrent entre la version 1.006 et la version 2.0 GB. Ces différences incluent aussi 66 octets au secteur 1 de la piste 0 où le copyright "SEDORIC V1.006 01/01/86" a été changé en "SEDORIC V2.0 08/11/91 Upgraded by Ray McLaughlin to allow 80 track double sided drives."; 1 octet au secteur suivant (SEDORIC V2.0 au lieu de V1.0) et bien sûr la quasi-totalité du secteur 3 de la piste 20 qui ne contenait que des zéros. De plus l'octet en C5FE à été changé (#F1 devient #2D), mais la signification de ce changement m'échappe.

ANNEXE C

Exercices de passage ROM <--> RAMOV

En RAMOV, l'adresse C024 (ATMORI) contient la valeur #00 pour les machines équipées d'une ROM V1.0 et la valeur #80 pour celles qui sont équipées de la V1.1. En ROM, cette adresse contient la valeur #31 (ROM V1.0) ou la valeur #08 (ROM V1.1).

Allumez votre machine et tapez: ?PEEK(#C024), l'écran affiche alors 8 si ROM V1.1 ou 49 si ROM V1.0. Normalement, vous êtes donc sur la ROM.

L'accès à la RAMOV qui contient le Sédoric est prévu (heureusement!). Le "truc" est simple et indiqué dans le manuel (Annexe 8: passages RAM <-> ROM). Il suffit, dans le programme en langage machine de faire un JSR 04F2 pour accéder à la RAMOV, d'appeler le ou les sous-programmes voulus et de terminer par un autre JSR 04F2 pour revenir aux conditions normales, c'est à dire sur la ROM. Cette procédure n'affecte aucun registre (ou plutôt, ils sont sauvés puis restaurés).

Voici donc un petit exercice, tapez le programme qui suit. Il s'agit d'un chargeur de langage machine écrit en BASIC:

```
100 DATA #20, #F2, #04        :REM JSR 04F2
110 DATA #AD, #24, #C0        :REM LDA C024
120 DATA #8D, #0E, #98        :REM STA 980E
130 DATA #20, #F2, #04        :REM JSR 04F2
140 DATA #60, #EA             :REM RTS et NOP où sera mis le résultat
200 FOR K=#9801 TO #980E :REM on place ce programme de 9801 à 980E
210 READ V:POKE K,V
220 NEXT
```

Maintenant sauvegardez votre programme, puis faites un CALL#9801 suivi d'un ?PEEK(#980E) qui vous affichera 128 (soit #80) si votre ROM est une V1.1 ou 0 si c'est une version 1.0

Il faut encore noter que lorsqu'on exécute une commande Sédoric, on est sous RAMOV. Si cette commande concerne des manipulations de la mémoire, il est donc possible de consulter, voire d'altérer Sédoric lui-même. Ainsi pour modifier Sédoric, vous avez le choix entre 2 méthodes.

Vous pouvez travailler directement sur la disquette en utilisant un éditeur de secteurs du type BDDISK ou NIBBLE et en vous aidant des tableaux "Emplacement de Sédoric sur une disquette master" en annexe A pour avoir la correspondance entre l'adresse en RAMOV et les coordonnées piste/secteur.

Vous pouvez aussi plus simplement effectuer un SAVE de la zone à modifier, suivi d'un LOAD,A en mémoire basse, modifier cette zone à l'aide d'un moniteur/assembleur/desassembleur classique, la resauver (utilisez les adresses en mémoire basse), la recharger en RAMOV à l'aide d'un LOAD,A et enfin péréniser votre travail à l'aide d'un INIT qui recopiera sur disquette le code présent en RAMOV. La deuxième méthode est un peu plus longue que la première, mais beaucoup plus aisée et confortable.

Voici la liste des logiciels "moniteur/assembleur/desassembleur" qui ont été adaptés pour fonctinner avec le Sédoric, que nous avons testés et qui peuvent être obtenus au CEO (avec entre parenthèses l'auteur de l'adaptation, qu'ils en soient remerciés, nos excuses pour les oublis et omissions):

- Automon de André Chénière (D.Henninot),

- Hades de ERE Informatique (D.Henninot),

- Sédutil de F.Taraud (D.Henninot),

- Supmon et Supdes de J.P.Laurent (semble être le code originel) et

- Assembleur de Micrologic (François Launay).

Il serait pratique de créer une banque n°7 contenant un utilitaire de ce type.

ANNEXE D

Utilisation d'une commande Sédoric sans argument à partir d'un programme écrit en langage machine

Nous allons utiliser la routine 04F2 comme indiqué à l'annexe C.

Par exemple, pour exécuter la commande Sédoric OLD il suffit d'insérer dans votre programme LM la séquence: JSR 04F2 JSR E0AF JSR 04F2, simple non?

Tapez le petit programme qui suit:

```
100 DATA #20, #F2, #04          :REM JSR 04F2
110 DATA #20, #AF, #E0          :REM JSR E0AF
120 DATA #20, #F2, #04          :REM JSR 04F2
130 DATA #60                    :REM RTS
200 FOR K=#9801 TO #980A        :REM on place ce programme
210 READ V:POKE K,V             :REM de 9801 à 980A
220 NEXT
```

Sauvegardez, implantez avec un RUN, effacez le programme avec un NEW et restaurez le avec un CALL #9801 Un LIST vous persuadera que ça marche!

Si vous préférez utiliser un moniteur, par exemple Supmon, pas de problème, charger ce moniteur, tapez:

```
I 9801 20 F2 04 20 AF E0 20 F2 04 60 <RETURN>
F <RETURN>
10 REM ESSAI OLD <RETURN>
NEW
CALL #9801
LIST
```

ANNEXE E

Utilisation d'une routine en RAMOV à partir d'un programme écrit en langage machine

Tapez le petit programme qui suit:

```
100 DATA #48, #45, #4C, #4C, #4F, #20     :REM message "HELLO_
110 DATA #41, #4F, #44, #52, #45, #00     :REM ANDRE" terminé par zéro
120 DATA #20, #F2, #04                    :REM JSR #04F2
130 DATA #A9, #01                         :REM LDA #01
140 DATA #A0, #98                         :REM LDY #98
150 DATA #20, #37, #D6                    :REM JSR D637
160 DATA #20, #F2, #04                    :REM JSR #04F2
170 DATA #60                              :REM RTS
200 FOR K=#9801 TO #981A                  :REM on place ce programme
210 READ V:POKE K,V                       :REM de #9801 à #981A
220 NEXT
```

Sauvegardez ce chef d'oeuvre, RUN pour implanter, CALL #980D pour afficher "HELLO ANDRE". Vous avez utilisé le sous-programme XAFSTR situé en D637 en RAMOV. Ce sous-programme permet d'afficher toute chaîne d'adresse AY terminée par un zéro. Il y a encore des centaines de routines en RAMOV qui ne demandent qu'à être utilisées. Faites votre choix à l'aide de "Sédoric à nu"!

Si vous préférez utiliser un moniteur, par exemple Supmon, tapez:

```
T 9801 "HELLO ANDRE" <RETURN>
I 980C 00 20 F2 04 A9 01 A0 98 20 37 D6 20 F2 04 60 <RETURN>
F <RETURN>
```

CALL #980D affiche le message ou un autre, selon votre fantaisie, mais attention à l'adresse du CALL si la longueur du message est différente.

ANNEXE F

Utilisation d'une commande sédoric avec paramètres à partir d'un programme écrit en langage machine

Mais, diriez-vous comment faire avec une commande comportant des paramètres? Un embryon de solution est indiqué dans la banque zéro, lorsque Sédoric exécute les instructions de démarrage (INIST): il suffit de placer la ou les commandes Sédoric avec paramètres dans le TIB (tampon clavier), d'initialiser correctement TXTPTR et de faire appel à l'interpréteur en ROM. Cette méthode, très simple, a un inconvénient: on retourne au Ready.

Copie les instructions terminées par "fin de commandes" dans le TIB

```
100 DATA #44,#49,#52,#22,#2A,#2E,#42,#41,#53,#22,#00 :REM DIR"*.BAS"#00
110 DATA #A2, #0B          :REM LDX #0B          pour copier 12 octets
120 DATA #BD, #01, #98     :REM LDA 9801,X       lit les octets de 9801 à 980B
130 DATA #95, #35          :REM STA 35,X         et les copie dans le TIB de 35 à 3F
140 DATA #CA               :REM DEX              octet précédent (par la fin)
150 DATA #10, #F8          :REM BPL 120          et reboucle tant qu'il en reste
```

Exécute les instructions présentes dans le tampon clavier
avec l'interpréteur BASIC et retour au "Ready"

```
200 DATA #A2, #34          :REM LDX #34          XY pour ajuster TXTPTR à 0034
210 DATA #A0, #00          :REM LDY #00          adresse C4BD de l'interpréteur
220 DATA #20, #BD, #C4     :JSR C4BD             Atmos (prendre C4CD pour l'Oric-1)
```

Mise en place du programme

```
300 DATA #4C, #B5, #FA     :REM JMP FAB5         SHOOT (FA9B pour la ROM V1.0)
310 FOR K=#9801 TO #981F:READ V:POKE K,V: NEXT
CALL#980C <RETURN>
```

Affiche le catalogue des fichiers "*.BAS" et retourne au "Ready" sans SHOOT (et oui!).

Une autre méthode, préconisée par Denis Henninot, permet de retourner au point d'appel dans le programme appelant en langage machine. Denis utilise le moniteur Hadès, mais on peut aussi procéder avec un autre assembleur ou avec un chargeur BASIC comme ci-dessus. Sa méthode consiste à détourner le vecteur 1B/1C vers le programme appelant. Sur la ROM, l'affichage du message "Ready" se fait par un JSR 001A avec en 1B/1C l'adresse CCB0 qui est celle de la routine "Afficher la chaîne AY". Cette méthode simple de mise en oeuvre permet de bénéficier de la gestion des erreurs:

```
I <RETURN> pour inserer le texte source
1 ORG $9801
2 CMD NUL 'DIR"*.BAS"'          ;ou autre chaîne à exécuter (avec paramètres)
3 DEB LDX #$00                  ;transfert
4 >1      LDA CMD,X             ;de la (ou des)
5         STA $35,X             ;commande(s)
6         BEQ >2                ;et des éventuels paramètres
7         INX                   ;dans le
8         BNE <1                ;buffer clavier (35 à 84)
9 >2      LDA #RET              :LL de l'adresse de retour pour détournement du
10        STA $1B               ;vecteur 1B/1C (affichage du "Ready") vers RET
11        LDA /RET              :idem avec HH
12        STA $1C               ;
13        LDX #$34              ;pour ajuster TXTPTR juste avant le début de la commande CMD
14        LDY #$00              ;au début du tampon clavier
15        JMP $C4BD             ;continue à l'Interpréteur (C4CD si ROM V1.0)
16 RET    LDA #$B0
17        STA $1B
18        LDA #$CC              ;CCB0 (CBED si ROM V1.0) affichage du "Ready"
19        STA $1C
20        JSR $FAB5             ;FA9B si ROM V1.0
21        RTS
22        <RETURN>
```

A <RETURN> pour assembler

Hadès affiche: fin des labels $29B8

```
ORG $9801                ;début du s/p
CMD $9801                ;adresse de la commande
DEB $980C                ;adresse d'exécution du s/p
RET $9827                ;adresse de retour après passage sous Sédoric
FIN $9833                ;fin du s/p
```

B <RETURN> pour retourner au BASIC
SAVE"SP",A#9801,E#9833 <RETURN> pour sauver le s/p
CALL#980C <RETURN> Affiche le catalogue des fichiers "*.BAS", suivi d'un SHOOT et retour au "Ready"

Pour lire ou écrire un secteur

Voici un autre exemple, fourni par Denis Henninot, qui permet d'utiliser une routine Sédoric avec paramètres à partir d'un programme en langage machine:

```
ORG $9800
JSR $04F2                ;passage sur la RAMOV
LDA #$..                 ;n° du drive à utiliser (de 0 à 3)
STA $C000                ;que l'on place dans DRIVE
LDA #$..                 ;n° de la piste à lire ou à écrire
STA $C001                ;que l'on place dans PISTE
LDA #$..                 ;n° du secteur à lire ou à écrire
STA $C002                ;que l'on place dans SECTEUR
LDA #BUFFER              ;LL de l'adresse du tampon lecture/écriture
STA $C003                ;que l'on souhaite utiliser, par exemple 9900
LDA /BUFFER              ;idem HH
STA $C004                ;cette adresse est placée dans RWBUF
```

et l'une des deux instructions suivantes doit être insérée:

```
JSR $DA73                ;XPRSEC lit un secteur selon DRIVE, PISTE, SECTEUR et RWBUF
JSR $DAA4                ;XSVSEC écrit un secteur selon DRIVE, PISTE, SECTEUR et RWBUF
JSR $04F2                ;et enfin retour sur la ROM
```

Rechercher un secteur disponible sur la disquette

Encore une routine de Denis, très utile pour les amateurs de langage machine qui désire utiliser au mieux les avantages de Sédoric.

```
JSR $04F2                ;passage sur la RAMOV
LDA #$..                 ;n° du drive à utiliser (de 0 à 3)
STA $C000                ;que l'on place dans DRIVE
JSR $DA4C                ;XPMAP prend le secteur de bitmap dans BUF2
JSR $DC6C                ;XLIBSE cherche un secteur libre, revient avec coordonnées AY
STA $C001                ;que l'on place dans PISTE
STY $C002                ;et dans SECTEUR
JSR $DA8A                ;XSMAP sauve le secteur de bitmap sur la disquette
JSR $04F2                ;et enfin retour sur la ROM
```

Libère un secteur déjà occupé

Cette routine, qui est la contrepartie de la routine précédente, est elle aussi bien utile.

```
JSR $04F2                ;passage sur la RAMOV
LDA #$..                 ;n° du drive à utiliser (de 0 à 3)
STA $C000                ;que l'on place dans DRIVE
JSR $DA4C                ;XPMAP prend le secteur de bitmap dans BUF2
LDA $C001                ;PISTE du secteur à libérer
LDY $C002                ;SECTEUR à libérer
JSR $DD15                ;XDETSE libère le secteur Y de la piste A sur la bitmap
JSR $DA8A                ;XSMAP sauve le secteur de bitmap sur la disquette
JSR $04F2                ;et enfin retour sur la ROM
```

ANNEXE G

Commandes utilisées par la routine XRWDRV pour piloter le circuit intégré FD1783
(Controlleur du lecteur de disquette)

Selon Fabrice Broche, les commandes utilisées par la routine XRWDRV sont conformes à la notice du 1793 de Western Digital (5"1/4 mini floppy MFM controller/formatter). Cette notice indique qu'il existe 11 commandes qui peuvent être chargées seulement si le bit "Busy Status" est OFF (sauf pour la commande "Force Interrupt"). Le bit "Busy Status" est à 1 pendant l'exécution d'une commande et à 0 après la fin d'exécution. Le "Status Register" indique si l'exécution se termine avec ou sans erreur. Les 11 commandes sont divisées en 4 groupes (I à IV).

Groupe I: (déplacement de la tête)

-Restore (#08) positionne la tête sur la piste #00
-Seek (#18) positionne la tête et met à jour le "Track Register"
-Step (#28 ou #38) avance la tête d'une piste dans la même direction que précédemment, et met à jour (#38) ou non (#28) le "Track Register"
-Step in (#48 ou #58) idem vers les n° de pistes croissants
-Step out (#68 ou #78) idem en direction de la piste zéro
Les b0 et b1 (Stepping Motor Rate) des commandes du groupe I indiquent la vitesse de changement de piste. Avec la carte controlleur Microdisc (MFM, 1 MHz), ils sont à zéro, ce qui correspond à un changement de piste en 6 ms.
Le b2 (Track Number Verify Flag) est à zéro (pas de vérification).
Le b3 (Head Load Flag) est à 1 (Load at beginning).
Enfin le b4 (Track Update Flag) est soit à 0 (pas de mise à jour du "Track Register"), soit à 1 (mise à jour du "Track Register").

Groupe II: (lecture/écriture d'un secteur)

-Read Sector (#80 ou #88) lit un secteur sans tester le n° de face
-Read Sector (#90 ou #98) lit plusieurs secteurs sans tester le n° de face
-Write Sector (#A0 ou #A8) écrit un secteur sans tester le n° de face
-Write Sector (#B0 ou #B8) écrit plusieurs secteurs sans tester le n° de face
Avant d'envoyer une commande de lecture/écriture, la tête doit avoir été positionnée sur la bonne piste (commande du groupe I) et donc le "Track Register" contenir le n° de piste voulu. L'ordinateur doit encore mettre à jour le "Sector Register" avec le n° du secteur désiré.
Le b0 (Data Address Mark) est toujours à 0 et indique que les data commencent après un #FB.
Le b1 (Side Compare flag) est à 0 s'il ne faut pas et à 1 s'il faut comparer le n° de face lu dans le champ ID sur la disquette et le n° de face indiqué par b3 (voir plus loin).
Le b2 (15 ms Delay) est toujours à zéro (pas de délai).
Le b3 (Side Compare Flag) est à 0 (face n°0) ou à 1 (face n°1). En pratique #80 et #88, par exemple, donnent le même résultat puisque dans les 2 cas le b1 est à 0 (pas de comparaison).
Le b4 (Multiple Record Flag) est à 0 si un seul secteur ou à 1 si plusieurs secteurs doivent être lus ou écrits.

Groupe III:

-Read Address (#C0) le FD1793 lit le prochain champ ID, en assemble les 6 octets (n° de piste, n° de face, n° de secteur, taille du secteur, CRC1 et CRC2), les transfère dans le DR, génère un DRQ pour chaque octet, vérifie la validité et génère une CRC error si besoin.
-Read Track (#E0) tous les octets de gaps, en-têtes et data sont lus, assemblés et transférés dans le "Data Register" et un DRQ est généré pour chaque octet. Il n'y a pas de vérification de CRC, mais le "Lost Data Status Flag" peut éventuellement être mis à 1.
-Write Track (#F0) formate une piste. Toutes les informations à écrire sur la piste doivent être prêtes en mémoire. Il suffit alors de positionner la tête sur la piste à formater, puis d'envoyer la commande #F0. Attention, tous les octets de #00 à #F4 et #F8 à #FF sont écrits tels quels sur la piste, mais pas les octets de #F5 à #F7. Les #F5 sont convertits en #A1 et le générateur de CRC est initialisé. Les F6 sont convertits en #C2. Finalement, chaque #F7 génère 2 octets de CRC.

Groupe IV:

-Force Interrupt (#DX) commande utilisée pour terminer une commande de lecture/écriture multiple. Les bits b0 à b3 (représentés par un "X") sont positionnés selon diverses "Interupt Conditions". Apparemment Sédoric n'utilise pas les commandes #90, #98, #B0, #B8 et #DX. Voir la notice du FD1793 pour toute utilisation spéciale de la routine XRWDRV avec ces commandes.

ANNEXE H

Les bogues de sédoric

(Sans vouloir porter atteinte à ce système d'exploitation génial)

Problème de l'utilisation des minuscules

Le manuel indique (page 22) qu'il est possible de taper les commandes Sédoric en minuscules. Ceci n'est pas très pratique, puisqu'il faut continuer à entrer les commandes BASIC en majuscules, ce qui entraine une continuelle utilisation du CTRL/T. Enfin, il y a de nombreux problèmes avec l'utilisation des minuscules pour entrer les commandes Sédoric: ça ne marche pas à tous les coups (bogue).

Le nom des commandes Sédoric contenant un token BASIC a été bien géré (voir la table des mots-clés en C9DE/CBBA). Mais certaines commandes Sédoric exigent en outre un token BASIC pour satisfaire leur syntaxe et là, rien n'a été prévu. Cela ne prète pas à conséquence lorsqu'il s'agit de "-" (commande DELETE ou lecteur-), "<" (commandes RSET et LSET), ">" (commande NC > variable), "&" (commandes NL et -NL), "@" (commande LINPUT), ? et PRINT (commande EXT) mais c'est catastrophique pour: "AUTO" (commandes SAVE et STATUS), "TO" (commandes REN, BACKUP, COPY, CHANGE et FIELD), END (commande NUM), ELSE, GOTO et THEN (commande KEYIF), NEXT (commande RESUME), LPRINT (commande WIDTH), DEF (commande USER) et enfin la commande RESTORE.

La présence des octets corresp à ces caractères et tokens est demandée à TXTPTR par la routine D22E. Cette routine n'effectue bien sûr aucun contrôle ni aucune conversion. Elle est utilisée aussi pour détecter la présence de certaines options, mais pas de toutes. Par exemple, pour la commande USER, "DEF" et "O" doivent être tapés en majuscules, par contre "A", "X", "Y" et "P" sont acceptés en minuscule! Ces 4 dernières lettres sont lues par la routine D398 qui lit un caractère à TXTPTR avec conversion en majuscule.

Voici une liste des options qu'il faut absolument taper en majuscules: "S" (mais pas "A") pour la commande DKEY, "L" de la commande MERGE, "M" (mais pas "S") pour la commande SEEK, "O" (mais pas "A", "X", "Y" ou "P") de la commande USER, "D" (mais pas "S") de la commande TRACK.

Les options "S", "D" et "R" de la commande OPEN ne posent pas de problème (il faut seulement laisser un espace entre OPEN et D).

Les options qui sont précédées d'une virgule sont correctement traitées, c'est à dire converties en majuscule. C'est le cas de:

",C" et ",N" des commandes COPY, COPYO et COPYM

",O" de la commande FIELD

",S" et ",D" de la commande INIT

",C", ",E", ",J", ",K" et ",S" pour la commande LINPUT

",A", ",V", ",J" et ",N" des commandes LOAD et chargement direct

",A", ",E" et ",T" des commandes SAVE, SAVEO, SAVEM et SAVEU

",A" et ",T" de la commande STATUS.

La validation du drive indiqué après les commandes: BACKUP, DELBAK, DKEY, DNAME, DNUM, DSYS, DTRACK, INIST, INIT, OPEN D, PMAP, PUT, SMAP, SYSTEM et TAKE se passe sans problème grace à la routine E60D.

La validation d'un drive spécifié par un nom de fichier ambigu aprrès les commandes: COPY, DEL, DESTROY, DIR, LDIR, PROT, REN, SEARCH et UNPROT se passe sans problème grace à la routine D451.

Malgré tout, l'utilisation des minuscules, qui n'est pas sans risque, n'a rien de pratique et pourrait être purement et simplement supprimée. Cela libérerait 11 entrées dans la table des mots-clés Sédoric (en C9DE/CBBA), permettrait de donner un n° à ceux qui n'en ont pas (USING, UNPROT, VUSER, WIDTH, WINDOW et RESTORE) et de créer 5 nouveaux mots-clés Sédoric.

Problème de la mise à jour du flag Double face

La commande INIT est sévèrement boguée. Le paramètre ",D" provoque bien un formatage en Double face, mais l'indicateur de Double face (le b7 de l'octet n°#09 de la bitmap) n'est pas mis à jour, ainsi qu'en témoigne le directory, qui indique désespérément "S/" au lieu de "D/". Le nombre de face n'est correctement mis à jour que si on ne formate pas ! Cette bogue est très génante car elle se répercute sur d'autres commandes, notamment BACKUP. Cette bogue a été corrigée dans la version 2.0 GB.

Fautes d'orthographe dans les messages affichés:

"UNKNOW'N" (pour "UNKNOWN"),

"sectors free" (au lieu de "free sectors"),

"DISC" et "DISCS" (pour "DISK" et "DISKS"),

"Founds" (pour "Found"),

"LINES ALREADY EXISTS" (au lieu de "LINE ALREADY EXISTS")

Valeurs incorrectes:

#4E en C531c et en C550c (il faudrait #4F, commande CHANGE, longueur des chaînes).

#31 en D71B (il faudrait #32, le n° d'erreur utilisateur minimal est 50 et non 49).

#0C en E558 (il faudrait #0B, commande REN, comparaison des "?" de l'ancien nom et du nouveau nom: cette comparaison inclut un octet de trop).

#4F en E8A7 (il faudrait #50, commande TKEN, longueur de la chaîne, le manuel indique 79 caractères).

#02 en EA1B (il faudrait #03, commande EXT, la validité du troisième caractère de l'extension n'est pas vérifiée. Il est donc possible de mettre n'importe quoi comme troisième caractère. Mais attention quand même, ce n'est pas sans risque: une extention à "CO?" est acceptée, mais les fichiers "*.CO?" ne le seront pas).

#5F en F1D3 (il faudrait #5E, commande INIT, pour charger les 94 premiers secteurs de la disquette master et non les 95 premiers).

#63 en FBA4 (il faudrait #3F, c'est à dire 63 en décimal, cette bogue est très grave et empèche absolument l'utilisation de la commande CLOSE sans paramètre).

Code incorrect:

C5A4/C5A6 (résidu de mise au point mal digéré dans la commande CHANGE).

D16F (routine "Affiche le message 'DISP TYPE MISMATCH ERROR'", le LDA #A3 doit être remplacé par un LDX #A3).

D479/D47D (rempli BUFNOM de "?", or X n'est pas nul en entrée mais vaut #FF (sortie de la boucle D465/D469) donc le 1er "?" est écrit en C128 au lieu de C029! De plus au retour Z = 0 car le dernier DEX entraine X = #0B (non nul). Le BEQ suivant ne sert donc à rien.

D4FD/D505 (il y a un JSR D7BD, qui valide le drive demandé, de trop).

D801/D802 (la variable EO est inutilisée et semble être un résidu de mise au point).

D907/D927 (routine "Prendre un caractère au clavier": le s/p traitant des codes corresp aux mots-clés Sédoric est complètement bogué et ne marche pas. Il est clair qu'au dernier moment les codes Sédoric ont été remplacés par des #00 dans le tableau C800/C87F!).

DE80/DE87 (il y a un LDY 0269 de trop, il s'agit d'une bogue mineure due à la fatigue du programmeur!).

E1F8/E20A (routine XLOADA, bogue évitée de justesse pour l'option ",V" car Z = 1 par chance, il aurait été mieux en E1F8 de brancher en E20A).
E38E/E39B (commande DIR, bogue bénigne: il aurait fallu un BNE E39B).

E68C/E6BB (commande STATUS, incompatibilité entre les options "T" et "AUTO", absence de vérification: ces options ne doivent pas être utilisées conjointement).

E6C1/E6CF (commande STATUS, absence de vérification du type de fichier avant de forcer le flag AUTO).

E8CE/E8D5 (commande TKEN, 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).

E9DE/E9EC (commande RESUME, lors du rajustement de TXTPTR sur l'instruction ayant causé l'erreur la routine cherche un octet ":" cette procédure est dangeureuse car la valeur #3A peut ainsi être le HH d'un n° de ligne et alors bonjour le plantage...).

EB25/EB90 (la commande NUM n'effectue 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: attention à ce que vous tapez!).

ED3C/ED51 (c'est la plus grosse bogue de LINPUT, qui provoque un grave problème de gestion du curseur. Ceci apparaît lorsque la longueur de la chaîne demandée dépasse 38 caractères. Les facéties du curseur sont quasi-imprévisibles et rendent impossible l'utilisation, à coup sûr, du paramètre "@x,y,").

F210/F233 (commande WINDOW: la vérification du type de fichier intervient après son chargement en RAMOV, en cas d'erreur, Sédoric a toutes les chances d'être écrasé!).

F325/F327/F3CA (encore la commande WINDOW, cette fois c'est l'existence du tableau WI$ qui n'est pas vérifiée).

F3F3/F424 (gestion de fichiers, routine de vérification de l'existence et création éventuelle de **FI**, cette vérification est plus que cavalière et peut conduire à toutes les catastrophes de plus cette routine, qui est l'une des plus utilisées, se propose à tout moment de créer le pseudo-tableau **FI**, même si elle n'est pas appelée par une commande OPEN).

F526/F525 (gestion de fichiers, routine de gestion des champs, il est possible d'avoir le même "nom_de_champ(index)" dans plusieurs fichiers, cependant lors d'un transfert de ou vers un champ, grâce aux commandes LSET ou RSET, seul le 1er "nom_de_champ(index)" est accessible. Ceci est dû au fait que Sédoric 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 Sédoric: remplacer BVC F548 (50 20) par BVC F54B (50 23) en F526).

FA62/FA63 (gestion de fichiers, routine d'extension de **FI**, le LDY A1 qui se trouve là est absolument inutile, toutefois, il ne crée aucun problème).

FC48/FC4A (commande FIELD, analyse de syntaxe, il faudrait remplacer le JSR D22E par un JSR D22C, sinon le séparateur de paramètres peut être non seulement une virgule, mais n'importe quoi!).

Version 2.0 GB: en C4D5e le saut à "ILLEGAL QUANTITY ERROR" appelé depuis C4A9 et C4AD ne marche plus car il manque le JMP.

Bogues dans le manuel Sédoric

Le nombre maximum de secteurs par disquette n'est pas de 1200 comme indiqué page 37, mais de 1919.

Bogue page 62 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.

Commande ERR, en mode direct le n° de ligne est #FFFF (soit 65535 et non 65635 comme indiqué dans le manuel page 59).

Erreur dans le manuel page 60: 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.

Commande USER, le manuel comporte une erreur page 70 dans la syntaxe indiquée: la virgule est indispensable devant DEF (les exemples donnés sont eux corrects), sinon Sédoric considère alors qu'il s'agit d'un paramètre utilisateur.

Erreur dans le manuel concernant ce que retourne la commande INSTR: renvoie IN = 0 si la chaîne à examiner est vide ou si la chaîne recherchée 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.

LINPUT: l'option ",S" contrairement à ce qui est indiqué dans le manuel page 61, interdit de sortir avec les flèches de déplacement (mode par défaut).

Toujours LINPUT page 61, parmi les caractères de contrôles 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), le manuel oublie le CTRL/D.

Dans le préambule concernant la gestion de fichiers, page 76, ainsi que pour les commande OPEN S, OPEN R et OPEN D, pages 77, 82 et 88, le manuel passe sous silence le pseudo-tableau **FI** de type entier qui est réservé et, beaucoup plus grave, oublie d'indiquer qu'il est interdit de créer un tableau avec la commande DIM dès qu'un OPEN a été utilisé et ceci tant que tous les fichiers ouverts ne sont pas fermés avec CLOSE.

Commande &(), 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.

ANNEXE I

Tables et figures

ANNEXE J

Mots clés sédoric

ANNEXE K

Table des matières

Emplacement de Sédoric sur une disquette Master 16 secteurs/piste

	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16		
0	copyright et boot			Desc	C400	C500	C600	C700	C800	C900	CA00	CB00	CC00	CD00	CE00	CF00		
1	D000	D100	D200	D300	D400	D500	D600	D700	D800	D900	DA00	DB00	DC00	DD00	DE00	DF00		
2	E000	E100	E200	E300	E400	E500	E600	E700	E800	E900	EA00	EB00	EC00	ED00	EE00	EF00		
3	F000	F100	F200	F300	F400	F500	F600	F700	F800	F900	FA00	FB00	FC00	FD00	FE00	FF00		
4	BK1	C400	C500	C600	C700	BK2	C400	C500	C600	C700	BK3	C400	C500	C600	C700	BK4		
5	C400	C500	C600	C700	BK5	C400	C500	C600	C700	BK6	C400	C500	C600	C700				

Emplacement de Sédoric sur une disquette Master 17 secteurs/piste

	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	
0	copyright et boot			Desc	C400	C500	C600	C700	C800	C900	CA00	CB00	CC00	CD00	CE00	CF00	D000	
1	D100	D200	D300	D400	D500	D600	D700	D800	D900	DA00	DB00	DC00	DD00	DE00	DF00	E000	E100	
2	E200	E300	E400	E500	E600	E700	E800	E900	EA00	EB00	EC00	ED00	EE00	EF00	F000	F100	F200	
3	F300	F400	F500	F600	F700	F800	F900	FA00	FB00	FC00	FD00	FE00	FF00	BK1	C400	C500	C600	
4	C700	BK2	C400	C500	C600	C700	BK3	C400	C500	C600	C700	BK4	C400	C500	C600	C700	BK5	
5	C400	C500	C600	C700	BK6	C400	C500	C600	C700									

Emplacement de Sédoric sur une disquette Master 18 secteurs/piste

	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18
0	copyright et boot			Desc	C400	C500	C600	C700	C800	C900	CA00	CB00	CC00	CD00	CE00	CF00	D000	D100
1	D200	D300	D400	D500	D600	D700	D800	D900	DA00	DB00	DC00	DD00	DE00	DF00	E000	E100	E200	E300
2	E400	E500	E600	E700	E800	E900	EA00	EB00	EC00	ED00	EE00	EF00	F000	F100	F200	F300	F400	F500
3	F600	F700	F800	F900	FA00	FB00	FC00	FD00	FE00	FF00	BK1	C400	C500	C600	C700	BK2	C400	C500
4	C600	C700	BK3	C400	C500	C600	C700	BK4	C400	C500	C600	C700	BK5	C400	C500	C600	C700	BK60
5	C400	C500	C600	C700														

Desc = "descripteur" du noyau Sédoric; BK1 à BK6 "descripteurs" des banques interchangeables 1 à 6.